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。

Application Note

RENESAS

SH7000 シリーズ　CPU 編

アプリケーションマニュアル

ルネサスSuperH RISC engine

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　1994.09

はじめに

SH7000シリーズは、RISCタイプのCPUにより、高性能な演算処理を実現し、システム構成に必要な周辺機能を集積すると同時に、携帯機器応用に不可欠な低消費電力を同時に実現する新世代シングルチップRISCマイコンです。

SH7000シリーズCPUは、RISC(Reduced instruction set computer)タイプの命令セットをもっており、基本命令は1命令1ステート(1システムクロックサイクル)で動作するので、命令実行速度が飛躍的に向上しています。また内部32ビット構成を採っていてデータ処理能力を強化しています。

このアプリケーションノートCPU編は、SH7000シリーズCPUのアーキテクチャの詳細およびSH7000シリーズCPUの特長を生かしたソフトウェア応用例を掲載しています。アドレッシングモード、命令の機能等を知るために使ってください。

目　　次

1. CPUアーキテクチャ

第1章　目次

1．1　特長

SH7000シリーズのCPUは、RISC(Reduced instruction set computer)タイプの命令セットを持っており、基本命令は1命令1ステート(1システムクロックサイクル)で動作するので、命令実行速度が飛躍的に向上しています。また内部32ビット構成を採用しておりデータ処理能力を強化しています。

SH7000シリーズCPUの特長を表1．1に示します。

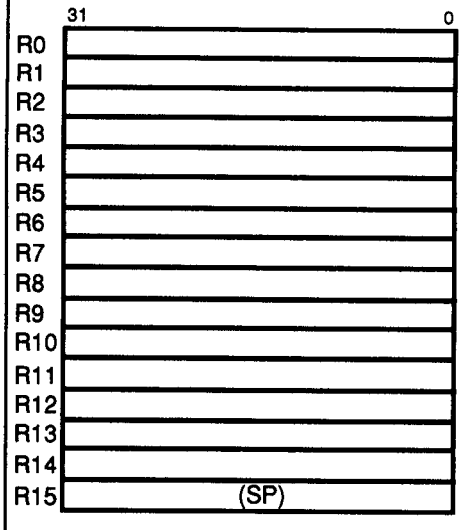
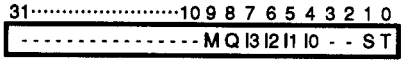
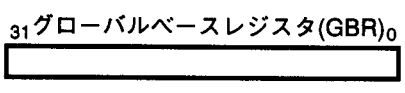
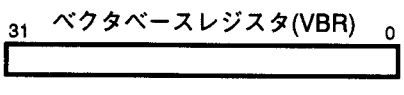
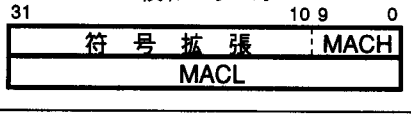
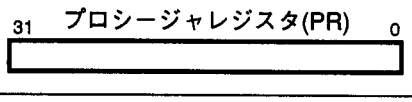
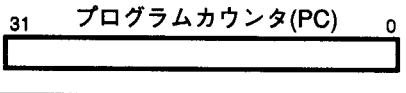
表1．1　SH7000シリーズCPUの特長

項目	特長
アーキテクチャ	・ 日立オリジナルアーキテクチャ ・ 内部32ビット構成
汎用レジスタマシン	・ 汎用レジスタ　32ビット×16本 ・ コントロールレジスタ　32ビット×3本 ・ システムレジスタ　32ビット×4本
命令セット	・ RISCタイプの命令セット - 命令長は16ビット固定長、これによるコード効率の向上 - ロード・ストア・アーキテクチャ(基本演算はレジスタ間で実行) - 無条件分岐命令を遅延分岐方式とすることで、分岐時のパイプラインの乱れを軽減 - C言語指向の命令セット
命令実行時間	・ 基本命令は1命令/1ステート(20MHz動作時、1ステートは50ns)
アドレス空間	・ アーキテクチャ上は4GB
乗算器内蔵	・ 乗算器内蔵により、16×16→32の乗算を1～3ステートで実行、 16×16＋42→42の積和演算を2～3ステートで実行
パイプライン	・ 5段パイプライン方式
処理状態	・ プログラム実行状態 ・ 例外処理状態 ・ バス権開放状態 ・ リセット状態 ・ 低消費電力状態
低消費電力状態	・ スリープモード ・ スタンバイモード

1．2　レジスタ構成

SH7000シリーズCPUのレジスタ構成を表1．2に示します。SH7000シリーズCPUのレジスタには、汎用レジスタ(32ビット×16本)、コントロールレジスタ(32ビット×3本)、システムレジスタ(32ビット×4本)の3種類があります。

表1．2　SH7000シリーズCPUのレジスタ構成

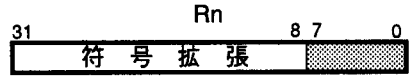
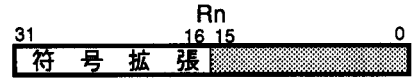
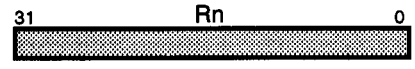
<table>
<tr><th>分類</th><th>レジスタ</th><th>説明</th></tr>
<tr><td>汎用レジスタ</td><td> 
</td><td>汎用レジスタ(Rn)は、32ビット長のレジスタで、R0からR15までの16本あります。汎用レジスタは、データ処理、アドレス計算に使用されます。いくつかの命令では使用できる汎用レジスタがR0に限定されています。R15は例外処理の中で、ハードウェアスタックポインタ(SP)として使用されるので、R15の使用に際しては注意が必要です。</td></tr>
<tr><td rowspan="4">コントロールレジスタ</td><td>ステータスレジスタ(SR)
 
</td><td>ステータスレジスタ(SR)は、32ビット長のレジスタで、CPUの処理状態を示します。SRの各ビットの機能を下表に示します。
<table>
<tr><th>ビット番号</th><th>ビット名</th><th>説明</th></tr>
<tr><td>0</td><td>Tビット</td><td>以下の命令では、真(1)、偽(0)を表します。
MOVT,CMP/cond,TAS,TST,BT,BF,SETT,CLRT
以下の命令では、キャリ、ボロー、オーバフロー、アンダフローを表します。
ADDV,ADDC,SUBV,SUBC,NEGC,DIV0U,DIV0S
DIV1,SHAR,SHAL,SHLR,SHLL,ROTR,ROTL
ROTCR,ROTCL</td></tr>
<tr><td>1</td><td>Sビット</td><td>MAC命令で使用します。</td></tr>
<tr><td>2～3</td><td>－</td><td>予約ビットです。0が読み出されます。書き込みは無効です。</td></tr>
<tr><td>4
5
6
7</td><td>I0ビット
I1ビット
I2ビット
I3ビット</td><td>割り込みマスクビットです。</td></tr>
<tr><td>8
9</td><td>Qビット
Mビット</td><td>DIV0U,DIV0S,DIV1命令で使用します。</td></tr>
<tr><td>10～31</td><td>－</td><td>予約ビットです。0が読み出されます。書き込みは無効です。</td></tr>
</table></td></tr>
<tr><td> 
</td><td>GBR間接アドレッシングモードのベースアドレスを示します。</td></tr>
<tr><td> 
</td><td>例外処理ベクタテーブルのベースアドレスを示します。</td></tr>
<tr><td rowspan="4">システムレジスタ</td><td>積和レジスタ
 
</td><td>乗算、積和演算の結果の格納レジスタです。MACHは下位10ビットが有効であり、読み出すときは符号拡張されます。</td></tr>
<tr><td> 
</td><td>サブルーチンプロシージャからの復帰先アドレスの格納レジスタです。</td></tr>
<tr><td> 
</td><td>実行中の命令の4バイト(2命令)先を示します。</td></tr>
</table>

1．3　データ形式

SH7000シリーズCPUは、バイト(8ビット)、ワード(16ビット)およびロングワード(32ビット)のデータを扱うことができます。汎用レジスタおよびメモリ上のデータ形式を以下に示します。

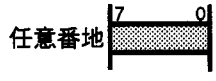
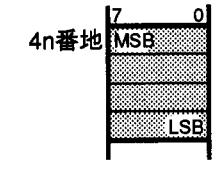
（1）汎用レジスタのデータ形式

汎用レジスタのデータサイズは常にロングワードです。バイトデータおよびワードのデータは、ロングワードに符号拡張されて、汎用レジスタへ格納されます。

データサイズ	汎用レジスタのデータ形式
バイト	
ワード	
ロングワード	

（2）メモリ上のデータ形式

メモリ上のデータサイズには、バイト、ワードおよびロングワードがあります。バイトデータは任意番地、ワードデータは2n番地、ロングワードは4n番地から配置してください。この境界以外からアクセスすると、アドレスエラーが発生します。このとき、アクセスした結果は保証しません。アドレスエラーについては、ハードウェアマニュアルを参照してください。

データサイズ	メモリ上でのデータ形式
バイト	
ワード	
ロングワード	

1．4　命令

1．4．1　命令の特長

命令はRISCタイプです。特長は次の通りです。

(1) 16ビット固定長命令

命令長はすべて16ビット固定長です。これによりプログラムのコード効率が向上します。

(2) 1命令/1ステート

パイプライン方式を採用し、基本命令は、1命令を1ステートで実行できます。20MHz動作時、1ステートは50nsになります。

(3) データサイズ

演算の基本的なデータサイズはロングワードです。メモリのアクセスサイズは、バイト、ワードおよびロングワードを選択できます。定数データは算術演算では符号拡張後、論理演算ではゼロ拡張後、ロングワードで演算されます。

表1．3　ワードデータの符号拡張

SH7000シリーズCPU	説明	他のCPU
MOV.W　@(disp,PC),R1 ADD　R1,R0 ⋮ .data.w　H'1234	32ビットに符号拡張され、R1はH'00001234になります。次にADD命令で演算されます。	ADD.W　#H'1234,R0

[注]　@(disp,PC)で定数データを参照します。

(4) ロード・ストア・アーキテクチャ

基本演算はレジスタ間で実行します。メモリとの演算は、レジスタにデータをロードし実行します(ロード・ストア・アーキテクチャ)。但し、ANDなどのビットを操作する命令は直接メモリに対して実行します。

(5) 遅延分岐

無条件分岐などは、遅延分岐命令です。遅延分岐命令の場合、遅延分岐命令の直後の命令を実行してから、分岐します。これにより、分岐時のパイプラインの乱れを軽減しています。

表1．4　遅延分岐命令

SH7000シリーズCPU	説明	他のCPU
BRA　TRGET ADD　R1,R0	TRGETに分岐する前にADDを実行します。	ADD.W　R1,R0 BRA　TRGET

(6) 乗算、積和演算

5段パイプライン方式と乗算機内蔵により、16×16→32の乗算を1～3ステートで、16×16＋42→42の積和演算を2～3ステートで実行します。

(7) Tビット

比較結果はステータスレジスタ(SR)のTビットに反映し、その真、為によって条件分岐します。必要最小限の命令によってのみTビットを変化させ、処理速度を向上させています。

表1.5 Tビット

SH7000シリーズCPU	説明	他のCPU
CMP/GE R1,R0	R0≧R1のときTビットがセットされます。	CMP.W R1,R0
BT TRGET0	R0≧R1のときTRGET0へ分岐します。	BEG TRGET0
BF TRGET1	R0＜R1のときTRGET1へ分岐します。	BLT TRGET1
ADD #-1,R0	ADDではTビットが変化しません。	SUB.W #1,R0
CMP/EQ #0,R0	R0＝0のときTビットがセットされます。	BEQ TRGET
BT TRGET	R0＝0のとき分岐します。	

(8) 定数データ

バイトの定数データは命令コードの中に配置します。ワードとロングワードの定数データは命令コードの中に配置せず、メモリ上のテーブルに配置します。メモリ上のテーブルはディスプレースメント付きPC相対アドレッシングモードを使った定数データのデータ転送命令(MOV)で参照します。

表1.6 イミディエイトデータによる参照

区分	SH7000シリーズCPU	他のCPU
8ビット定数	MOV #H'12,R0	MOV.B #H'12,R0
16ビット定数	MOV.W @(disp,PC),R0 ⋮ .data.w H'1234	MOV.W #H'1234,R0
32ビット定数	MOV.L @(disp,PC),R0 ⋮ ..data.w H'12345678	MOV.W #H'12345678,R0

[注] @(disp,PC)で定数データを参照します。

(9) 絶対アドレス

絶対アドレスでデータを参照するときは、あらかじめ絶対アドレスの値を、メモリ上のテーブルに配置しておきます。命令実行時に定数データをロードする方法で、この値をレジスタに転送し、レジスタ間接アドレッシングモードでデータを参照します。

表1.7 絶対アドレスによる参照

区分	SH7000シリーズCPU	他のCPU
絶対アドレス	MOV.L @(disp,PC),R1 MOV.B @R1,R0 ⋮ .data.l H'12345678	MOV.B @H'12345678,R0

(10) 16ビット、32ビットディスプレースメント

16ビットまたは32ビットディスプレースメントでデータを参照するときは、あらかじめディスプレースメントの値をメモリ上のテーブルに配置しておきます。命令実行時にイミディエイトデータをロードする方法で、この値をレジスタに転送し、インデックス付きレジスタ間接アドッレシングモードでデータを参照します。

表1.8 ディスプレースメントによる参照

区分	SH7000のCPU	他のCPU
16ビットディスプレースメント	MOV.W @(disp,PC),R0 MOV.W @(R0,R1),R2 ⋮ .DATA.W H'1234	MOV.W @(H'1234,R1),R2

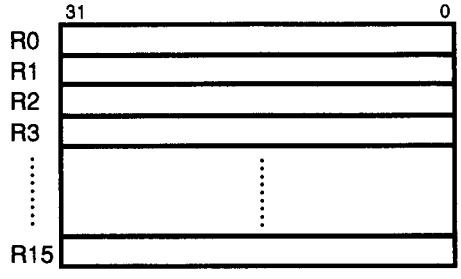
1．4．2　アドレッシングモード

SH7000シリーズCPUは、表1.9に示すアドレッシングモードをサポートしています。命令によって使用できるアドレッシングモードは異なります。

表1.9　SH7000シリーズCPUのアドレッシングモード

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	指定対象	備　考
レジスタ直接	レジスタ	
イミディエイト	8ビット定数データ	
レジスタ間接	アドレス	
ポストインクリメントレジスタ間接		
プリデクリメントレジスタ間接		
ディスプレースメント付きレジスタ間接		
インデックス付きレジスタ間接		
ディスプレースメント付きGBR間接		内蔵周辺モジュールレジスタのアドレス
インデックス付きGBR間接		内蔵周辺モジュールレジスタのアドレス
PC相対		分岐先のアドレス
ディスプレースメント付きPC相対		16ビットおよび32ビット定数データのアドレス

■レジスタ直接

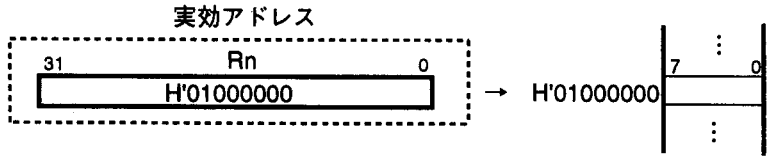
ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	表　記	説　明
レジスタ直接	Rn	R0～R15の汎用レジスタの指定に用います。いくつかの命令では、指定できる汎用レジスタがR0に限定されているので注意してください。 

■イミディエイト

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	表　記	説　明
イミディエイト	#imm:8	命令コードの第2バイトに配置されている8ビット定数データの指定に用います。8ビット定数データは、命令によって下表に示すように拡張されます。

命令	拡張方法
TST,AND,OR,XOR	32ビットにゼロ拡張されます。
MOV,ADD,CMP/EQ	32ビットに符号拡張されます。
TRAPA	32ビットにゼロ拡張後、4倍されます。

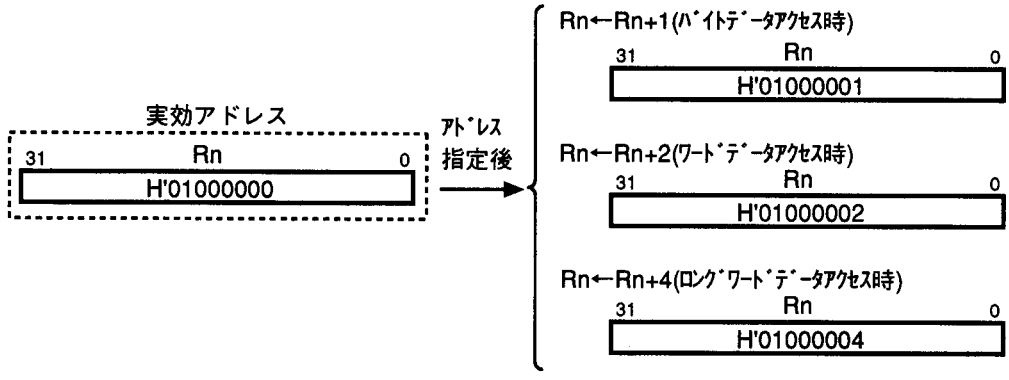
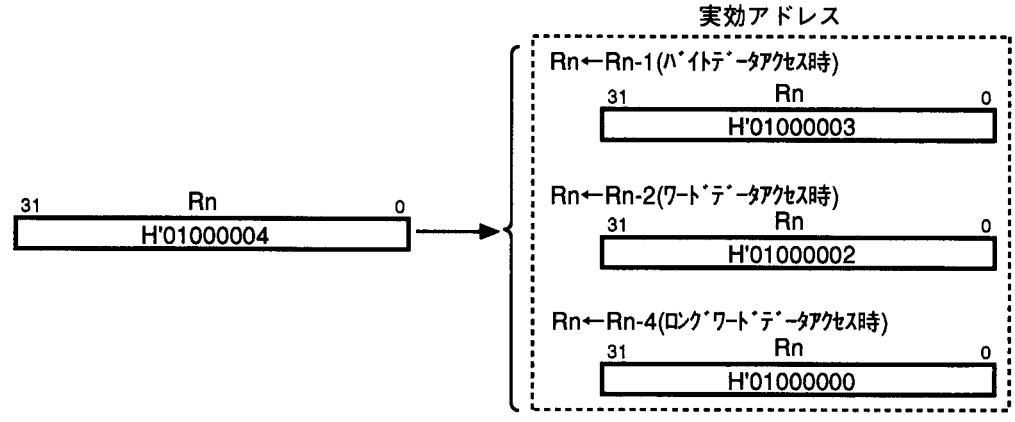
■レジスタ間接

<table>
<tr><th>ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ</th><th>表　記</th><th>説　明</th></tr>
<tr><td>レジスタ間接</td><td>@Rn</td><td>汎用レジスタRnの内容が実効アドレスです。アクセスするメモリのデータサイズにより、汎用レジスタRnへ設定するアドレスは下表に示すようにしてください。
 

<table>
<tr><th>ｱｸｾｽﾃﾞｰﾀｻｲｽﾞ</th><th>アドレス</th></tr>
<tr><td>バイト</td><td>任意番地</td></tr>
<tr><td>ワード</td><td>2n番地</td></tr>
<tr><td>ロングワード</td><td>4n番地</td></tr>
</table></td></tr>
</table>

■ポストインクリメントレジスタ間接、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およびプリデクリメントレジスタ間接は、スタック上またはデータテーブル上のデータのアドレス指定に用いられます。

<table>
<tr><th>ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ</th><th>表　記</th><th>説　明</th></tr>
<tr><td>ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄ
レジスタ間接</td><td>@Rn+</td><td>汎用レジスタRnの内容が実効アドレスです。但し、アドレス指定後に汎用レジスタRnの内容にアクセスしたメモリのデータサイズ(バイト=1、ワード=2、ロングワード=4)が自動的に加算されます。スタックからのデータの復帰やデータテーブルのデータの参照に使用します。
 
</td></tr>
<tr><td>ﾌﾟﾘﾃﾞｸﾘﾒﾝﾄ
レジスタ間接</td><td>@-Rn</td><td>汎用レジスタRnの内容が実効アドレスです。但し、アドレス指定前に汎用レジスタRnの内容からアクセスするメモリのデータサイズ(バイト=1、ワード=2、ロングワード=4)が自動的に減算されます。このアドレッシングモードは、デスティネーションオペランドのみサポートされています。スタックへのデータの退避等に使用します。
 
</td></tr>
</table>

■ディスプレースメント付きレジスタ間接、インデックス付きレジスタ間接

ディスプレースメント付きレジスタ間接およびインデックス付きレジスタ間接は、図1.1に示すようにベースアドレスとなる汎用レジスタRnの内容に指定先アドレスとの相対距離を加算して実効アドレスを求めます。

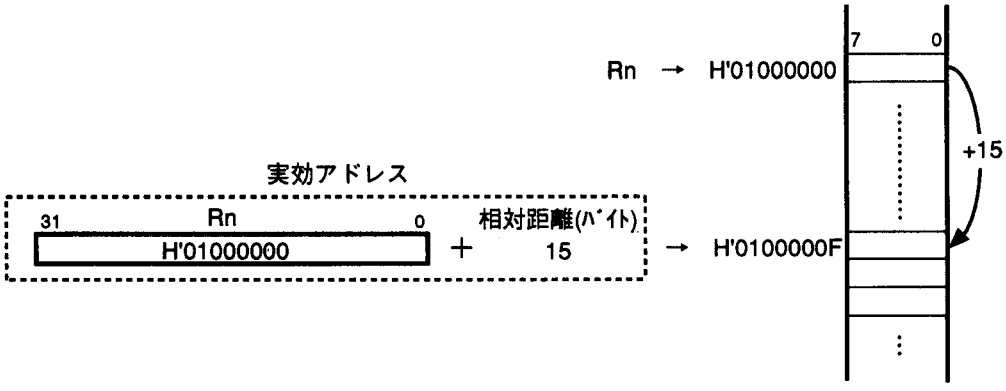


図1.1　実効アドレス

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	表　記	ベースアドレス	相対距離
ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きレジスタ間接	@(disp:4 , Rn)	ベースアドレスは汎用レジスタRnの内容です。汎用レジスタRnは相対距離の加算によって変化しません。 31 Rn 0 ベースアドレス	相対距離は4ビットディスプレースメントで示します。4ビットディスプレースメントはゼロ拡張後、アクセスデータサイズがバイトのときは1倍、ワードのときは2倍、ロングワードの時は4倍されるので、相対距離は下表に示す範囲になります。 ｱｸｾｽﾃﾞｰﾀｻｲｽﾞ / 相対距離(ﾊﾞｲﾄ) バイト / 0～+15 ワード / 0～+30の間で2の倍数 ロングワード / 0～+60の間で4の倍数
インデックス付きレジスタ間接	@(R0 , Rn)		相対距離は汎用レジスタに設定します。但し、汎用レジスタはR0に限定されます。相対距離が4ビットディスプレースメントで届かない場合に使用します。 31 R0 0 相対距離

■ディスプレースメント付きGBR間接、インデックス付きGBR間接

ディスプレースメント付きGBR間接およびインデックス付きGBR間接は、内蔵周辺モジュールレジスタのアドレス指定に用います。これらのアドレッシングモードは、図1.2に示すようにベースアドレスとなるGBRの内容に指定先アドレスとの相対距離を加算して実効アドレスを求めます。ディスプレースメント付きGBR間接はMOV命令をサポートし、インデックス付きGBR間接は論理演算命令(AND,OR,NOT,TST)をサポートしてます。ベースアドレスに相対距離を加算した値が実効アドレスです。

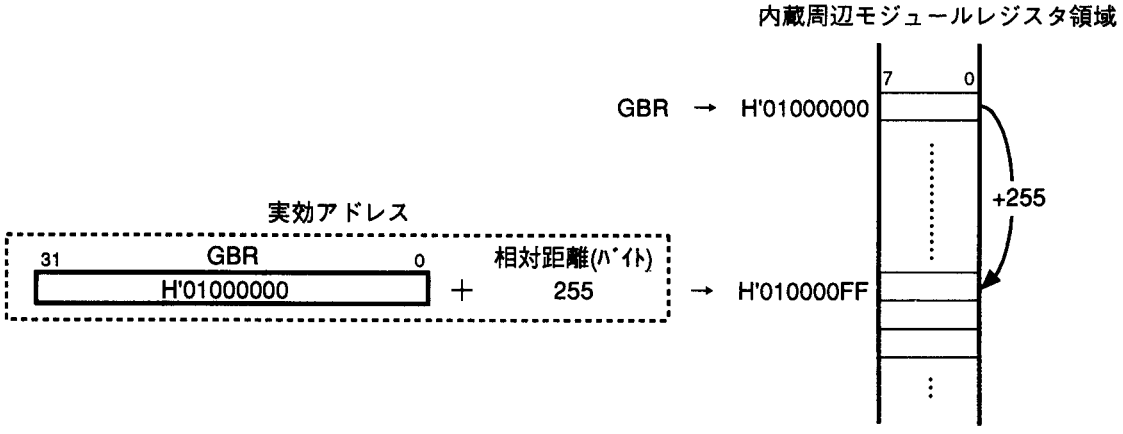


図1.2　実効アドレス

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	表　記	ベースアドレス	相対距離
ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きGBR間接	@(disp:8, GBR)	ベースアドレスはGBRの内容です。GBRの内容は相対距離の加算によって変化しません。 31 GBR 0 ベースアドレス	相対距離は8ビットディスプレースメントで示します。8ビットディスプレースメントはゼロ拡張後、アクセスデータサイズがバイトのときは1倍、ワードのときは2倍、ロングワードの時は4倍されるので、相対距離は下表に示す範囲になります。 ｱｸｾｽﾃﾞｰﾀｻｲｽﾞ：相対距離(ﾊﾞｲﾄ) バイト：0～＋255 ワード：0～＋510の間で2の倍数 ロングワード：0～＋1020の間で4の倍数
インデックス付きGBR間接	@(R0, GBR)		相対距離は汎用レジスタに設定します。但し、汎用レジスタはR0に限定されます。相対距離が8ビットディスプレースメントで届かない場合に使用します。 31 R0 0 相対距離

■PC相対、ディスプレースメント付きPC相対

PC相対およびディスプレースメント付きPC相対はプログラム領域のアドレス指定に用いられます。これらのアドレッシングモードは、図1.3に示すようにベースアドレスとなるPCの内容に指定先アドレスとの相対距離を加算して実効アドレスを求めます。PC相対は分岐命令(BF,BT,BRA,BSR)の分岐先アドレスの指定に用いられ、ディスプレースメント付きPC相対はプログラム領域内に配置された16ビットまたは32ビットの定数データのアドレスの指定に用いられます。

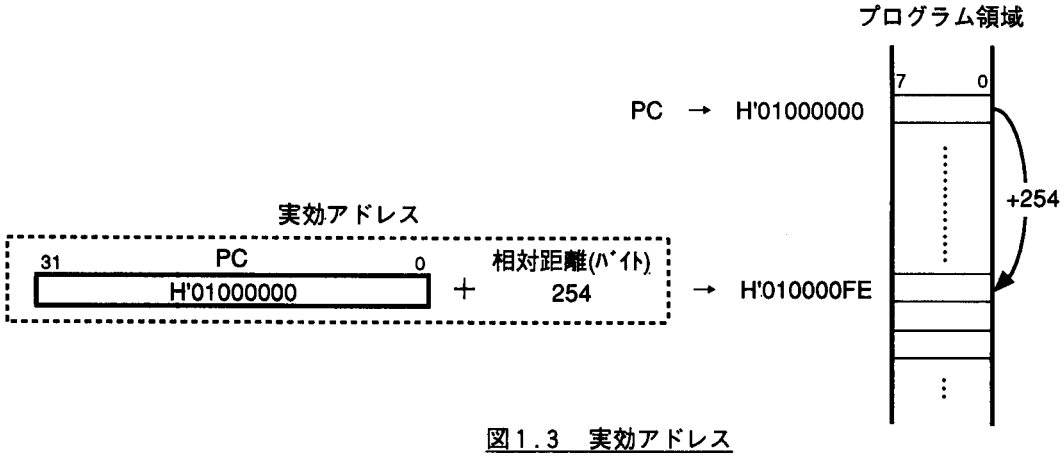


図1.3　実効アドレス

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	表　記	ベースアドレス	相対距離
PC相対	disp：8	ベースアドレスはPCの内容(2命令後のアドレス)です。相対距離の加算により、PCの内容は分岐先のアドレスとなります。 31 PC 0 ベースアドレス	相対距離は8ビットディスプレースメントで示します。8ビットディスプレースメントは符号拡張後2倍されるので、相対距離は−256～＋254の範囲になります。
	disp：12		相対距離は12ビットディスプレースメントで示します。12ビットディスプレースメントは符号拡張後2倍されるので、相対距離は−4096～＋4094の範囲になります。
ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きPC相対	@(disp:8，PC)	ベースアドレスはPCの内容(2命令後のアドレス)です。PCの内容は相対距離の加算により変化しません。アクセスデータサイズがロングワードの時はPCの内容の下位2ビットをマスクした値をベースアドレスとします。 31 PC 0 ベースアドレス	相対距離は8ビットディスプレースメントで示します。8ビットディスプレースメントはゼロ拡張後、アクセスデータサイズがワードのときは2倍、ロングワードの時は4倍されるので、相対距離は下表に示す範囲になります。

ｱｸｾｽﾃﾞｰﾀｻｲｽﾞ	相対距離(ﾊﾞｲﾄ)
ワード	0～＋510の間で2の倍数
ロングワード	0～＋1020のの間4の倍数

１．４．３　命令の分類

SH7000シリーズCPUの命令の種類は、各命令のもつ機能によって、表１．１０に示すように分類されます。各命令についての詳細は、１．４．４の各命令の説明を参照してください。

表１．１０　命令の分類

分類	オペコード	機能
データ転送命令	MOV	データ転送
	MOVA	実効アドレスの転送
	MOVT	Tビットの転送
	SWAP	上位と下位の交換
	XTRCT	連結レジスタの中央切り出し
算術演算命令	ADD	2進加算
	ADDC	キャリ付き2進加算
	ADDV	オーバフロー付き2進加算
	CMP/cond	比較
	DIV1	除算
	DIV0S	符号付き除算の初期化
	DIV0U	符号なし除算の初期化
	EXTS	符号拡張
	EXTU	ゼロ拡張
	MAC	積和演算
	MULS	符号付き乗算
	MULU	符号なし乗算
	NEG	符号反転
	NEGC	ボロー付き符号反転
	SUB	2進減算
	SUBC	ボロー付き2進減算
	SUBV	アンダフロー付き2進減算
論理演算命令	AND	論理積演算
	NOT	ビット反転
	OR	論理和演算
	TAS	メモリテストとビットセット
	TST	論理積演算のTビットセット
	XOR	排他的論理和演算

(続く)

分類	オペコード	機能
シフト命令	ROTL	1ビット左回転
	ROTR	1ビット右回転
	ROTCL	Tビット付き1ビット左回転
	ROTCR	Tビット付き1ビット右回転
	SHAL	算術的1ビット左シフト
	SHAR	算術的1ビット右シフト
	SHLL	論理的1ビット左シフト
	SHLLn	論理的nビット左シフト
	SHLR	論理的1ビット右シフト
	SHLRn	論理的nビット右シフト
分岐命令	BF	条件分岐(T=0で分岐)
	BT	条件分岐(T=1で分岐)
	BRA	無条件分岐
	BSR	サブルーチンプロシージャへの分岐
	JMP	無条件分岐
	JSR	サブルーチンプロシージャへの分岐
	RTS	サブルーチンプロシージャからの復帰
システム制御命令	CLRT	Tビットのクリア
	CLRMAC	MACレジスタのクリア
	LDC	コントロールレジスタへのロード
	LDS	システムレジスタへのロード
	NOP	無操作
	RTE	例外処理からの復帰
	SETT	Tビットのセット
	SLEEP	低消費電力モードへの遷移
	STC	コントロールレジスタからのストア
	STS	システムレジスタからのストア
	TRAPA	トラップ例外処理

１．４．４　各命令の説明

図１.４にSH7000シリーズCPUのアセンブラフォーマットを示します。

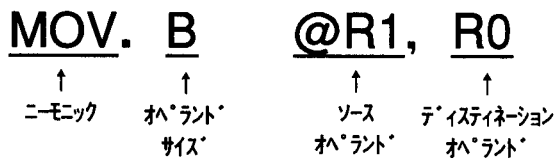


図１.４　SH7000シリーズCPUのアセンブラフォーマット

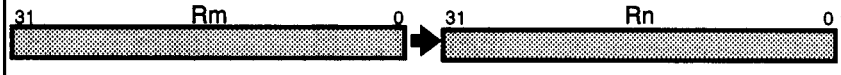
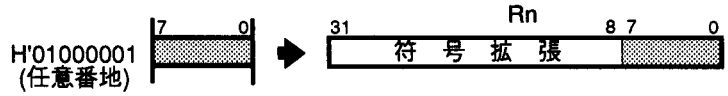
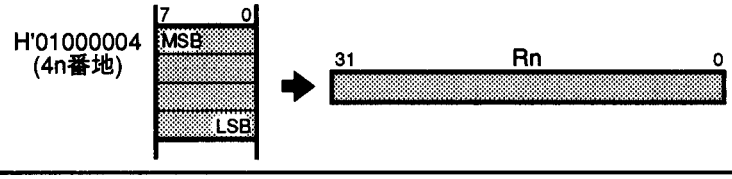
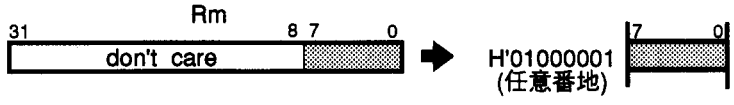
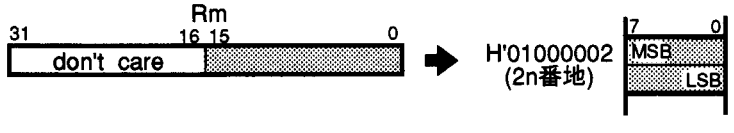
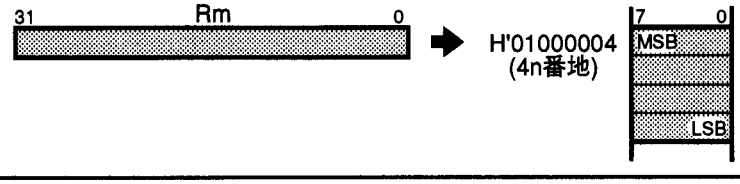
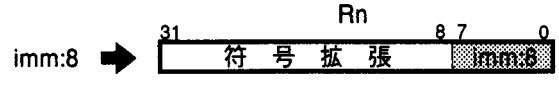
各命令は図１.５に示すフォーマットで説明します。

命令の種類

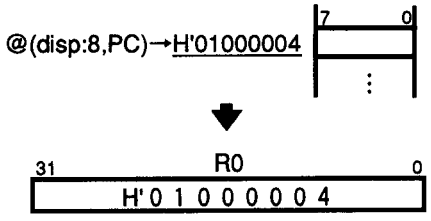
命令

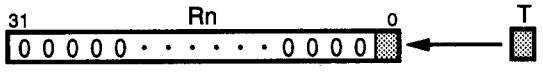
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
命令のﾆｰﾓﾆｯｸを示します。	命令のｵﾍﾟﾗﾝﾄﾞ･ｻｲｽﾞを示します。 但し、命令によっては指定ないものもあります。	命令のｿｰｽｵﾍﾟﾗﾝﾄﾞを示します。 但し、命令によっては指定ないものもあります。	命令のﾃﾞｨｽﾃｨﾈｰｼｮﾝｵﾍﾟﾗﾝﾄﾞを示します。 但し、命令によっては指定ないものもあります。	命令の機能を説明します。

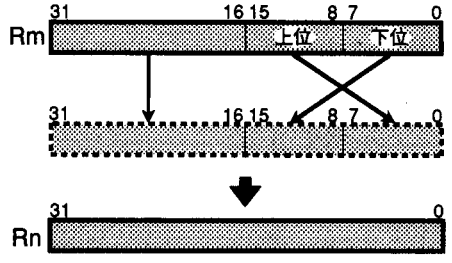
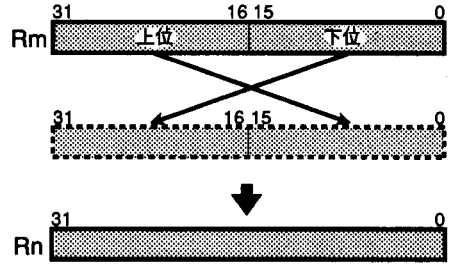
図１.５　命令の説明フォーマット

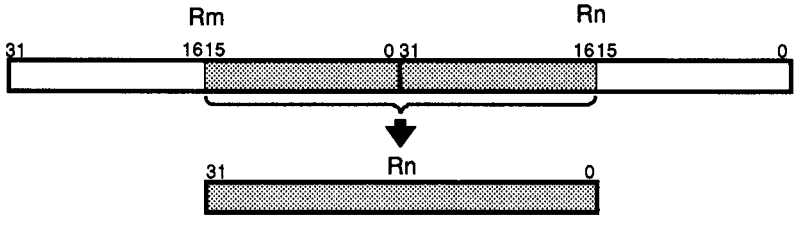
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
MOV		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRmの内容を汎用レジスタRnへ転送します。 
	B	@Rm @Rm+ @(disp:4,Rm) @(R0,Rm) @(disp:8,GBR)	Rn (但し、ソースオペランドが@(disp:4,Rm) @(disp:8,GBR)の場合は、R0に限定されます。)	メモリ(任意番地)上のバイトデータを汎用レジスタRnへ転送します。汎用レジスタRn上ではバイトデータは32ビットに符号拡張されます。 
	W			メモリ(2n番地)上のワードデータを汎用レジスタRnへ転送します。汎用レジスタRn上ではワードデータは32ビットに符号拡張されます。 
	L		Rn (R0 ～ R15)	メモリ(4n番地)上のロングワードデータを汎用レジスタRnへ転送します。 
	B	Rm (但し、デスティネーションオペランドが@(disp:4,Rn) @(disp:8,GBR)の場合は、R0に限定されます。)	@Rn @-Rn @(disp:4,Rn) @(R0,Rn) @(disp:8,GBR)	汎用レジスタRmの下位8ビットの内容をメモリ(任意番地)へ転送します。 
	W			汎用レジスタRmの下位16ビットの内容をメモリ(2n番地)へ転送します。 
	L	Rm (R0 ～ R15)		汎用レジスタRmの内容をメモリ(4n番地)へ転送します。 
		#imm:8	Rn (R0 ～ R15)	8ビット定数データを汎用レジスタRnへ転送します。汎用レジスタRn上では8ビット定数データは32ビットに符号拡張されます。 

MOVA　MOVT

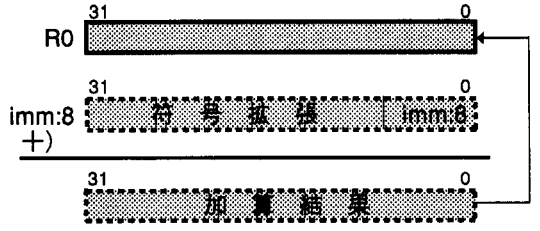
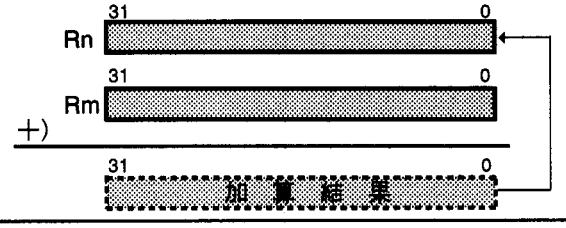
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
MOVA		@(disp:8,PC) または シンボル	R0に限定	アドレス（@(disp:8,PC)）を汎用レジスタR0へ格納します。データテーブルを使用する処理において、データテーブルの先頭番地を定数データ(32ビット)で持つ必要がなくなります。 

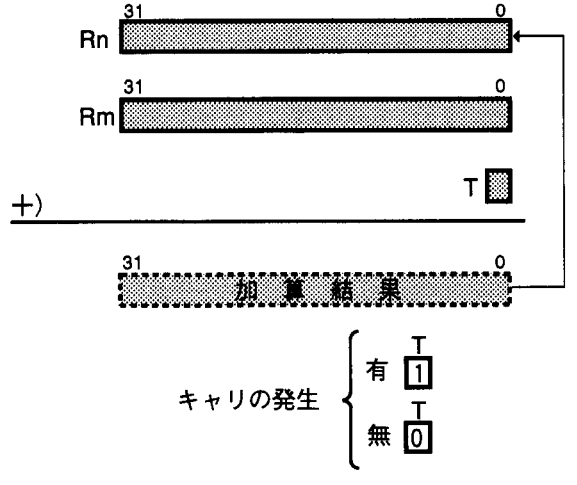
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
MOVT			Rn (R0 ～ R15)	Tビットの内容を汎用レジスタRnのビット0へ転送します。ビット1～ビット31は0となります。 

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
SWAP	B	Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRmの上位16ビットはそのままで、下位16ビットの上位8ビットと下位8ビットを交換した内容を汎用レジスタRnへ格納します。 
	W			汎用レジスタRmの上位16ビットと下位16ビットを交換した内容を汎用レジスタRnへ格納します。 

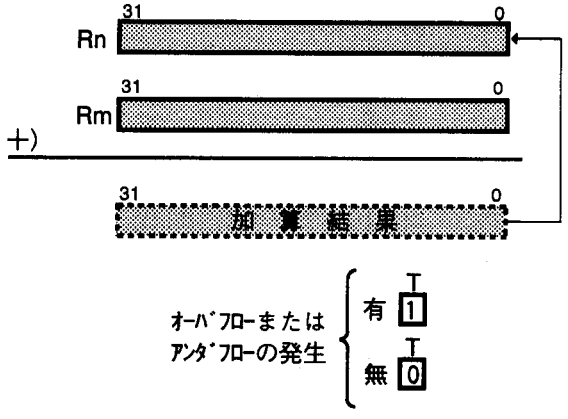
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
XTRCT		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRmを上位32ビット、汎用レジスタRnを下位32ビットとした64ビットの内容の中央の32ビットの内容を汎用レジスタRnに転送します。 

ADD　ADDC

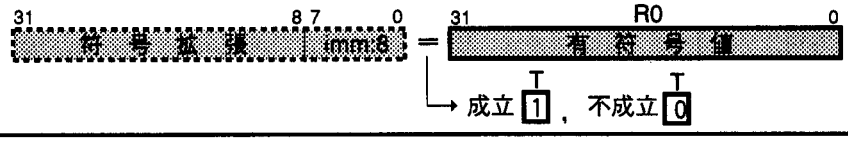
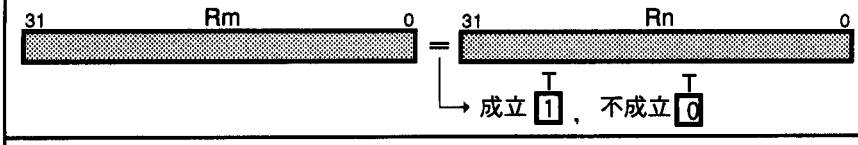
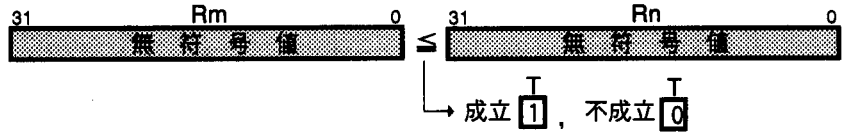
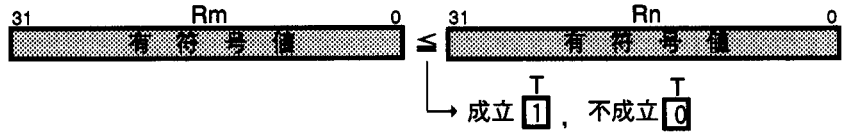
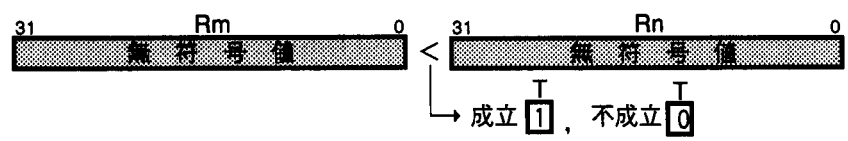
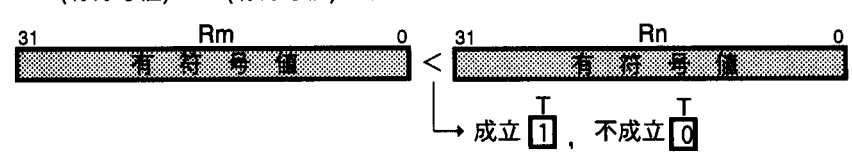
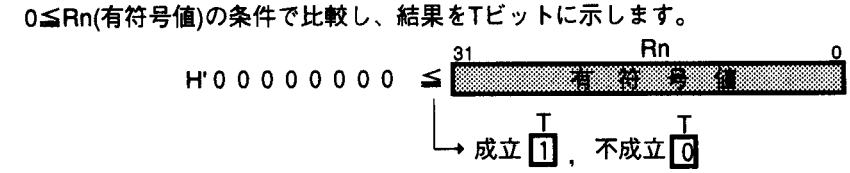
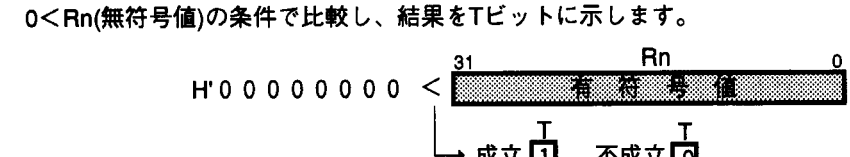
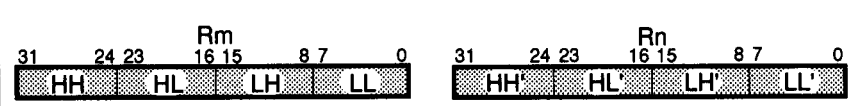
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
ADD		#imm:8	R0に限定	汎用レジスタR0の内容に8ビット定数データを加算します。加算結果は汎用レジスタR0へ返します。8ビット定数データは32ビットに符号拡張され−128～+127の範囲となるので、この命令で減算も可能です。 
		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRnの内容に汎用レジスタRmの内容を加算します。加算結果は汎用レジスタRnへ返します。 

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
ADDC		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRnの内容に汎用レジスタRmの内容およびTビットの内容を加算します。加算結果は汎用レジスタRnへ返します。また、キャリの発生の有無をTビットに示します(有:T=1,無:T=0)。32ビットを超える加算に使用します。 

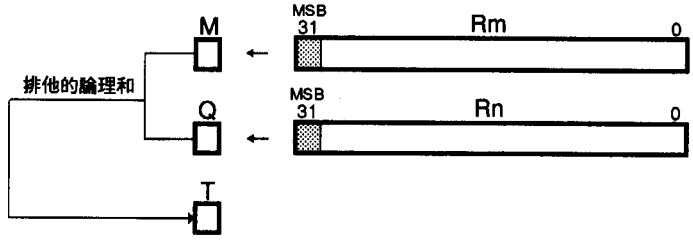
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
ADDV		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRnの内容に汎用レジスタRmの内容を加算します。加算結果は汎用レジスタRnへ返します。また、オーバフローまたはアンダーフローの発生の有無をTビットに示します(有:T=1,無:T=0)。オーバフローの発生かアンダーフローの発生かの判定は、下表をもとに行ってください。有符号値の加算に使用します。

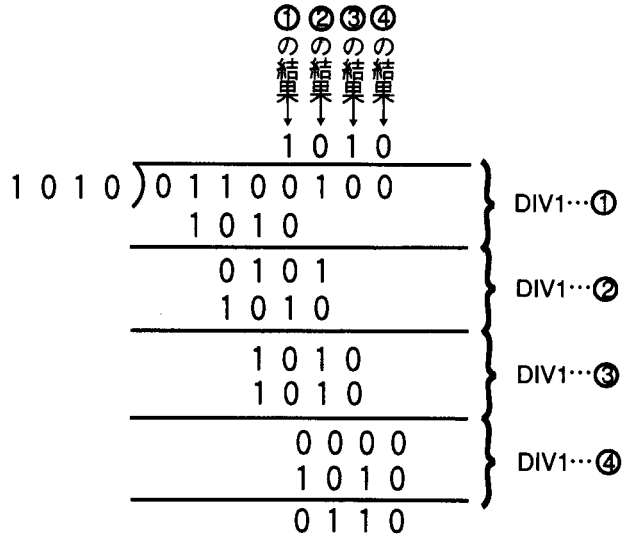


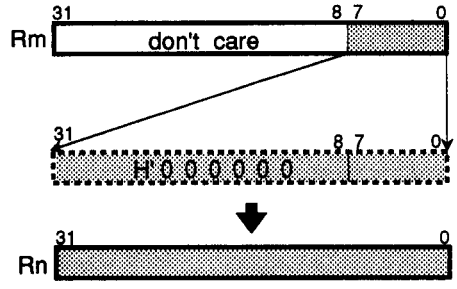
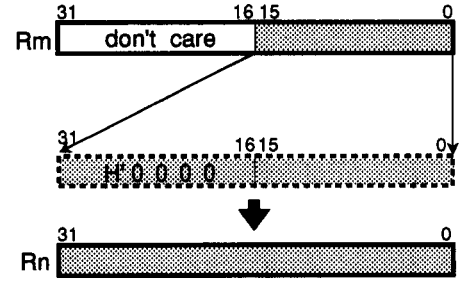
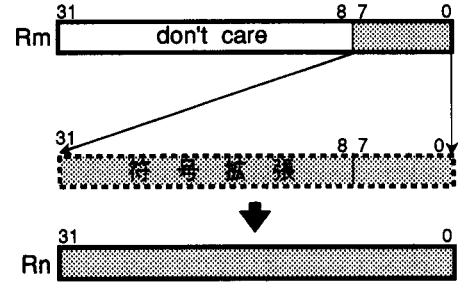
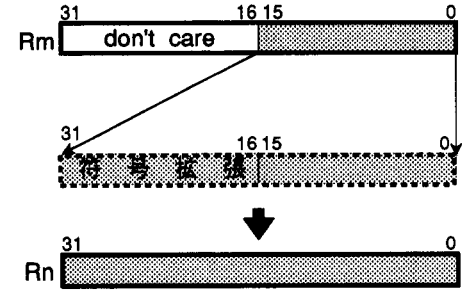
汎用レジスタRn (被加数)	汎用レジスタRm (加数)	汎用レジスタRn (結果)	オーバフロー, アンダフロー
正	正	正	
		負	オーバフロー
正	負	正	
		負	
負	正	正	
		負	
負	負	正	アンダフロー
		負	

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
CMP/EQ		#imm:8	R0に限定	imm:8(-128～+127)＝R0(有符号値)の条件で比較し、結果をTビットに示します。 
CMP/EQ		Rm (R0～R15)	Rn (R0～R15)	Rm＝Rnの条件で比較し、結果をTビットに示します。 
CMP/HS				Rm(無符号値)≦Rn(無符号値)の条件で比較し、結果をTビットに示します。 
CMP/GE				Rm(有符号値)≦Rn(有符号値)の条件で比較し、結果をTビットに示します。 
CMP/HI				Rm(無符号値)＜Rn(無符号値)の条件で比較し、結果をTビットに示します。 
CMP/GT				Rm(有符号値)＜Rn(有符号値)の条件で比較し、結果をTビットに示します。 
CMP/PZ				0≦Rn(有符号値)の条件で比較し、結果をTビットに示します。 
CMP/PL				0＜Rn(無符号値)の条件で比較し、結果をTビットに示します。 
CMP/STR				Rm(HH)＝Rn(HH')、Rm(LH)＝Rn(LH')、Rm(HL)＝Rn(HL')およびRm(LL)＝Rn(LL')の比較を行い、いずれか1つが成立していればTビットをセットし、すべて不成立ならばTビットをクリアします。 

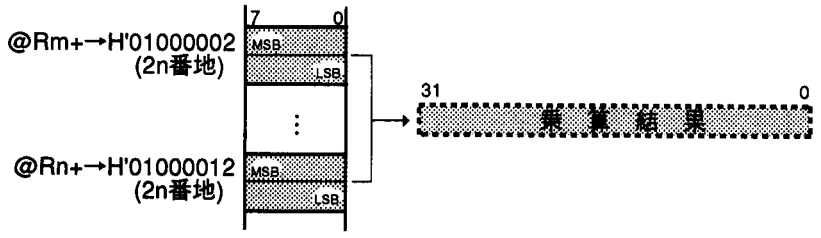
DIV0S　DIV0U　DIV1

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
DIV0U				符号なし除算の初期設定をします。SRのMビット、Qビット、Tビットをクリアします。 M 0　Q 0　T 0
DIV0S		Rm (R0 ～ R15)	Rn (R0 ～ R15)	符号付き除算の初期設定をします。汎用レジスタRm(除数)のMSBの内容をMビットへ格納し、汎用レジスタRn(被除数)のMSBの内容をQビットへ格納し、MビットとQビットとの排他的論理和(商の符号)をTビットへ格納します。 

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
DIV1		Rm	Rn	汎用レジスタRnの32ビットの内容(被除数)を汎用レジスタRmの内容(除数)で1桁分の除算(1ｽﾃｯﾌﾟ除算)を実行する命令です。1ｽﾃｯﾌﾟ除算とは、被除数を左に1ビットシフトし、それから除数の絶対値を減算（被除数が負のときは加算）し、結果の正負によって商（1桁）をTビットに求めます。下図に示すように、本命令を除数のビット数分繰り返して行ないます。この繰り返し中は、M、Q、Tビットを書き替えないでください。詳しい除算のシーケンスは、ソフトウェア応用例を参考にしてください。 

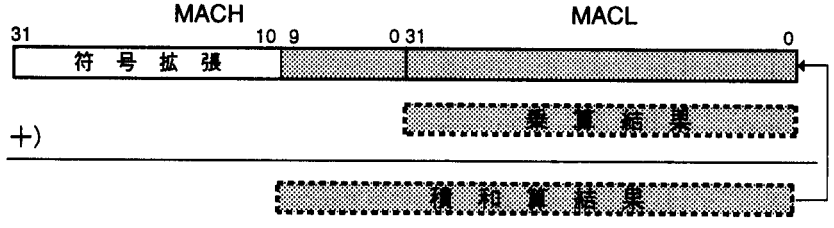
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
EXTU	B	Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRmの下位8ビットの内容を32ビットにゼロ拡張した内容を汎用レジスタRnへ格納します。 
	W			汎用レジスタRmの下位16ビットの内容を32ビットにゼロ拡張した内容を汎用レジスタRnへ格納します。 
EXTS	B			汎用レジスタRmの下位8ビットの内容を32ビットに符号拡張(ﾋﾞｯﾄ8～ﾋﾞｯﾄ31にﾋﾞｯﾄ7の内容を転送)した内容を汎用レジスタRnへ格納します。 
	W			汎用レジスタRmの下位16ビットの内容を32ビットに符号拡張(ﾋﾞｯﾄ8～ﾋﾞｯﾄ31にﾋﾞｯﾄ7の内容を転送)した内容を汎用レジスタRnへ格納します。 

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
MAC	W	@Rm+	@Rn+	MACの内容に@Rm+のメモリの内容(16ビット)と@Rn+のメモリの内容(16ビット)を有符号値として乗算した結果を加算し、加算結果をMACに返します。但し、SRのSビットの内容により結果の範囲が異なります。



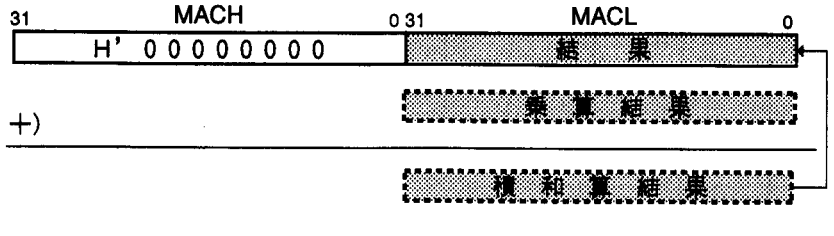
《Sビット=0のとき》

MACHおよびMACLを使用します。MACLには結果の下位32ビットが格納され、MACHには結果の上位10ビットが格納されます。また、MACHの上位22ビットは符号拡張されます。

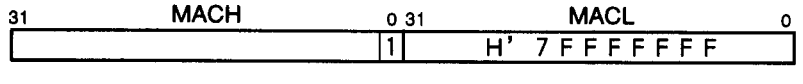


《Sビット=1のとき(飽和演算)》

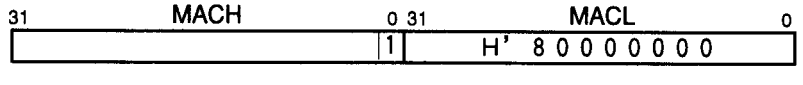
MACLレジスタのみが有効となり、結果の範囲をH'80000000～H'7FFFFFFFに制限します。また、オーバフローまたはアンダフローが発生した場合、MACHのLSBを1にセットし、オーバフローのときはH'7FFFFFFF、アンダフローのときはH'80000000をMACLへ格納します。



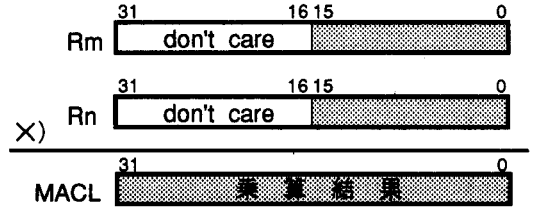
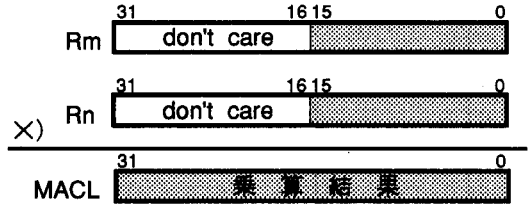
〈オーバフローしたとき〉

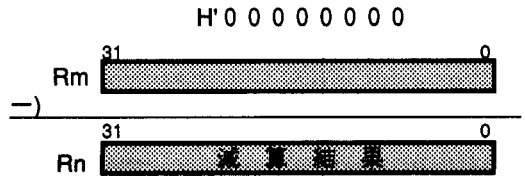
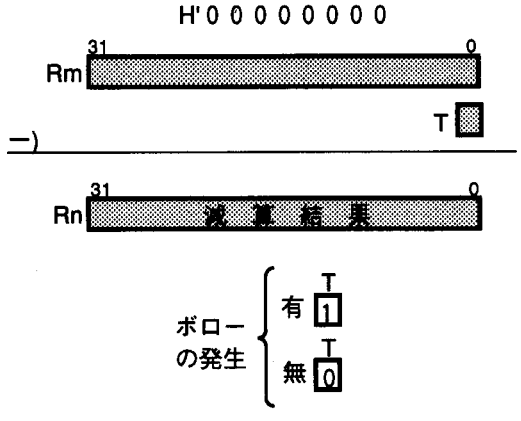


〈アンダフローしたとき〉



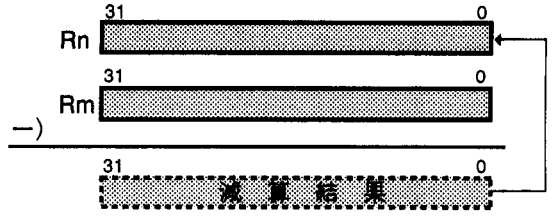
MULU　MULS

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
MULU		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRmの下位16ビットの内容と汎用レジスタRnの下位16ビットの内容を無符号値として乗算し、乗算結果(32ビット)をMACLに格納します。乗算によって汎用レジスタRmの内容および汎用レジスタRnの内容は変化しません。 
MULS				汎用レジスタRmの下位16ビットの内容と汎用レジスタRnの下位16ビットの内容を有符号値として乗算し、乗算結果(32ビット)をMACLに格納します。乗算によって汎用レジスタRmの内容および汎用レジスタRnの内容は変化しません。 

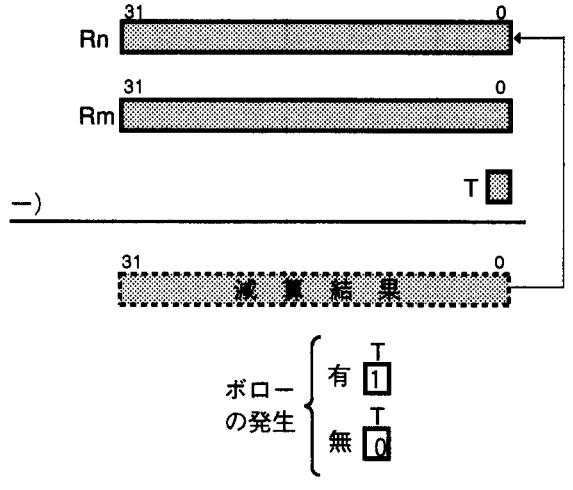
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
NEG		Rm (R0 ～ R15)	Rn (R0 ～ R15)	H'00000000から汎用レジスタRmの内容を減算し、減算結果を汎用レジスタRnへ格納します。32ビット以下のデータの符号反転に使用します。 
NEGC				H'00000000から汎用レジスタRmの内容およびTビットの内容を減算し、減算結果を汎用レジスタRnへ格納します。また、ボローの発生の有無をTビットに示します(有:T=1無:T=0)。32ビットを超えるデータの符号反転に使用します。 

SUB　SUBC

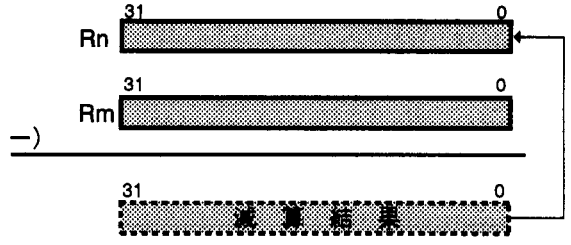
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
SUB		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRnの内容から汎用レジスタRmの内容を減算します。減算結果は汎用レジスタRnへ返します。

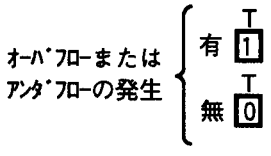


ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
SUBC		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRnの内容から汎用レジスタRmの内容およびTビットの内容を減算します。減算結果は汎用レジスタRnへ返します。また、ボローの発生の有無をTビットに示します(有:T=1無:T=0)。32ビットを超える減算に使用します。



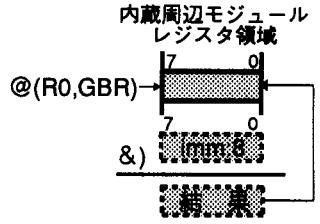
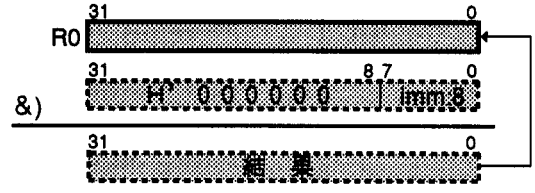
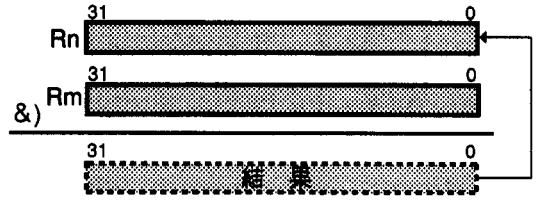
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
SUBV		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRnの内容から汎用レジスタRmの内容を減算します。減算結果は汎用レジスタRnへ返します。オーバフローまたはアンダーフローの発生の有無をTビットに示します(有:T=1無:T=0)。オーバフローかアンダーフローかの判定は、下表をもとに行ってください。有符号値の減算に使用します。

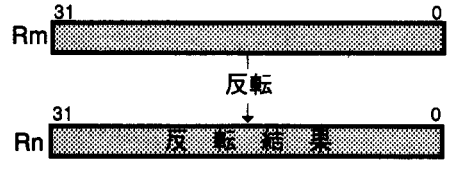


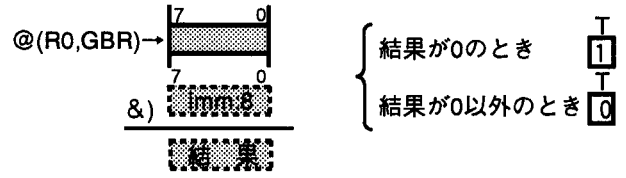
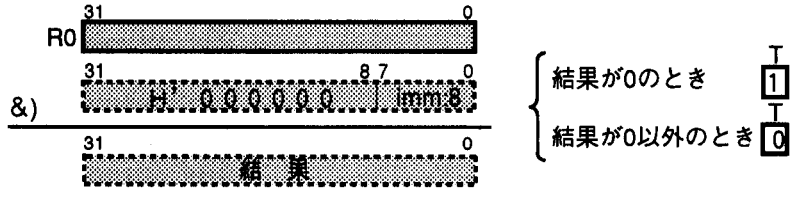
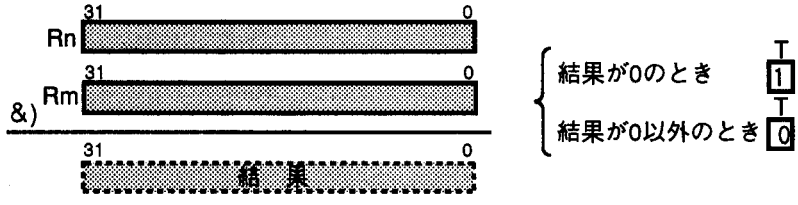


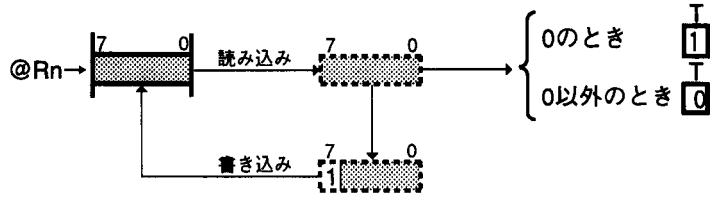
汎用レジスタRn (被減数)	汎用レジスタRm (減数)	汎用レジスタRn (減算結果)	オーバフロー, アンダフロー
正	正	正	
		負	
正	負	正	オーバフロー
		負	
負	正	正	アンダフロー
		負	
負	負	正	
		負	

AND | NOT | OR | XOR

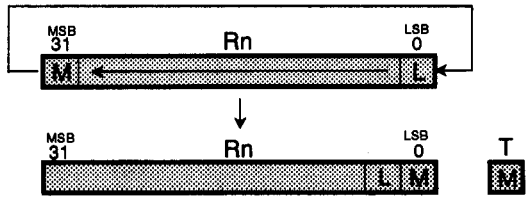
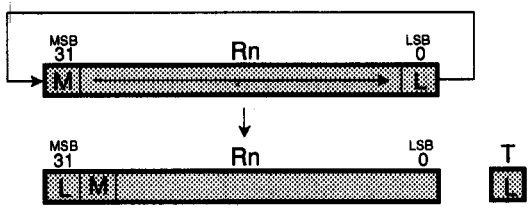
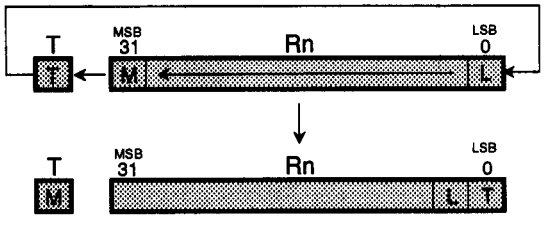
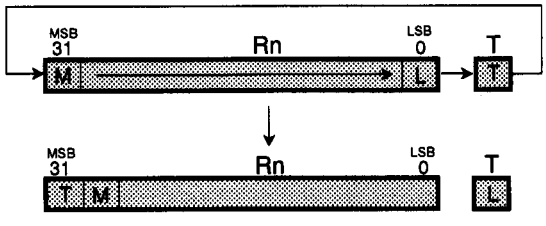
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
AND OR XOR	B	#imm:8	@(R0,GBR)	@(R0,GBR)のメモリの内容(8ﾋﾞｯﾄ)と8ビット定数データとの論理演算(論理積、論理和、排他的論理和)を行い、結果を@(R0,GBR)へ返します。内蔵周辺モジュールレジスタのビット操作(ﾋﾞｯﾄｾｯﾄ、ﾋﾞｯﾄｸﾘｱ、ﾋﾞｯﾄ反転)に使用すると便利です。 
		#imm:8	R0に限定	汎用レジスタR0の内容と8ビット定数データ(ｾﾞﾛ拡張)との論理演算(論理積、論理和、排他的論理和)を行い、結果を汎用レジスタR0へ返します。 
		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRmの内容と汎用レジスタRnの内容との論理演算(論理積、論理和、排他的論理和)を行い、結果を汎用レジスタRnへ返します。 

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
NOT		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRmのビットを反転した結果を汎用レジスタRnへ格納します。 

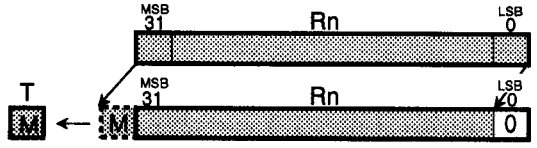
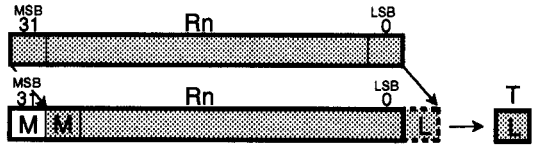
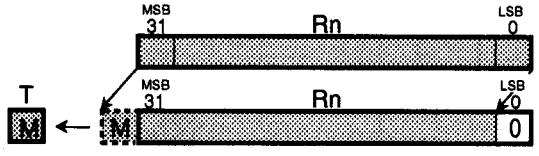
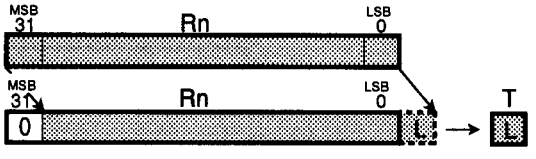
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
TST	B	#imm:8	@(R0,GBR)	@(R0,GBR)のメモリの内容と8ビット定数データの論理積をとり、結果が0のときはTビットをセットし、結果が0以外のときはTビットをクリアします。結果はどこにも返しません。内蔵周辺モジュールレジスタのビットテストに使用すると便利です。 
		#imm:8	R0に限定	汎用レジスタR0の内容と8ビット定数データ(ゼロ拡張)の論理積をとり、結果が0のときはTビットをセットし、結果が0以外のときはTビットをクリアします。結果はどこにも返しません。 
		Rm (R0 ～ R15)	Rn (R0 ～ R15)	汎用レジスタRmの内容と汎用レジスタRnの内容との論理積をとり、結果が0のときはTビットをセットし、結果が0以外のときはTビットをクリアします。結果はどこにも返しません。 

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
TAS			@Rn	@Rnのメモリの内容(8ﾋﾞｯﾄ)を読み込み、0のときはTビットをセットし、0以外のときはTビットをクリアします。その後、ビット7をセットして書き込みます。本命令実行中は、バス権は開放しません。 

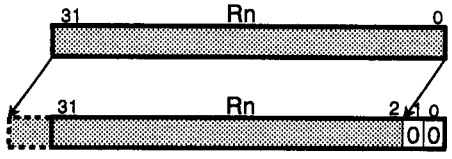
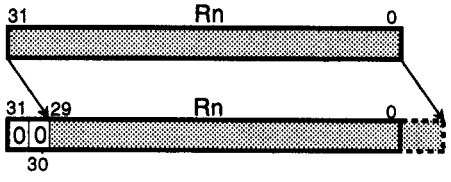
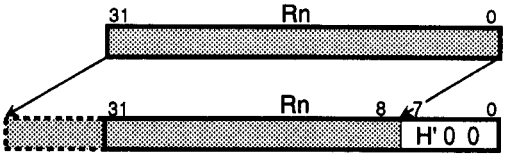
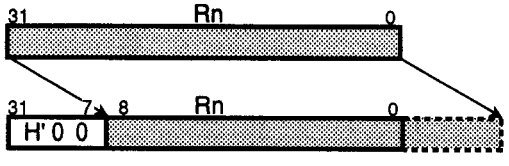
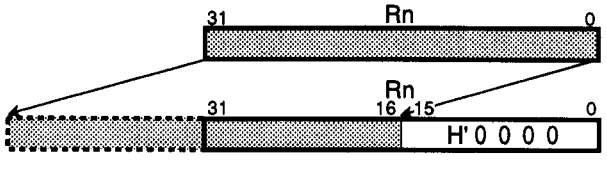
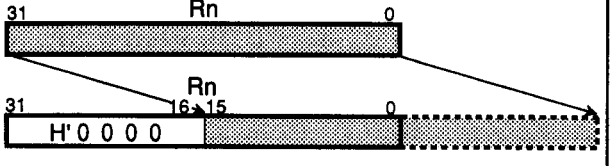
ROTL　ROTR　ROTCL　ROTCR

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
ROTL			Rn (R0 ～ R15)	汎用レジスタRnの内容を左方向へ1ビットローテート(回転)します。また、ローテートにより外に出たMSBの内容をTビットに示します。 
ROTR				汎用レジスタRnの内容を右方向へ1ビットローテート(回転)します。また、ローテートにより外に出たLSBの内容をTビットに示します。 
ROTCL				汎用レジスタRnの内容をTビットの内容を含めて左方向へ1ビットローテート(回転)します。 
ROTCR				汎用レジスタRnの内容をTビットの内容を含めて右方向へ1ビットローテート(回転)します。 

SHAL SHAR SHLL SHLR

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
SHAL			Rn (R0 ～ R15)	汎用レジスタRnの内容を左方向へ算術的に1ビットシフトします。シフトアウトしたMSBの内容はTビットへ格納し、空いたLSBには0を格納します。有符号値の左シフトに使用します。 
SHAR				汎用レジスタRnの内容を右方向へ算術的に1ビットシフトします。シフトアウトしたLSBの内容はTビットへ格納し、空いたMSBにはMSBの内容を格納します。有符号値の右シフトに使用します。 
SHLL				汎用レジスタRnの内容を左方向へ算術的に1ビットシフトします。シフトアウトしたMSBの内容はTビットへ格納し、空いたLSBには 0 を格納します。 
SHLR				汎用レジスタRnの内容を右方向へ算術的に1ビットシフトします。シフトアウトしたLSBの内容はTビットへ格納し、空いたMSBには 0 を格納します。 

SHLL2 SHLR2 SHLL8 SHLR8 SHLL16 SHLR16

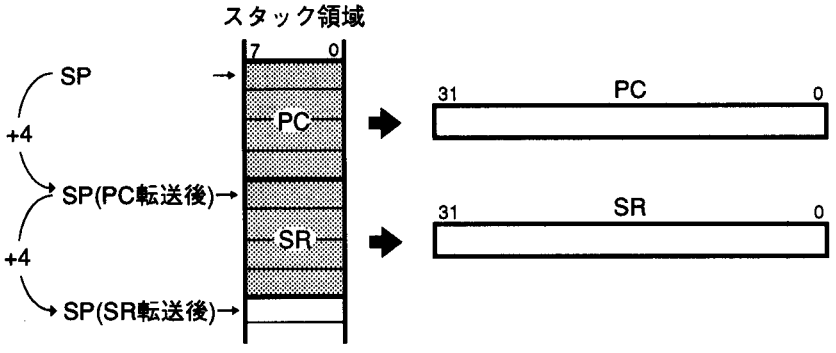
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
SHLL2			Rn (R0 ～ R15)	汎用レジスタRnの内容を左方向へ論理的に2ビットシフトします。シフトアウトしたビットは捨て、空いたビットには0を格納します。 
SHLR2				汎用レジスタRnの内容を右方向へ論理的に2ビットシフトします。シフトアウトしたビットは捨て、空いたビットには0を格納します。 
SHLL8				汎用レジスタRnの内容を左方向へ論理的に8ビットシフトします。シフトアウトしたビットは捨て、空いたビットには0を格納します。 
SHLR8				汎用レジスタRnの内容を右方向へ論理的に8ビットシフトします。シフトアウトしたビットは捨て、空いたビットには0を格納します。 
SHLL16				汎用レジスタRnの内容を左方向へ論理的に16ビットシフトします。シフトアウトしたビットは捨て、空いたビットには0を格納します。 
SHLR16				汎用レジスタRnの内容を右方向へ論理的に16ビットシフトします。シフトアウトしたビットは捨て、空いたビットには0を格納します。 

BF BT BRA BSR JMP JSR RTS

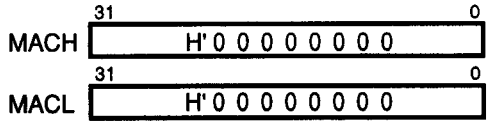
ニーモニック	オペランド	説明
BF	disp:8 または 分岐先のシンボル	Tビットを参照する条件付き分岐命令です。T=0のときは分岐先アドレス（disp:8）をPCへ転送し、T=1のときは何もしません。 分岐先に届かないときは、BRA命令またはJMP命令と組み合わせて対応する必要があります。 T=0のとき 分岐先アドレス（disp:8） → PC（31〜0） T=1のとき NOP
BT		Tビットを参照する条件付き分岐命令です。T=0のときは何もせず、T=1のときは分岐先アドレス（disp:8）をPCへ転送します。 分岐先に届かないときは、BRA命令またはJMP命令と組み合わせて対応する必要があります。 T=0のとき NOP T=1のとき 分岐先アドレス（disp:8） → PC（31〜0）
BRA	disp:12 または 分岐先のシンボル	無条件の分岐命令です。分岐先アドレス（disp:12）をPCへ転送します。分岐先に届かないときは、JMP命令を使用します。本命令は遅延分岐命令です。 分岐先アドレス（disp:12） → PC（31〜0）
BSR	disp:12 または サブルーチンの先頭アドレスのシンボル	サブルーチンへ分岐する命令です。PCの内容(サブルーチンからの復帰先アドレス)をPRへ退避し、サブルーチンの先頭アドレス（disp:12）をPCへ転送します。サブルーチンからの復帰先アドレスは本命令の2命令後のアドレスです。分岐先に届かないときは、JSR命令を使用します。本命令は遅延分岐命令です。 PC（31〜0）：サブルーチンからの復帰先アドレス → PR（31〜0） サブルーチンの先頭アドレス（disp:12）
JMP	@Rn	無条件の分岐命令です。分岐先アドレス（@Rn）をPCへ転送します。本命令は遅延分岐命令です。 分岐先アドレス（@Rn） → PC（31〜0）
JSR	@Rn	サブルーチンへ分岐する命令です。PCの内容(サブルーチンからの復帰先アドレス)をPRへ退避し、サブルーチンの先頭アドレス（@Rn）をPCへ転送します。サブルーチンからの復帰先アドレスは本命令の2命令後のアドレスです。本命令は遅延分岐命令です。 PC（31〜0）：復帰先アドレス → PR（31〜0） サブルーチンの先頭アドレス（@Rn）
RTS		サブルーチンから復帰する命令です。PRの内容(サブルーチンからの復帰アドレス)をPCへ転送します。本命令は遅延分岐命令です。 PR（31〜0）：サブルーチンからの復帰先アドレス → PC（31〜0）

RTE | CLRMAC | SLEEP | NOP | CLRT | SETT

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
RTE				割り込みルーチンからの復帰命令です。PCとSRをスタックから復帰させます。本命令は遅延分岐命令です。



ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
CLRMAC				MACHおよびMACLをクリアします。

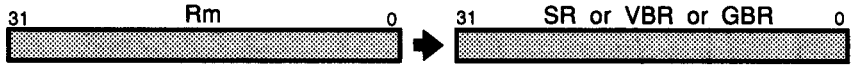
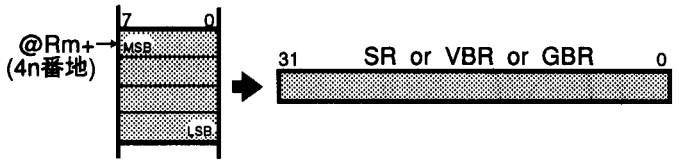
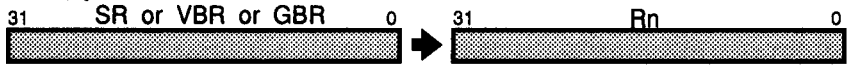
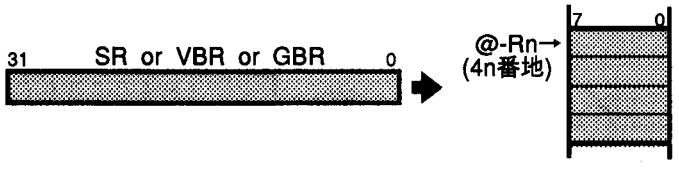


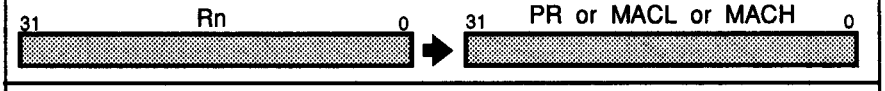
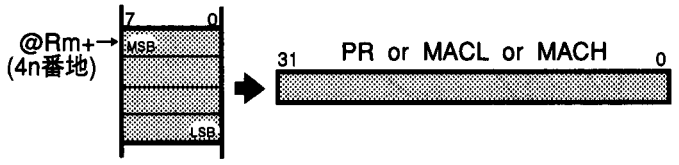
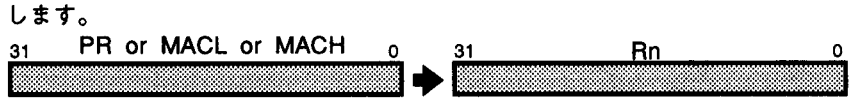
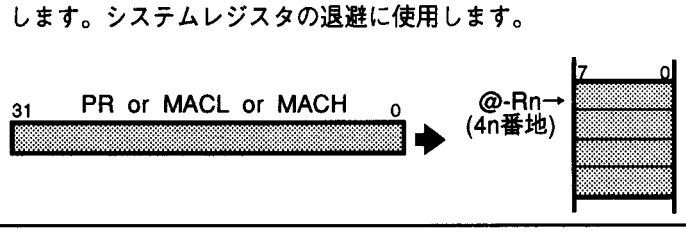
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
SLEEP				CPUを低消費電力状態にします。低消費電力状態では、CPUの内部状態を保持し、直後の命令の 実行を停止し、割り込み要求の発生を待ちます。割り込み要求が発生すると、低消費電力状態から抜けます。

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
NOP				PCのインクリメント(+2)のみを行います。CPUの内部状態には影響を与えません。

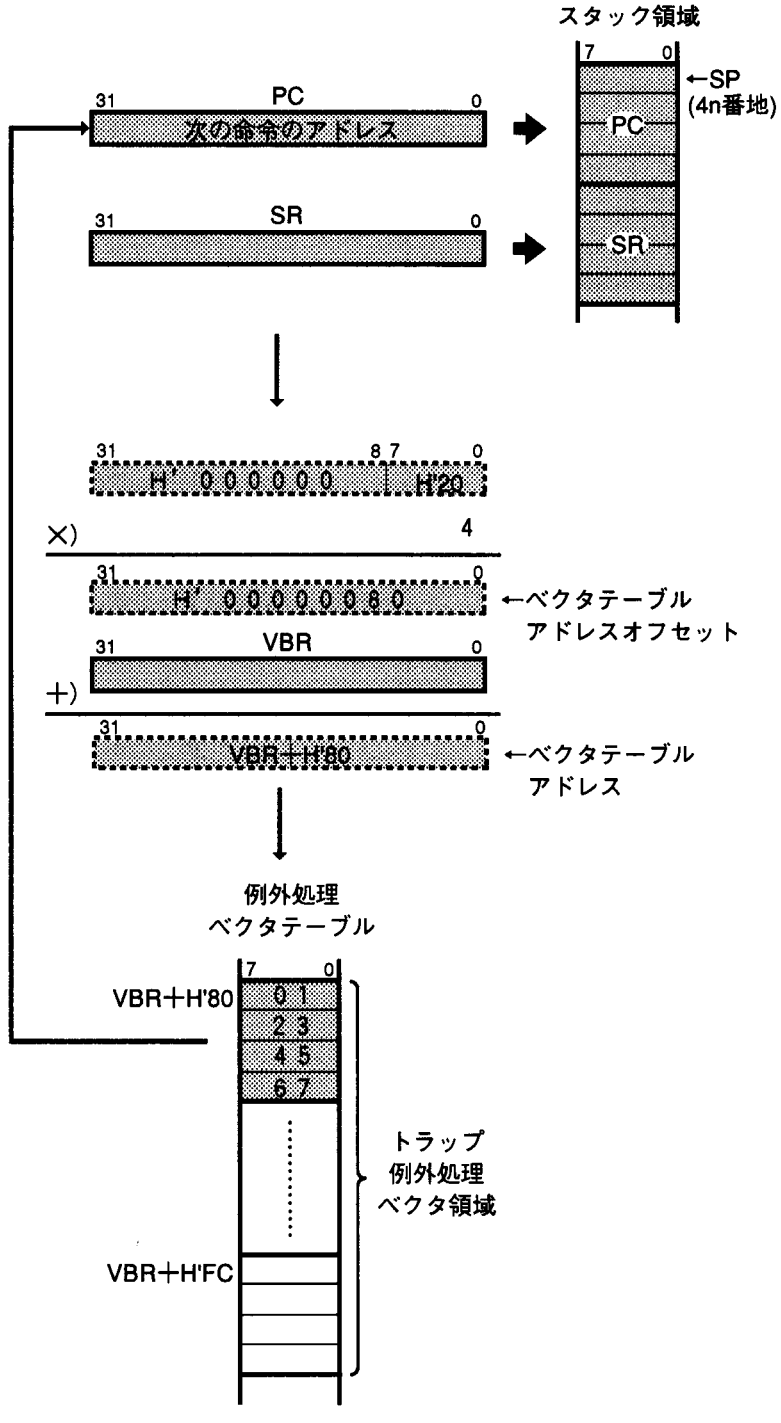
ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
CLRT				Tビットをクリアします。

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
SETT				Tビットをセットします。

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
LDC		Rm (R0 ～ R15)	SR VBR GBR	汎用レジスタRmの内容をコントロールレジスタ(SR、VBRまたはGBR)へ転送します。 
	L	@Rn+		@Rn+のメモリの内容をコントロールレジスタ(SR、VBRまたはGBR)へ転送します。コントロールレジスタの復帰に使用します。 
STC		SR VBR GBR	Rn (R0 ～ R15)	コントロールレジスタ(SR、VBRまたはGBR)の内容を汎用レジスタRnへ転送します。 
	L		@-Rn	コントロールレジスタ(SR、VBRまたはGBR)の内容を@-Rnのメモリへ転送します。コントロールレジスタの退避に使用します。 

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
LDS		Rm (R0 ～ R15)	PR MACL MACH	汎用レジスタRmの内容をシステムレジスタ(PR、MACL、MACH)へ転送します。但し、MACHには汎用レジスタRmの下位10ビットの内容を符号拡張した値が格納されます。 
	L	@Rm+		@Rn+のメモリの内容をシステムレジスタ(PR、MACLまたはMACH)へ転送します。システムレジスタの復帰に使用します。 
STS		PR MACL MACH	Rn (R0 ～ R15)	システムレジスタ(PR、MACLまたはMACH)の内容を汎用レジスタRnへ転送します。 
	L		@-Rn	システムレジスタ(PR、MACLまたはMACH)の内容を@-Rnのメモリへ転送します。システムレジスタの退避に使用します。 

ニーモニック	オペランドサイズ	ソースオペランド	デスティネーションオペランド	説明
TRAPA			#imm:8	トラップ例外処理へ分岐する命令です。PC(次の命令のｱﾄﾞﾚｽ)とSRの内容をスタックへ退避させ、VBRに8ビット定数データをゼロ拡張後4倍した値(ﾍﾞｸﾀﾃｰﾌﾞﾙｱﾄﾞﾚｽｵﾌｾｯﾄ)を加算し、加算結果(ﾍﾞｸﾀﾃｰﾌﾞﾙｱﾄﾞﾚｽ)が示すアドレスの内容(ﾄﾗｯﾌﾟ例外処理の先頭ｱﾄﾞﾚｽ)をPCへ転送します。RTEと組み合わせて、システムコールに使用します。

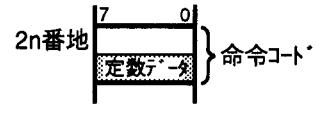
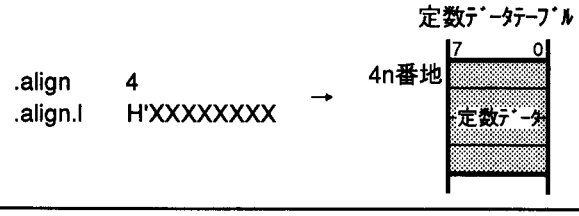


1．5　定数データテーブルの配置方法

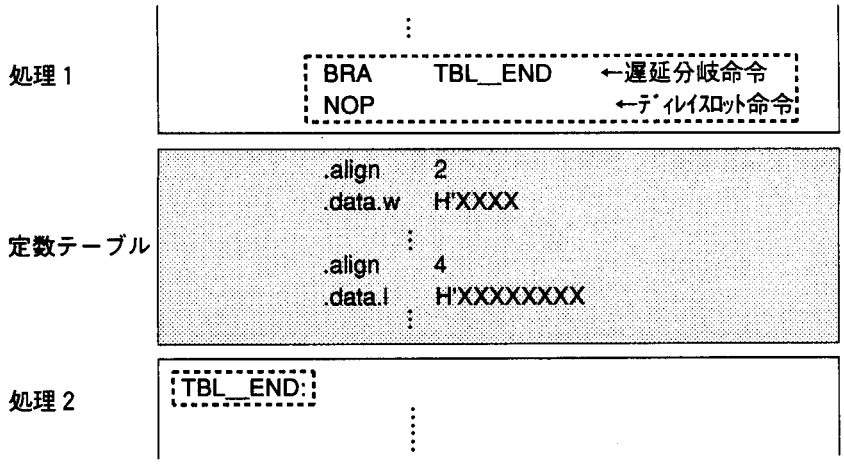
SH7000シリーズCPUの命令コードは16ビット固定長であるため、16ビットおよび32ビットの定数データは、8ビット定数データのように命令コード上に配置できません。

このため、16ビットおよび32ビットの定数データテーブルを用意し、ディスプレースメント付きPC相対アドレッシングにより定数データテーブル上の16ビットおよび32ビットの定数データを参照する方法をとります。表1.11に定数データの配置場所についての詳細を示します。

表1.11　各定数データの配置場所

<table>
<tr><th>定数ﾃﾞｰﾀ長</th><th>定数ﾃﾞｰﾀ参照
ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ</th><th>配　置　場　所</th></tr>
<tr><td>8ビット</td><td>イミディエイト</td><td>8ビットの定数データは、命令コードの2バイト目に配置されます。
 
</td></tr>
<tr><td>16ビット</td><td rowspan="2">ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付き
PC相対</td><td>16ビットの定数データは、定数データテーブル上の2n番地に配置します。
 
</td></tr>
<tr><td>32ビット</td><td>32ビットの定数データは、定数データテーブル上の4n番地に配置します。
 
</td></tr>
</table>

定数データテーブルの配置場所は、図1.6に示すように処理と処理の間に配置した場合は無条件分岐命令が必要になります。そこで、図1.7および図1.8の示すようにサブルーチンの直後または繰り返し処理の直後に配置すると効率の良いプログラミングが可能です。



：定数データテーブルを飛び越える分岐処理

図1.6　定数データテーブル配置場所（1）

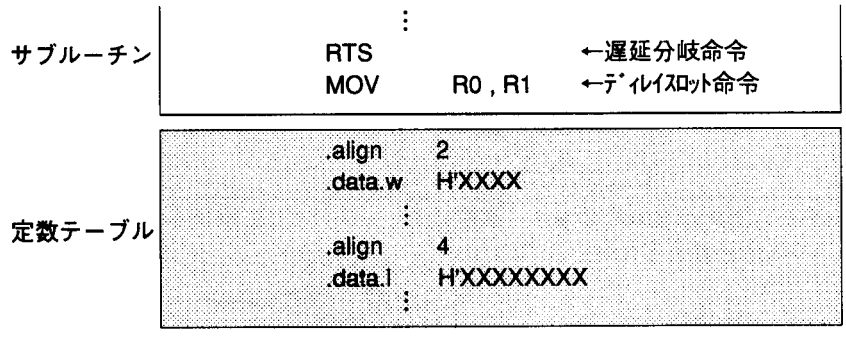


図1.7　定数データテーブル配置場所（2）

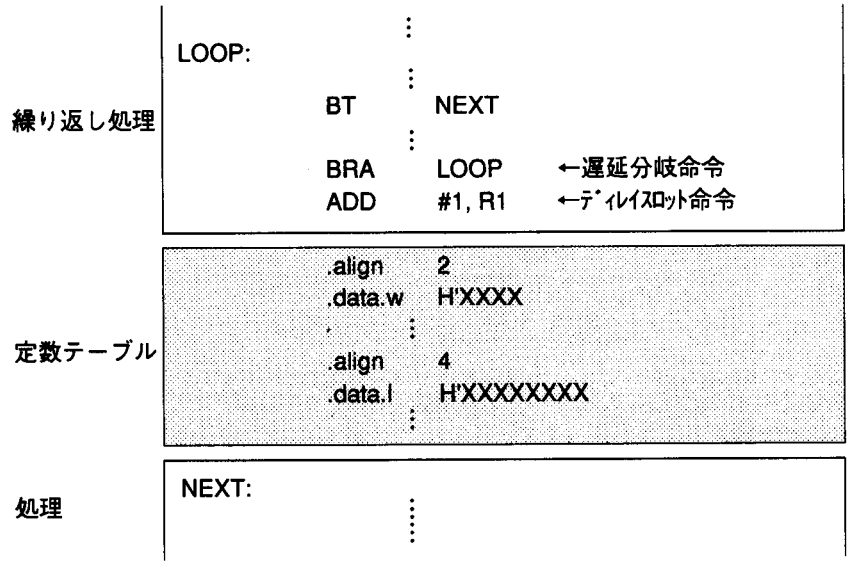


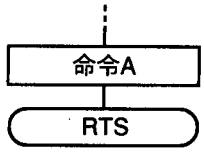
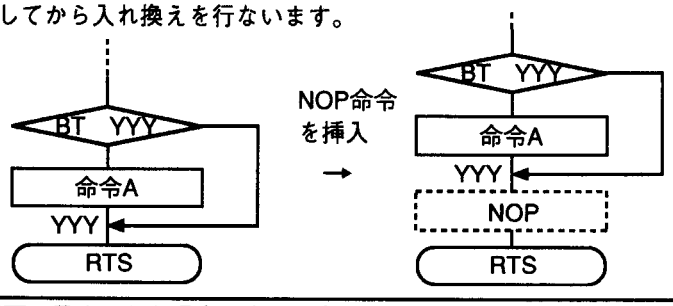
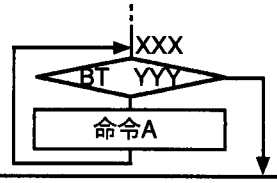
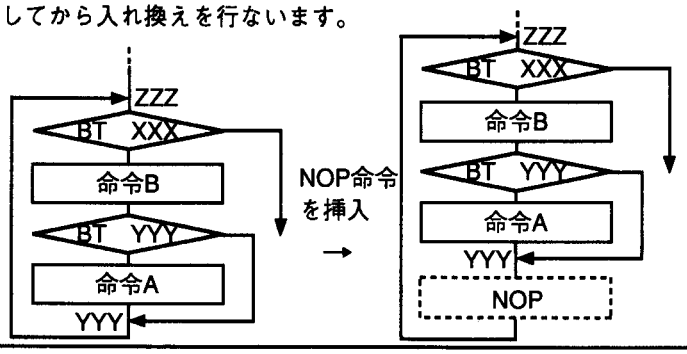
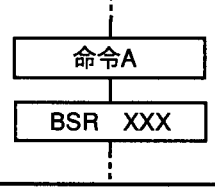
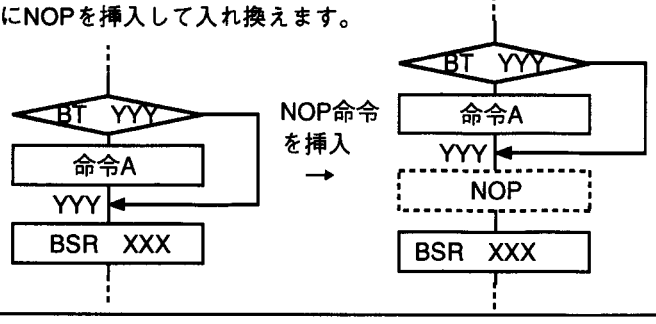
図1.8　定数データテーブル配置場所（3）

1．6　遅延分岐命令について

SH7000シリーズCPUでは、分岐時のパイプラインの乱れを軽減するため、無条件分岐命令(BRA、BSR、JMP、JSR、RTS、RTE)は遅延分岐命令となっています。遅延分岐命令は、直後の命令(ディレイスロット命令)を実行してから分岐を行います。表1.12に遅延分岐命令のソースプログラムでの記述を示します。

また、ディレイスロット命令が遅延分岐命令またはBF、BT、TRAPA、MOVA、MOV　@(disp,PC),Rnの場合、エラーまたはウォーニングとなります。詳細については、SHシリーズクロスアセンブラマニュアルの遅延分岐命令に関する注意をご覧ください。

表1.12　ソースプログラムの記述

遅延分岐命令	説　明	ソースプログラム記述
RTS、RTE	以下に示すような場合、直前の命令と入れ換えを行ないます。 	⋮ RTS 命令A←ﾃﾞｨﾚｲｽﾛｯﾄ命令
	RTS(またはRTE)が分岐先の場合、直前にダミー命令としてNOPを挿入してから入れ換えを行ないます。 	⋮ BT　YYY 命令A YYY　RTS NOP←ﾃﾞｨﾚｲｽﾛｯﾄ命令
JMP、BRA	以下に示すような場合は直前の命令と入れ換えます。 	⋮ BT　YYY BRA　XXX 命令A←ﾃﾞｨﾚｲｽﾛｯﾄ命令
	BRA(またはJMP)が分岐先の場合、直前にダミー命令としてNOPを挿入してから入れ換えを行ないます。 	⋮ ZZZ　BT　XXX 命令B BT　YYY 命令A YYY　BRA　ZZZ NOP←ﾃﾞｨﾚｲｽﾛｯﾄ命令
JSR、BSR	以下に示すような場合は直前の命令と入れ換えます。 	⋮ BSR　XXX 命令A←ﾃﾞｨﾚｲｽﾛｯﾄ命令 ⋮
	BSR(またはJSR)が分岐先となっている場合、BSR(またはJSR)の直前にNOPを挿入して入れ換えます。 	⋮ BT　YYY 命令A YYY　BSR　XXX NOP←ﾃﾞｨﾚｲｽﾛｯﾄ命令 ⋮

1．7　内蔵周辺モジュールレジスタ領域アクセス方法

SH7000シリーズCPUには内蔵周辺モジュールレジスタ領域のベースアドレスを設定するGBRレジスタがあります。これにより、内蔵周辺モジュールレジスタのアドレスは、GBRに設定されたベースアドレスに対するオフセット値から算出することが出来ます。このようなアドレッシングモードとして、ディスプレースメント付きGBR間接およびインデックス付きGBR間接があります。ディスプレースメント付きGBR間接はMOV命令を、インデックス付きGBR間接は論理演算命令をそれぞれサポートしており、データ転送およびビット操作において使用します。

以下にGBRレジスタにH'5FFFF00を設定した場合のデータ転送およびビット操作を説明します。

（1）データ転送

データ転送を行う内蔵周辺モジュールレジスタのアドレス指定は、図1.9に示すように、オフセット値が−256〜−129、−128〜−1および0〜+255の各領域において異なります。また、アクセスサイズが制限されているレジスタがあります。詳細はハードウェアマニュアルを参照してください。

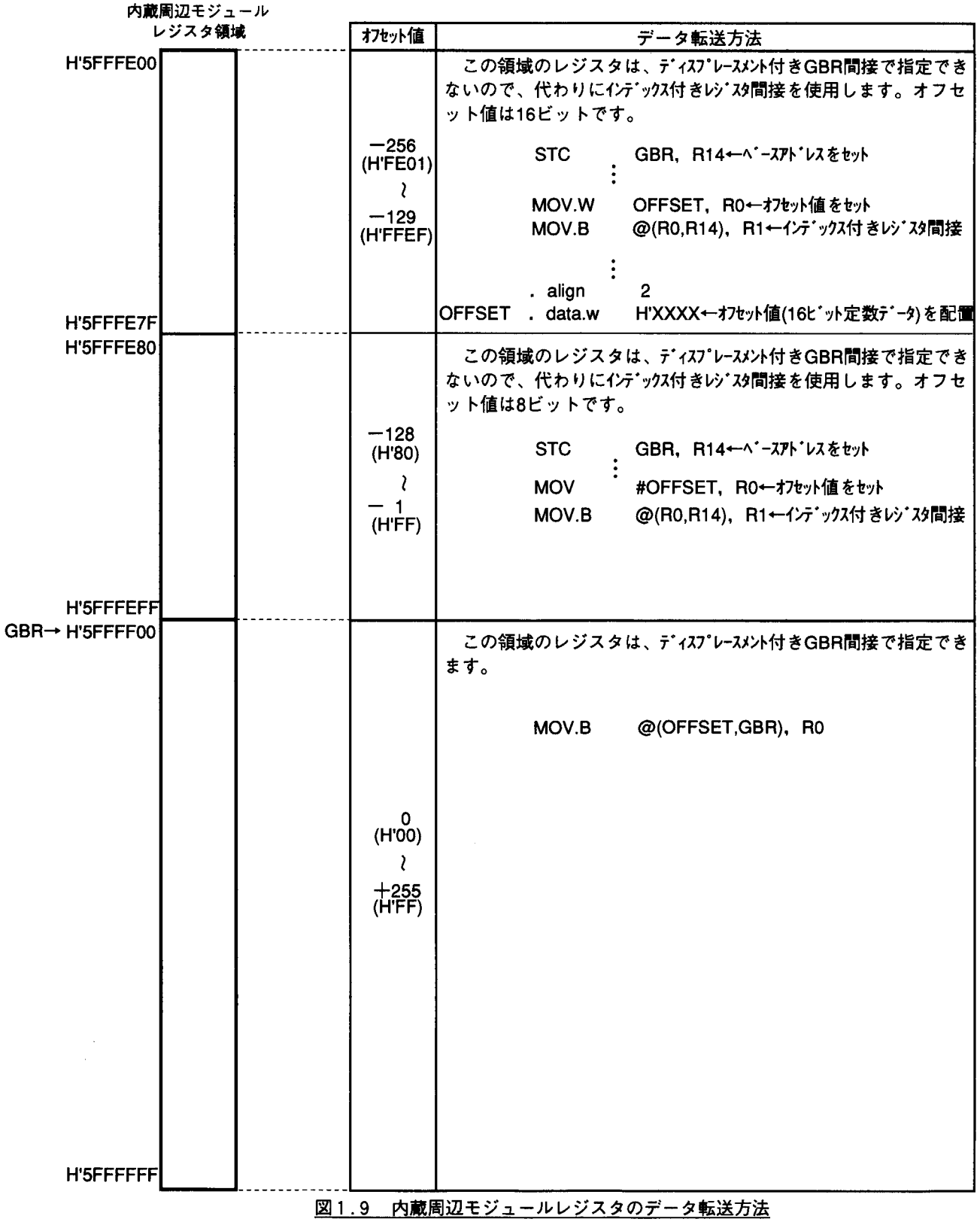


図1.9　内蔵周辺モジュールレジスタのデータ転送方法

（2）ビット操作

ビット操作を行う内蔵周辺モジュールレジスタのアドレス指定はインデックス付きGBR間接を用いますが、図1.10に示すように、オフセット値が−256〜−129、−128〜+127および+128〜+255の各領域においてオフセット値のデータ長が異なります。

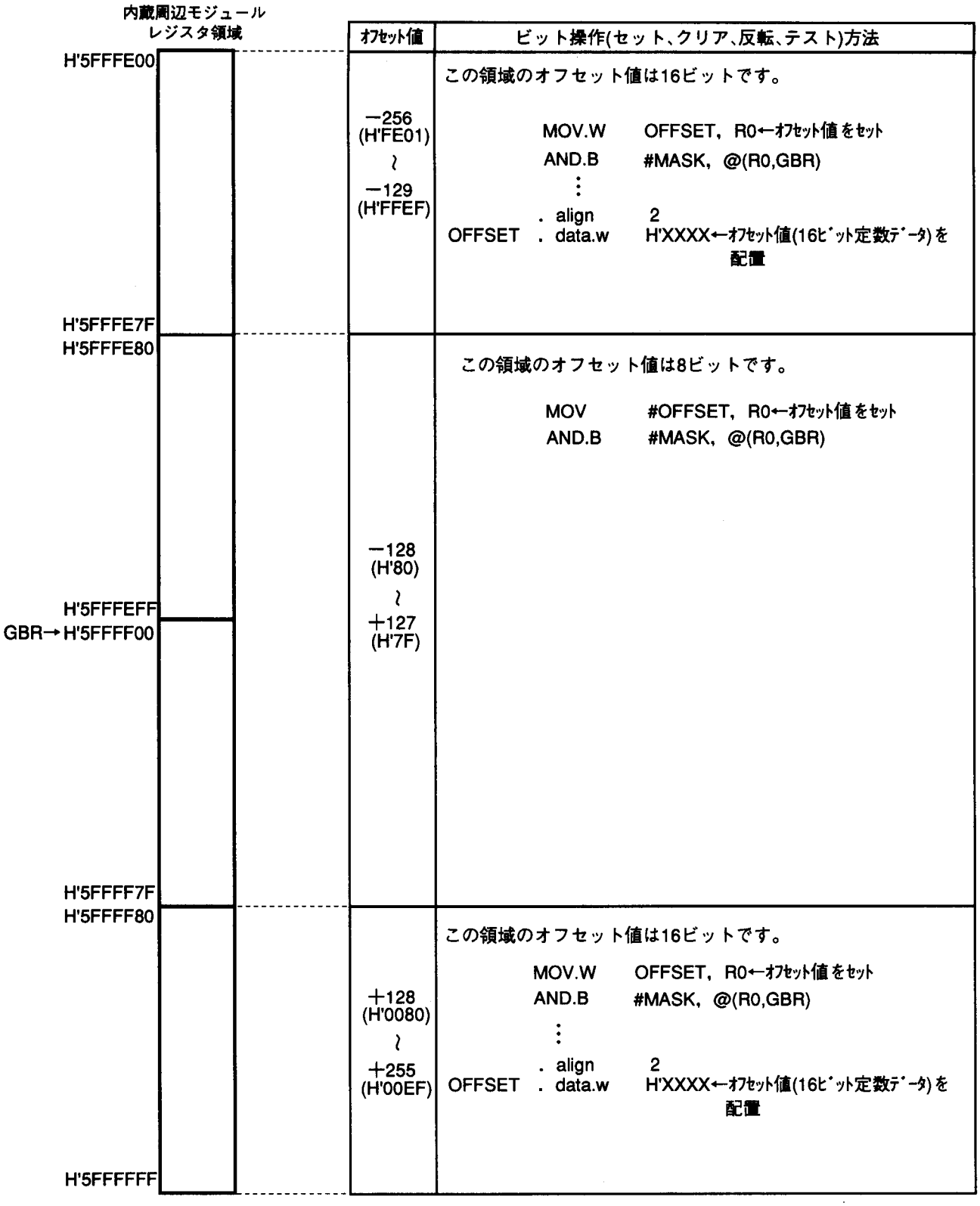


図1.10　内蔵周辺モジュールレジスタのビット操作方法

1．8　サブルーチンコールに関する注意

SH7000シリーズCPUでは、JSR命令またはBSR命令によりサブルーチンコールした場合、サブルーチンからの戻り先アドレスはPRレジスタへ自動的に退避されます。スタック領域と違いPRレジスタには戻り先アドレスを1つしか退避させることができません。このため、サブルーチン内で他のサブルーチンをコールした場合、コールした側のサブルーチンの戻り先アドレスは破壊されてしまいます。そこで、図1.10に示すように、PRの内容を退避および復帰させる必要があります。退避および復帰には、システム制御命令のSTS命令およびLDS命令を使用します。退避先は、メモリ(スタック領域等)または汎用レジスタを指定できます。

また、　割込みルーチン内でサブルーチンをコールしている場合においても、図1.11と同様にPRレジスタの退避および復帰を行ってください。

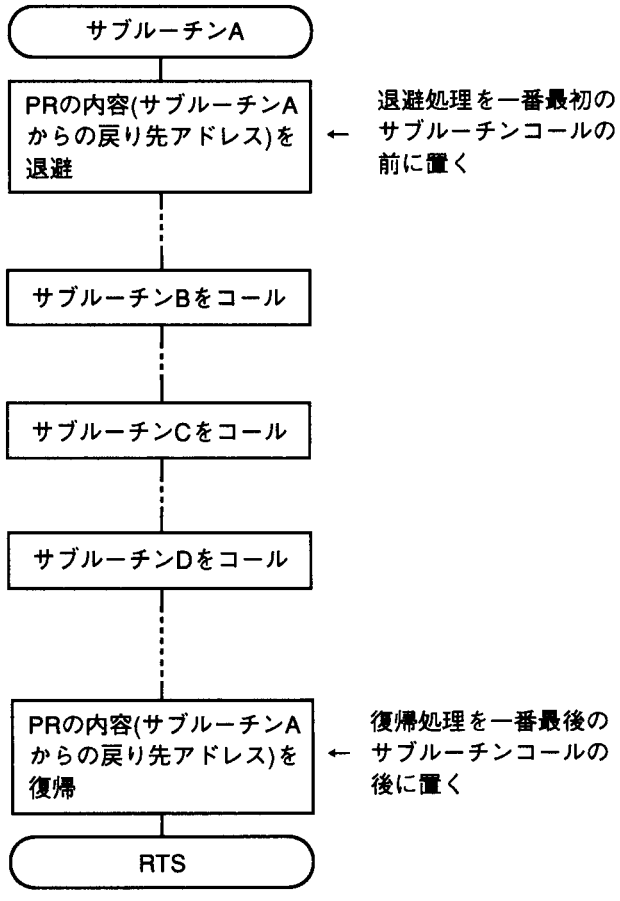


図1.11　PRの退避および復帰

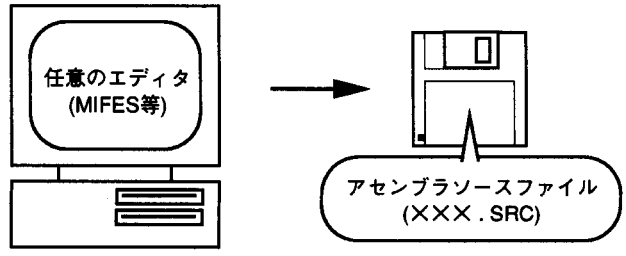
2. ロードモジュール変換手順

第2章　目次

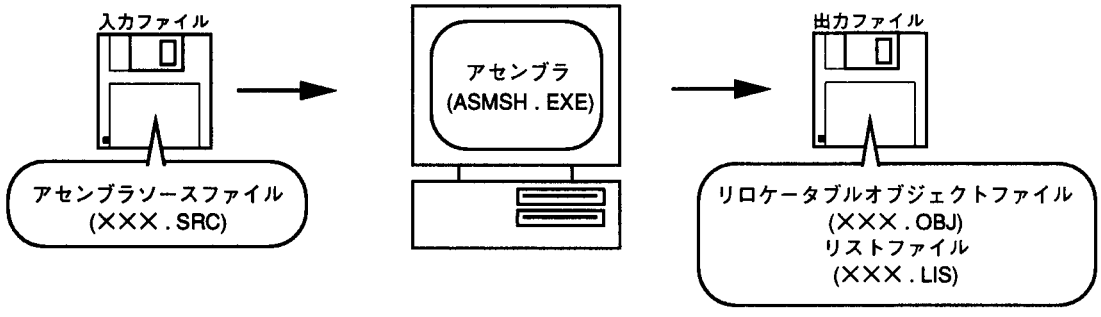
2．1　ロードモジュール変換手順

以下にSH7000シリーズCPUのロードモジュール変換手順を示します。

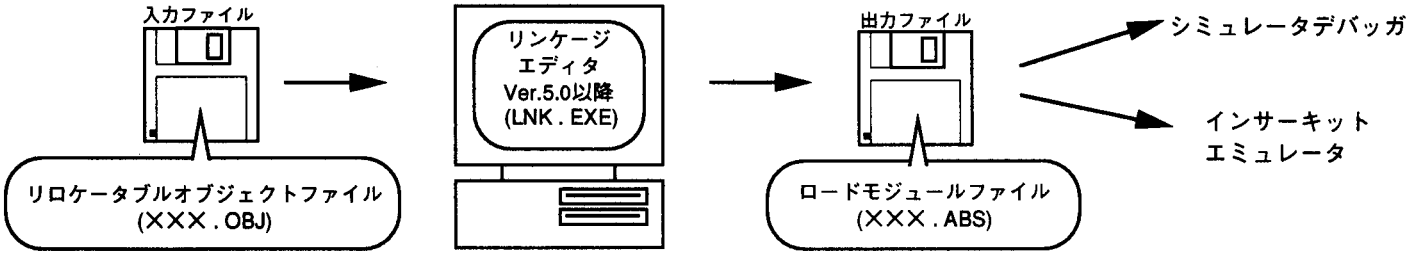
（1）任意のエディタ(MIFES等)によってアセンブラソースプログラムを作成します。



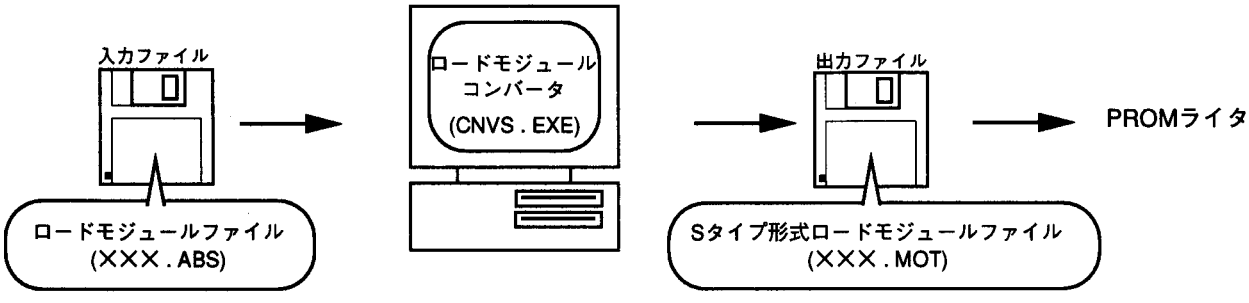
（2）アセンブラ(ASMSH．EXE)によって、アセンブラソースプログラムをオブジェクトモジュールに変換します。



（3）リンケージエディタ(LNK．EXE)によって、オブジェクトモジュールをロードモジュールに変換します。但し、リンケージエディタはVer.5.0以降のものを使用してください。



（4）ロードモジュールコンバータ(CNVS．EXE)によって、ロードモジュールをSタイプ形式ロードモジュールに変換します。



3. ソフトウェア応用例

第3章　目次

3．1　ソフトウェア例の一覧表

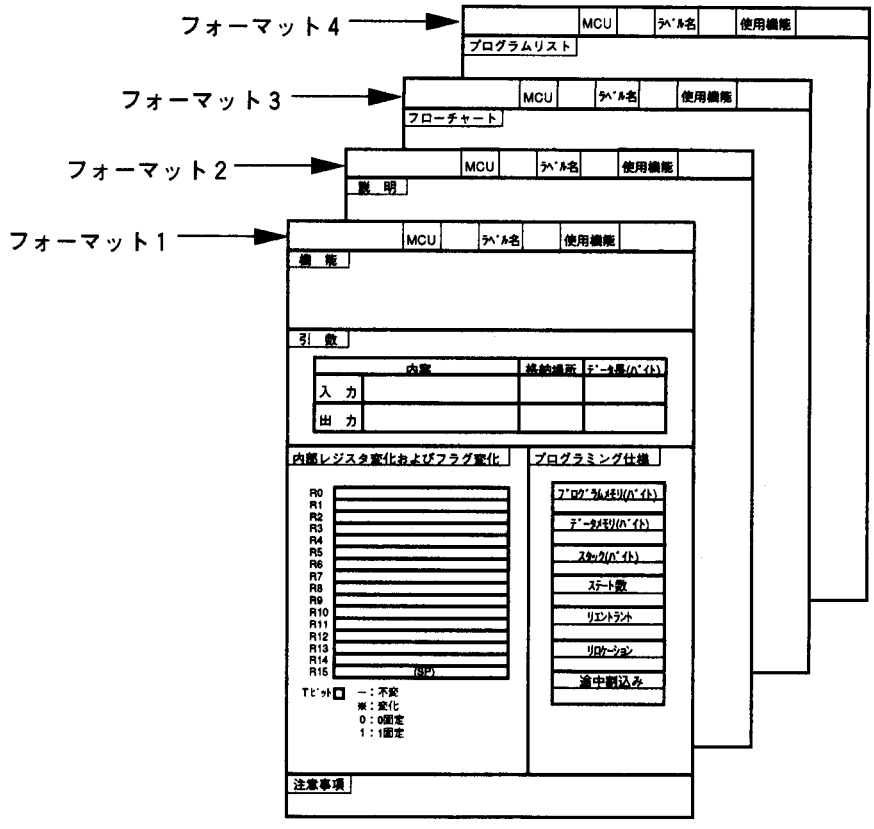
ソフトウェア例の一覧表

ソフトウェア名	ラベル名	使用機能	参照頁
ブロック転送（4バイト整合されていない）	MOVE	MOV．B命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付 きﾚｼﾞｽﾀ間接	59
ブロック転送（4バイト整合されている）	MOVE4	MOV．L命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付 きﾚｼﾞｽﾀ間接	69
32ビットデータの多ビットシフト（右算術シフト）	SHARN	SHLR2 命令 SHLR8 命令 SHLR16 命令	79
32ビットデータの多ビットシフト（右論理シフト）	SHLRN	SHLR2 命令 SHLR8 命令 SHLR16 命令	85
32ビットデータの多ビットシフト（左論理シフト）	SHLLN	SHLL2 命令 SHLL8 命令 SHLL16 命令	90
32ビットデータの最初の1の検出（Find First 1）	FIND1	SHLL 命令	95
64ビット＋64ビット=64ビット（符号なし）	ADDU64	ADDC 命令	100
64ビット＋64ビット=64ビット（符号付き）	ADDS64	ADDV 命令	105
32ビット×32ビット=64ビット（符号なし）	MULU32	MULU 命令 SWAP 命令	110
32ビット×32ビット=64ビット（符号付き）	MULS32	MULU 命令 SWAP 命令 NEGC 命令	115
32ビット÷32ビットの商（符号なし）	DIVU32Q	DIV0U 命令 DIV1 命令	122
32ビット÷32ビットの剰余（符号なし）	DIVU32R	DIV0U 命令 DIV1 命令	129
32ビット÷32ビットの商（符号付き）	DIVS32Q	DIV0S 命令 DIV1 命令	136
32ビット÷32ビットの剰余（符号付き）	DIVS32R	DIV0S 命令 DIV1 命令	143
アフィン変換	AFIN	MAC．W命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接	152

3．2　ソフトウェア例使用手引

1．フォーマット説明

図3.1にソフトウェア例のフォーマットを示します。図3.1に示すように、ソフトウェア例はフォーマット1～4の構成になっています。



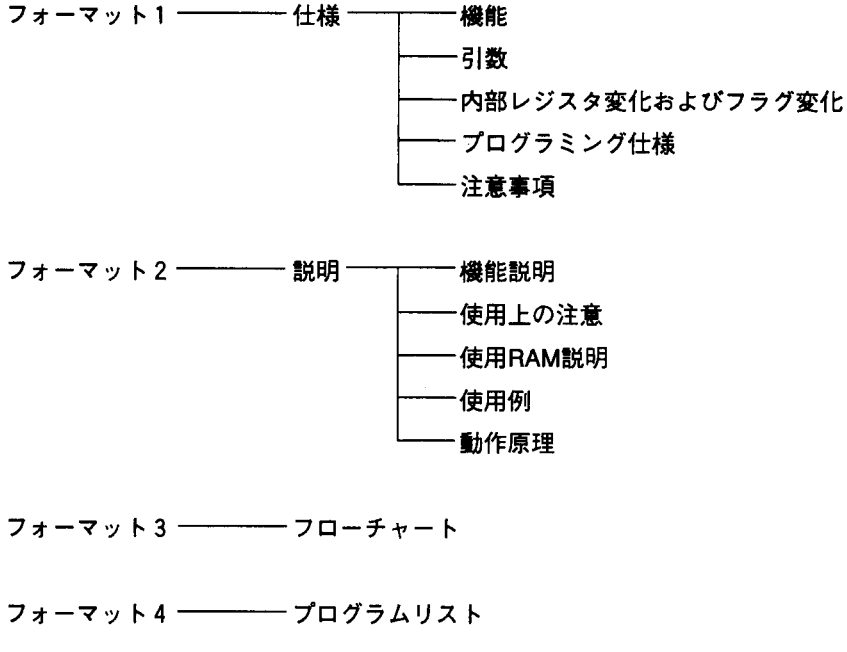


図3.1　ソフトウェア例のフォーマットおよび構成

2．仕様部のフォーマット（フォーマット1）

図3.2に仕様部のフォーマットを示します。仕様部はソフトウェア例の機能および仕様などについて紹介しています。

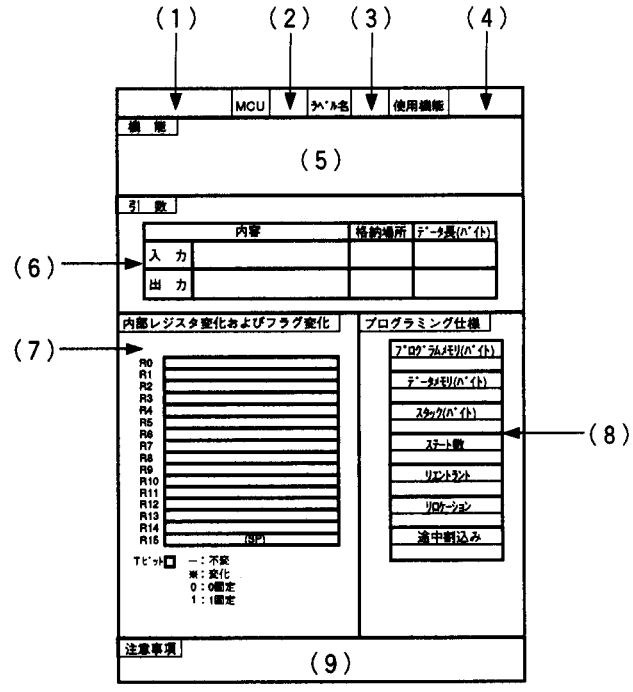


図3.2　仕様部のフォーマット

（1）ソフトウェア名
　ソフトウェア名を示しています。

（2）MCU
　本ソフトウェア例を適用できるマイクロコンピュータのシリーズ名を示しています。

（3）ラベル名
　ソフトウェア例のエントリポイントのラベル名を示します。
　ソフトウェア例をそのままサブルーチンとして使用する場合は、このラベル名をサブルーチンコールして下さい。

（4）使用機能
　本ソフトウェア例で使用しているアドレッシングモードおよび命令を示しています。

（5）機　能
　ソフトウェア例の機能について説明しています。

（6）引　数
　ソフトウェア例を実行する際に必要な入力引数と実行後の出力引数について説明しています。
　(a) 内　　容：入出力引数の内容を説明しています。
　(b) 格納場所：入出力引数をセットするレジスタを示します。
　(c) データ長（バイト）：入出力引数の使用バイト数を示します。

（7）内部レジスタ変化およびフラグ変化

ソフトウェア例実行前および実行後の内部レジスタの内容、ソフトウェア例実行後のTビットの内容について説明します。

(a) 内部レジスタ

R0～R15 ：32ビット汎用レジスタ

(b) ソフトウェア例実行後のTビットの状態表示

－：不変：ソフトウェア例実行後もTビットの内容が保存されます。

※：変化：ソフトウェア例実行によりTビットの内容が破壊されます。

0，1：固定：ソフトウェア例実行後、必ず0または1に設定されます。

（8）プログラミング仕様

ソフトウェア例の仕様について説明しています。

(a) プログラムメモリ（バイト）：ソフトウェア例で使用するROMの使用量を示します。

(b) データメモリ（バイト）：ソフトウェア例で使用するRAMの使用量を示します。

(c) スタック（バイト）：ソフトウェア例で使用するスタック容量を示します。
ユーザプログラムでのサブルーチンコールによるスタック容量は含みません。
ソフトウェア例を実行する際には，スタック部に示してあるバイト数分のスタック容量が必要ですので，スタック容量分のデータメモリ容量を確保した上で実行してください。

(d) ステート数：ソフトウェア例をシミュレータデバッカで実行したときの実行ステート数を示します。

(e) リエントラント：複数のプログラムから同時に使用可能な構造になっているかを示します。

(f) リロケーション：ソフトウェア例をどのメモリ空間に配置しても正常に動作するかを示します。

(g) 途中割込み：ソフトウェア例の実行中に割込みルーチンが実行されても，その後正常動作するかを示します。不可の場合は，ソフトウェア例を呼び出す前に割込みを禁止してください。

（9）注意事項

（8）の仕様に書かれている内容についての注意事項を説明しています。

3．説明部のフォーマット（フォーマット2）

図3.3に説明部のフォーマットを示します。説明部は、ソフトウェア例の機能説明、使用上の注意、使用RAM説明、使用例および動作原理を紹介しています。

（1）機能説明
　ソフトウェア例の具体的な実行例をもとに機能を説明しています。
（2）使用上の注意
　ソフトウェア例を使用する場合の注意および制限事項について説明しています。
（3）使用RAM説明
　ソフトウェア例で使用しているRAMのラベル名および内容を示しています。
（4）使用例
　ソフトウェア例を実際に使用する場合の使用例を示しています。
（5）動作原理
　ソフトウェア例の動作原理を説明しています。

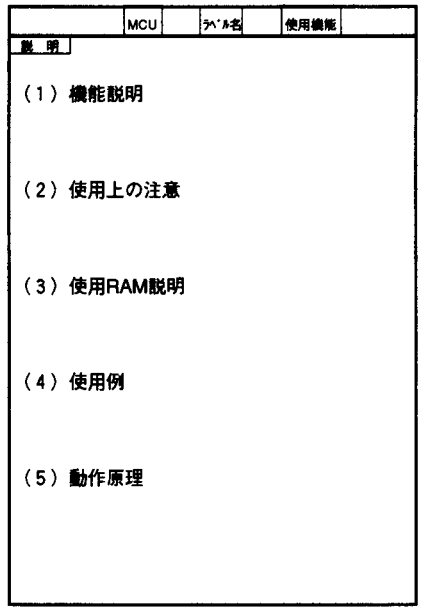


図3.3　説明部のフォーマット

4．フローチャート部のフォーマット（フォーマット3）

図3.4にフローチャート部のフォーマットを示します。フローチャート部は、ソフトウェア例のフローチャートを示します。

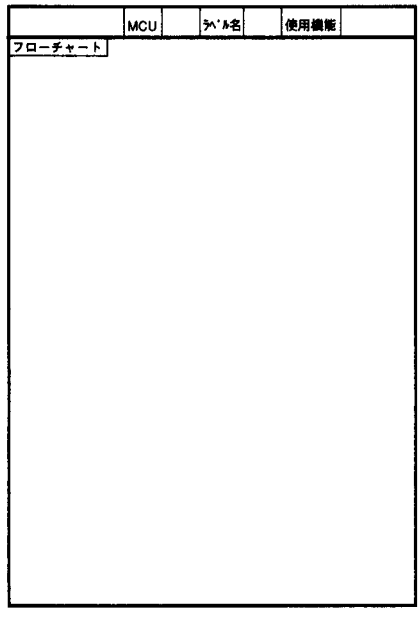


図3.4　フローチャート部のフォーマット

5．プログラムリスト部のフォーマット（フォーマット4）

図3.5にプログラムリスト部のフォーマットを示します。図3.5に示すように、本フォーマットではソフトウェア例で紹介している各ソフトウェアのプログラムリストを掲載しています。

64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDS64	使用機能	ADDV　命令

プログラムリスト

```
1                                   1  ;*************************************************
2                                   2  ;*                                               *
3                                   3  ;*          NAME  :  64 BIT SIGNED ADDITION (ADDS64)  *
4                                   4  ;*                                               *
5                                   5  ;*************************************************
6                                   6  ;*                                               *
7                                   7  ;*           ENTRY  :  R0    (UPPER 32 BIT AUGEND)     *
8                                   8  ;*                     R1    (LOWER 32 BIT AUGEND)     *
9                                   9  ;*                     R2    (UPPER 32 BIT ADDEND)     *
10                                  10 ;*                     R3    (LOWER 32 BIT ADDEND)     *
11                                  11 ;*         RETURNS  :  R0    (UPPER 32 BIT SUM)        *
12                                  12 ;*                     R1    (LOWER 32 BIT SUM)        *
13                                  13 ;*                     T BIT (OVERFLOW/UNDERFLOW -> TRUE ; T=1,FALSE ; T=0)  *
14                                  14 ;*                                               *
15                                  15 ;*************************************************
16 00001000                         16       (7) .SECTION  A,CODE,LOCATE=H'1000
17          00001000                17 ADDS64 (8) .EQU      $          ; Entry  point
18 00001000 2F46                    18            MOV.L     R4,@-R15   ; Escape  register
19 00001002 2F56                    19            MOV.L     R5,@-R15   ;
20 00001004 0008                    20            CLRT                 ; Clear  T  bit
21 00001006 313E                    21            ADDC      R3,R1      ; Lower  32  bit  augend  +  Lower  32  bit  addend
22 00001008 0429                    22            MOVT      R4         ; R4  <-  Carry
23 0000100A 304F                    23            ADDV      R4,R0      ; Upper  32  bit  augend  +  Carry
24 0000100C 0429                    24            MOVT      R4         ; R4  <-  Overflow / Underflow
25 0000100E 302F                    25            ADDV      R2,R0      ; Upper  32  bit  augend  +  Upper  32  bit  addend
26 00001010 0529                    26            MOVT      R5         ; R5  <-  Overflow / Underflow
27 00001012 245B                    27            OR        R5,R4      ; R4  <-  R5  or  R4
28 00001014 4401                    28            SHLR      R4         ; T  bit  <-  Overflow / Underflow
29 00001016 65F6                    29            MOV.L     @R15+,R5   ; Return  register
30 00001018 000B                    30            RTS                  ;
31 0000101A 64F6                    31            MOV.L     @R15+,R4   ;
32                                  32       (9) .END
(1) (2)  (3)                        (4)           (5)                  (6)
```

図3.5　プログラムリスト部のフォーマット

（1）リスト行番号
（2）ロケーションカウンタ値
（3）オブジェクトコード
（4）ソース行番号
（5）ソースステートメント
（6）コメント
（7）～（9）アセンブラ制御命令(詳細は添付資料を参照して下さい)

3．2．1　ブロック転送（4バイト整合されていない）

ブロック転送 (4バイト整合されていない)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE	使用機能	MOV．B 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

機　能

ブロックデータの転送を行ないます。ブロックデータの転送元領域および転送先領域の先頭アドレスは任意アドレス、ブロックデータは任意バイトを設定可能です。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	転送バイト数	R0	4
	転送先領域の先頭アドレス	R1	4
	転送元領域の先頭アドレス	R2	4
出　力	—	—	—

内部レジスタ変化およびフラグ変化

(実行前) → (実行後)

R0	転送バイト数 → 変化
R1	転送先領域の先頭ｱﾄﾞﾚｽ → 変化
R2	転送元領域の先頭ｱﾄﾞﾚｽ → 変化
R3	ワーク
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

ー：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
142
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
4
ｽﾃｰﾄ数
429
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

プログラミング仕様のステート数は、転送バイト数が100バイトときの値です。

ブロック転送 (4バイト整合されていない)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE	使用機能	MOV. B 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

説　明

（1） 機能説明

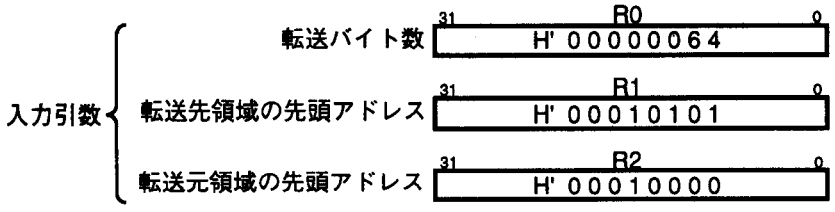
(a) 引数の詳細は以下の通りです。

R0：入力引数として、転送バイト数(任意バイト)をセットします。但し、ハードウェア上の制約があるので注意してください。

R1：入力引数として、転送先領域の先頭アドレス(任意アドレス)をセットします。

R2：入力引数として、転送元領域の先頭アドレス(任意アドレス)をセットします。

(b) 図3.6にソフトウェアMOVEの実行例を示します。



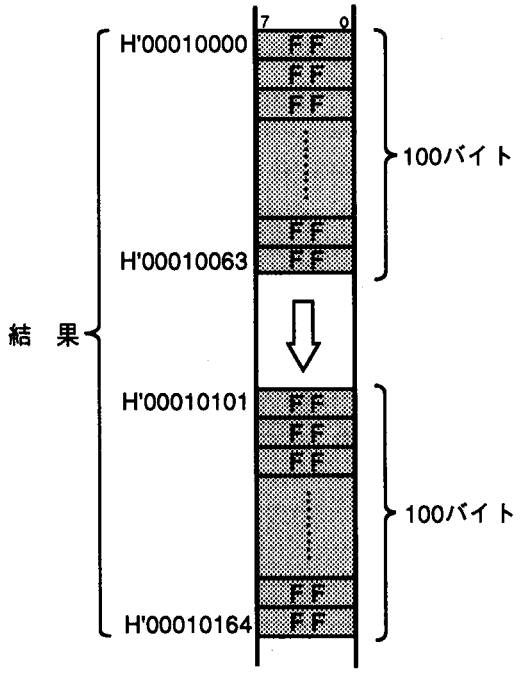


図3.6　ソフトウェアMOVEの実行例

ブロック転送 (4バイト整合されていない)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE	使用機能	MOV．B 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

説　明

（2） 使用上の注意

(a) 転送元領域と転送先領域が重ならないように入力引数を設定してください。転送元領域と転送先領域が重なった場合、図3.7に示すように、重なった部分の転送元領域のデータは破壊されます。

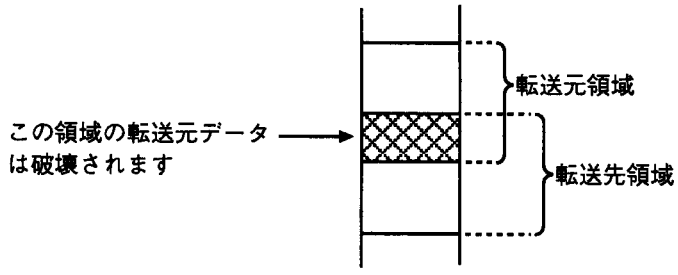


図3.7　データが重なる場合のブロック転送

(b) 転送バイト数、転送先領域の先頭アドレスおよび転送元領域の先頭アドレスがセットされていたR0、R1およびR2は、ソフトウェアMOVEの実行により内容が変化します。ソフトウェアMOVE実行後も、転送バイト数、転送先領域の先頭アドレスおよび転送元領域の先頭アドレスを必要とする場合、転送バイト数、転送先領域の先頭アドレスおよび転送元領域の先頭アドレスをあらかじめ退避してください。

（3） 使用RAM説明

ソフトウェアMOVEではRAMは使用していません。

（4） 使用例

転送元領域の先頭アドレス、転送先領域の先頭アドレスおよび転送バイト数を入力引数にセットしてからソフトウェアMOVEをサブルーチンコールします。

```
        MOV.L   DATA1,R0        ……転送バイト数を入力引数(R0)にセット
        MOV.L   DATA2,R1        ……転送先領域の先頭アドレスを入力引数(R1)に
                                    セット
        BSR     MOVE            ……ソフトウェアMOVEをサブルーチンコール
        MOV.L   DATA3,R2        ……転送元領域の先頭アドレスを入力引数(R2)に
                                    セット
                :
                .align  4
DATA1   .data.l H'00000064
DATA2   .data.l H'00010101
DATA3   .data.l H'00010000
```

ブロック転送 (4バイト整合されていない)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE	使用機能	MOV. B 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

説　明

(5) 動作原理

(a) 転送元および転送先のアドレスが共に任意アドレス(4バイト整合されていない)のため1バイト単位で転送元から転送先へ転送します。

(b) 転送元アドレスの指定には、ポストインクリメントレジスタ間接(@R2+)を使用します。転送元アドレスは+1ずつ自動的にインクリメントされます。

転送先アドレスの指定には、ディスプレースメント付きレジスタ間接を使用します。ディスプレースメントは、0～15なので、15バイト転送毎に転送先アドレスを+16インクリメントする必要がありますが、それ以外はインクリメントの処理が必要ありません。

(c) 転送元先頭アドレス(R2)+転送バイト数(R0)をR3に設定します。設定後、転送バイト数が設定されていたR0は、データ転送用ワークとして使用します。転送元のデータをR0へ転送後、R2≦R3の判定を行います。成立(R2≦R3)のとき、R0上のデータは転送元域内のデータなので、転送先へ転送します。不成立(R2＞R3)のとき、R0上のデータは転送元領域外のデータなので、転送を中止します。

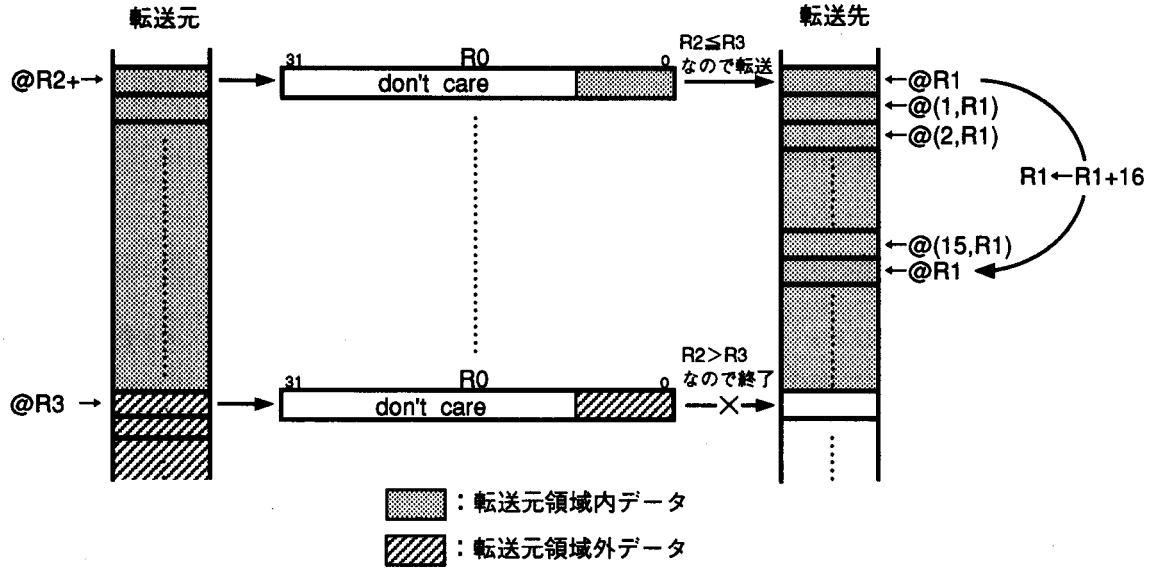
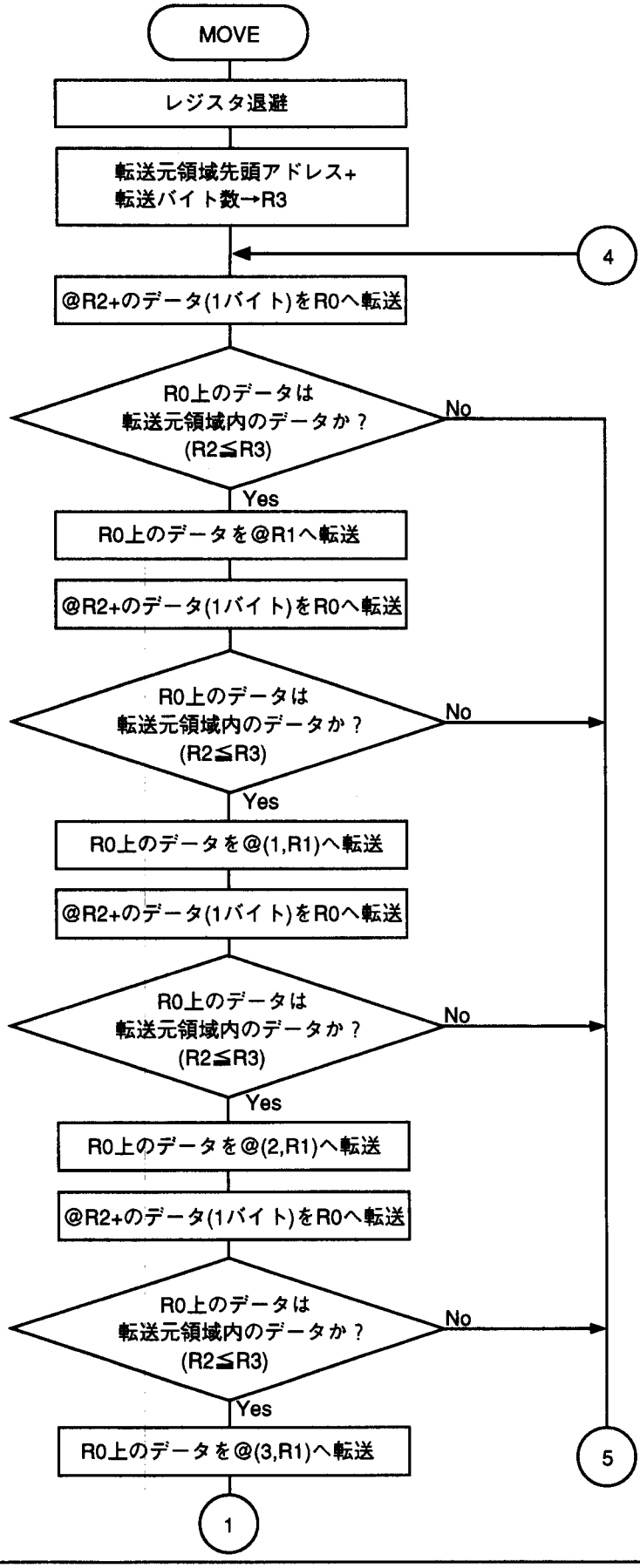


図3.8　データ転送方法

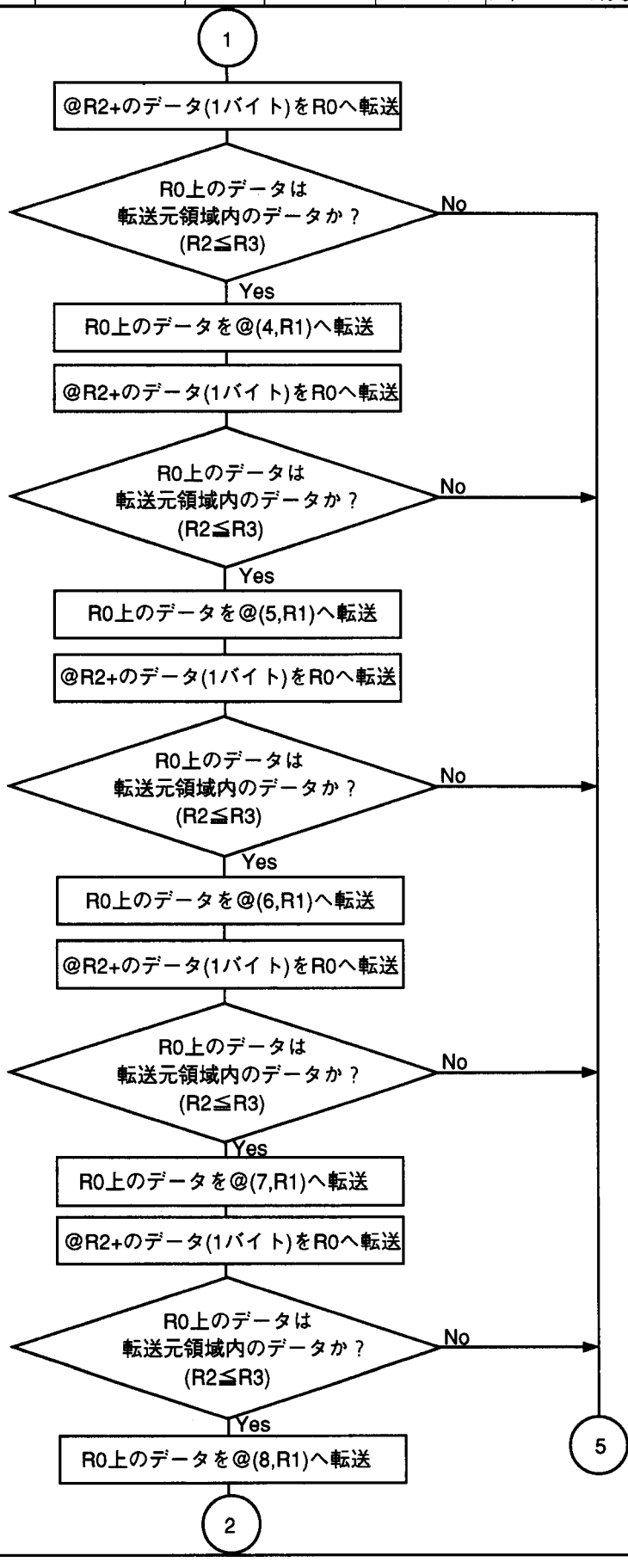
ブロック転送 (4バイト整合されていない)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE	使用機能	MOV．B 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

フローチャート



ブロック転送 (4バイト整合されていない)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE	使用機能	MOV. B 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

フローチャート



3．2．4　32ビットデータの多ビットシフト（右論理シフト）

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLRN	使用機能	SHLR2 命令 SHLR8 命令 SHLR16 命令

機　能

32ビットデータを多ビット(0～31)右論理シフトします。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	シフト数(0～31)	R0	4
	シフト前32ビットデータ	R1	4
出　力	シフト後32ビットデータ	R1	4

内部レジスタ変化およびフラグ変化

	実行前 → 実行後
R0	シフト数 → 変化なし
R1	シフト前32ビットデータ → シフト後32ビットデータ
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

－：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

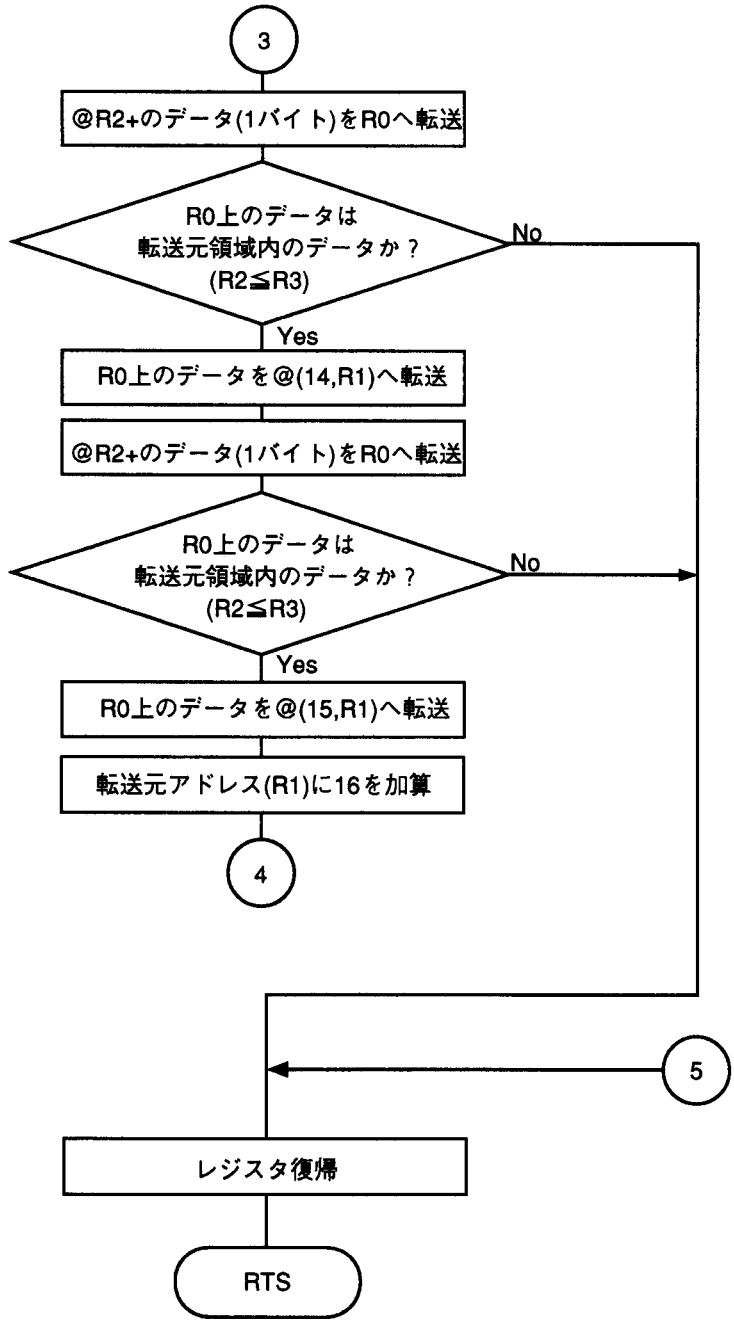
ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
36
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
0
ｽﾃｰﾄ数
19
ﾘｴﾝﾄﾗﾝﾄ
可
ﾘﾛｹｰｼｮﾝ
可
途中割込み
可

注意事項

プログラミング仕様のステート数は、31ビットシフトさせたときの値です。

ブロック転送 (4バイト整合されていない)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE	使用機能	MOV．B 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

フローチャート



ブロック転送 (4バイト整合されていない)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE	使用機能	MOV. B命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

プログラムリスト

```
                                                1  ;*******************************************************************************
                                                2  ;*                                                                             *
                                                3  ;*          NAME : MOVING MEMORY BLOCKS (MOVE)                                 *
                                                4  ;*                                                                             *
                                                5  ;*******************************************************************************
                                                6  ;*                                                                             *
                                                7  ;*         ENTRY : R0    (NUMBER OF TRANSFER)                                 *
                                                8  ;*                 R1    (DESTINATION ADDRESS)                                *
                                                9  ;*                 R2    (SOURCE ADDRESS)                                     *
                                               10  ;*       RETURNS : NOTHING                                                     *
                                               11  ;*                                                                             *
                                               12  ;*******************************************************************************
00001000                                       13          .SECTION  A,CODE,LOCATE=H'1000
00001000                00001000               14  MOVE    .EQU    $                   ; Entry point
00001000 2F36                                  15          MOV.L   R3,@-R15            ; Escape register
00001002 6323                                  16          MOV     R2,R3               ;
00001004 330C                                  17          ADD     R0,R3               ;
00001006                                       18  MOVE1                               ;
00001006 6024                                  19          MOV.B   @R2+,R0             ; Load source data
00001008 3322                                  20          CMP/HS  R2,R3               ; R2 <= R3 ?
0000100A 8B3E                                  21          BF      MOVE_END            ; No
0000100C 2100                                  22          MOV.B   R0,@R1              ; Yes -> Store source data
0000100E                                       23  MOVE2                               ;
0000100E 6024                                  24          MOV.B   @R2+,R0             ; Load source data
00001010 3322                                  25          CMP/HS  R2,R3               ; R2 <= R3 ?
00001012 8B3A                                  26          BF      MOVE_END            ; No
00001014 8011                                  27          MOV.B   R0,@(1,R1)          ; Yes -> Store source data
00001016                                       28  MOVE3                               ;
00001016 6024                                  29          MOV.B   @R2+,R0             ; Load source data
00001018 3322                                  30          CMP/HS  R2,R3               ; R2 <= R3 ?
0000101A 8B36                                  31          BF      MOVE_END            ; No
0000101C 8012                                  32          MOV.B   R0,@(2,R1)          ; Yes -> Store source data
0000101E                                       33  MOVE4                               ;
0000101E 6024                                  34          MOV.B   @R2+,R0             ; Load source data
00001020 3322                                  35          CMP/HS  R2,R3               ; R2 <= R3 ?
00001022 8B32                                  36          BF      MOVE_END            ; No
00001024 8013                                  37          MOV.B   R0,@(3,R1)          ; Yes -> Store source data
00001026                                       38  MOVE5                               ;
00001026 6024                                  39          MOV.B   @R2+,R0             ; Load source data
00001028 3322                                  40          CMP/HS  R2,R3               ; R2 <= R3 ?
0000102A 8B2E                                  41          BF      MOVE_END            ; No
0000102C 8014                                  42          MOV.B   R0,@(4,R1)          ; Yes -> Store source data
0000102E                                       43  MOVE6                               ;
0000102E 6024                                  44          MOV.B   @R2+,R0             ; Load source data
00001030 3322                                  45          CMP/HS  R2,R3               ; R2 <= R3 ?
00001032 8B2A                                  46          BF      MOVE_END            ; No
00001034 8015                                  47          MOV.B   R0,@(5,R1)          ; Yes -> Store source data
00001036                                       48  MOVE7                               ;
00001036 6024                                  49          MOV.B   @R2+,R0             ; Load source data
00001038 3322                                  50          CMP/HS  R2,R3               ; R2 <= R3 ?
0000103A 8B26                                  51          BF      MOVE_END            ; No
0000103C 8016                                  52          MOV.B   R0,@(6,R1)          ; Yes -> Store source data
0000103E                                       53  MOVE8                               ;
0000103E 6024                                  54          MOV.B   @R2+,R0             ; Load source data
00001040 3322                                  55          CMP/HS  R2,R3               ; R2 <= R3 ?
00001042 8B22                                  56          BF      MOVE_END            ; No
00001044 8017                                  57          MOV.B   R0,@(7,R1)          ; Yes -> Store source data
```

ブロック転送 (4バイト整合されていない)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE	使用機能	MOV. B命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

プログラムリスト

```
  58 00001046                           58    MOVE9
  59 00001046 6024                      59          MOV.B   @R2+,R0         ; Load source data
  60 00001048 3322                      60          CMP/HS  R2,R3           ; R2 <= R3 ?
  61 0000104A 8B1E                      61          BF      MOVE_END        ; No
  62 0000104C 8018                      62          MOV.B   R0,@(8,R1)      ; Yes -> Store source data
  63 0000104E                           63    MOVE10                        ;
  64 0000104E 6024                      64          MOV.B   @R2+,R0         ; Load source data
  65 00001050 3322                      65          CMP/HS  R2,R3           ; R2 <= R3 ?
  66 00001052 8B1A                      66          BF      MOVE_END        ; No
  67 00001054 8019                      67          MOV.B   R0,@(9,R1)      ; Yes -> Store source data
  68 00001056                           68    MOVE11                        ;
  69 00001056 6024                      69          MOV.B   @R2+,R0         ; Load source data
  70 00001058 3322                      70          CMP/HS  R2,R3           ; R2 <= R3 ?
  71 0000105A 8B16                      71          BF      MOVE_END        ; No
  72 0000105C 801A                      72          MOV.B   R0,@(10,R1)     ; Yes -> Store source data
  73 0000105E                           73    MOVE12                        ;
  74 0000105E 6024                      74          MOV.B   @R2+,R0         ; Load source data
  75 00001060 3322                      75          CMP/HS  R2,R3           ; R2 <= R3 ?
  76 00001062 8B12                      76          BF      MOVE_END        ; No
  77 00001064 801B                      77          MOV.B   R0,@(11,R1)     ; Yes -> Store source data
  78 00001066                           78    MOVE13                        ;
  79 00001066 6024                      79          MOV.B   @R2+,R0         ; Load source data
  80 00001068 3322                      80          CMP/HS  R2,R3           ; R2 <= R3 ?
  81 0000106A 8B0E                      81          BF      MOVE_END        ; No
  82 0000106C 801C                      82          MOV.B   R0,@(12,R1)     ; Yes -> Store source data
  83 0000106E                           83    MOVE14                        ;
  84 0000106E 6024                      84          MOV.B   @R2+,R0         ; Load source data
  85 00001070 3322                      85          CMP/HS  R2,R3           ; R2 <= R3 ?
  86 00001072 8B0A                      86          BF      MOVE_END        ; No
  87 00001074 801D                      87          MOV.B   R0,@(13,R1)     ; Yes -> Store source data
  88 00001076                           88    MOVE15                        ;
  89 00001076 6024                      89          MOV.B   @R2+,R0         ; Load source data
  90 00001078 3322                      90          CMP/HS  R2,R3           ; R2 <= R3 ?
  91 0000107A 8B06                      91          BF      MOVE_END        ; No
  92 0000107C 801E                      92          MOV.B   R0,@(14,R1)     ; Yes -> Store source data
  93 0000107E                           93    MOVE16                        ;
  94 0000107E 6024                      94          MOV.B   @R2+,R0         ; Load source data
  95 00001080 3322                      95          CMP/HS  R2,R3           ; R2 <= R3 ?
  96 00001082 8B02                      96          BF      MOVE_END        ; No
  97 00001084 801F                      97          MOV.B   R0,@(15,R1)     ; Yes -> Store source data
  98                                    98                                  ;
  99 00001086 AFBE                      99          BRA     MOVE1           ;
 100 00001088 7110                     100          ADD     #D'16,R1        ; R1 <- R1 + 16
 101 0000108A                          101    MOVE_END                      ;
 102 0000108A 000B                     102          RTS                     ;
 103 0000108C 63F6                     103          MOV.L   @R15+,R3        ; Return register
 104                                   104          .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．2　ブロック転送（4バイト整合されている）

ブロック転送 (4バイト整合されている)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE4	使用機能	MOV．L 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

機　能

ブロックデータの転送を行ないます。但し、ブロックデータの転送元領域および転送先領域の先頭アドレスは4nアドレス、ブロックデータは4nバイトの場合に限ります。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	転送バイト数(4nバイト)	R0	4
	転送先領域の先頭アドレス(4nアドレス)	R1	4
	転送元領域の先頭アドレス(4nアドレス)	R2	4
出　力	—	—	—

内部レジスタ変化およびフラグ変化

(実行前) → (実行後)

R0	転送バイト数 → 変化
R1	転送先領域の先頭ｱﾄﾞﾚｽ → 変化
R2	転送元領域の先頭ｱﾄﾞﾚｽ → 変化
R3	ワーク
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

－：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
142
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
4
ｽﾃｰﾄ数
114
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

プログラミング仕様のステート数は、転送バイト数が100バイトのときの値です。

ブロック転送 (4バイト整合されている)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE4	使用機能	MOV．L 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

説　明

(1) 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数として、転送バイト数(4nバイト)をセットします。但し、ハードウェア上の制約があるので注意してください。

R1：入力引数として、転送先領域の先頭アドレス(4nアドレス)をセットします。

R2：入力引数として、転送元領域の先頭アドレス(4nアドレス)をセットします。

(b) 図3.9にソフトウェアMOVE4の実行例を示します。

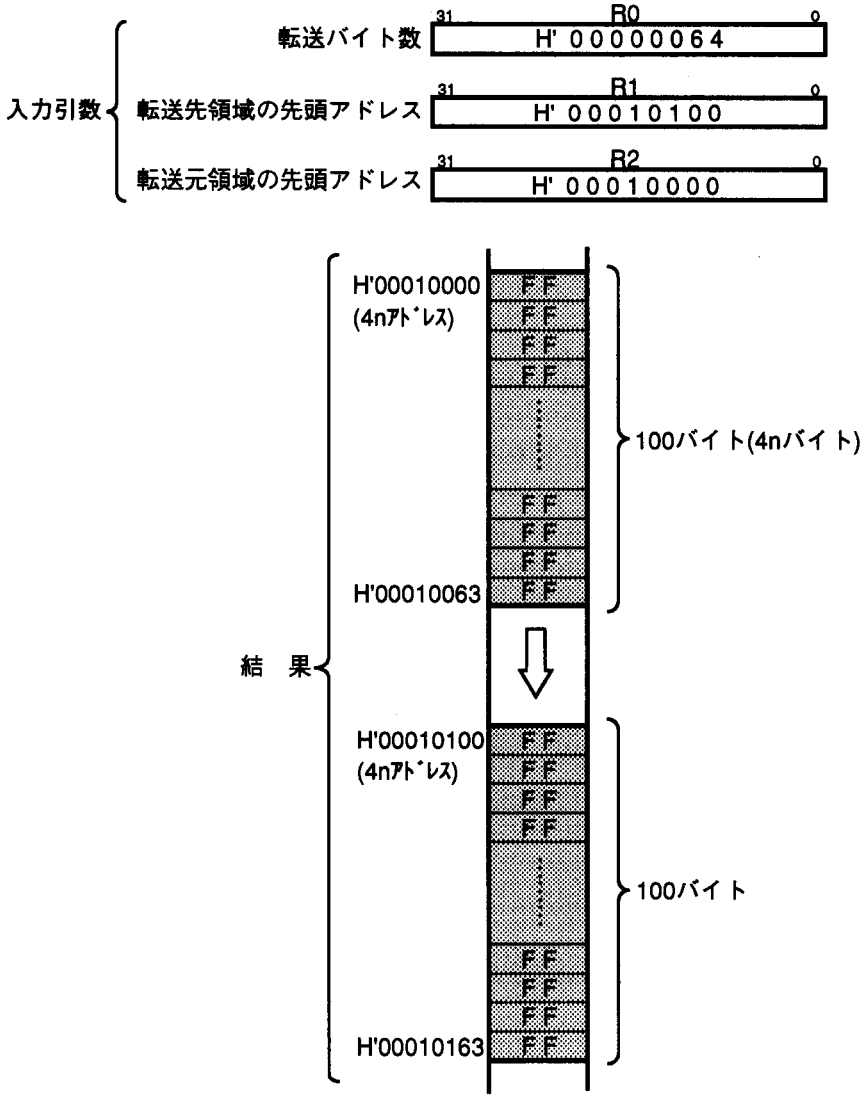


図3.9　ソフトウェアMOVE4の実行例

ブロック転送 (4バイト整合されている)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE4	使用機能	MOV. L 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

説　明

(2) 使用上の注意

(a) 転送元領域と転送先領域が重ならないように入力引数を設定してください。転送元領域と転送先領域が重なった場合、図3.10に示すように重なった部分の転送元領域のデータは破壊されます。

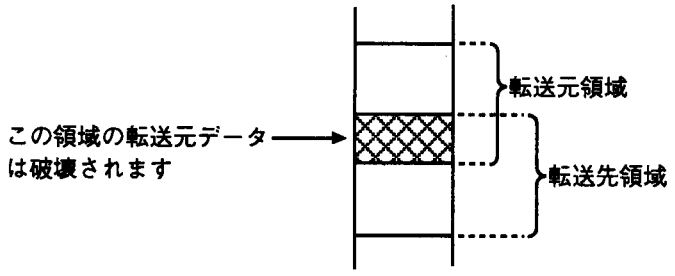


図3.10　データが重なる場合のブロック転送

(b) 転送バイト数、転送先領域の先頭アドレスおよび転送元領域の先頭アドレスがセットされていたR0、R1およびR2は、ソフトウェアMOVEの実行により内容が変化します。ソフトウェアMOVE実行後も、転送バイト数、転送先領域の先頭アドレスおよび転送元領域の先頭アドレスを必要とする場合、転送バイト数、転送先領域の先頭アドレスおよび転送元領域の先頭アドレスをあらかじめ退避してください。

(3) 使用RAM説明

ソフトウェアMOVE4ではRAMは使用していません。

(4) 使用例

転送元領域の先頭アドレス、転送先領域の先頭アドレスおよび転送バイト数を入力引数にセットしてからソフトウェアMOVE4をサブルーチンコールします。

```
        MOV. L   DATA1,R0      ……転送バイト数を入力引数(R0)にセット
        MOV. L   DATA2,R1      ……転送先領域の先頭アドレスを入力引数(R1)に
                                   セット
        BSR      MOVE4         ……ソフトウェアMOVE4をサブルーチンコール
        MOV. L   DATA3,R2      ……転送元領域の先頭アドレスを入力引数(R2)に
                                   セット
                 :
                 :
        .align   4
DATA1   .data.l  H'00000064
DATA2   .data.l  H'00010100
DATA3   .data.l  H'00010000
```

ブロック転送 (4バイト整合されている)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE4	使用機能	MOV. L 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

説　明

(5) 動作原理

(a) 転送元および転送先のアドレスが共に4nアドレス(4バイト整合されてる)のため4バイト単位で転送元から転送先へ転送します。

(b) 転送元アドレスの指定には、ポストインクリメントレジスタ間接(@R2+)を使用するので転送元アドレスは+4ずつ自動的にインクリメントされます。転送先アドレスの指定には、ディスプレースメント付きレジスタ間接を使用します。ディスプレースメントは、0～60なので、60バイト転送毎に転送先アドレスを+64インクリメントする必要がありますが、それ以外はインクリメントの処理が必要ありません。

(c) 転送元先頭アドレス(R2)+転送バイト数(R0)をR3に設定します。設定後、転送バイト数が設定されていたR0は、データ転送用ワークとして使用します。転送元のデータをR0へ転送後、R2≦R3の判定を行います。成立(R2≦R3)のとき、R0上のデータは転送元域内のデータなので、転送先へ転送します。不成立(R2＞R3)のとき、R0上のデータは転送元域外のデータなので、転送を中止します。

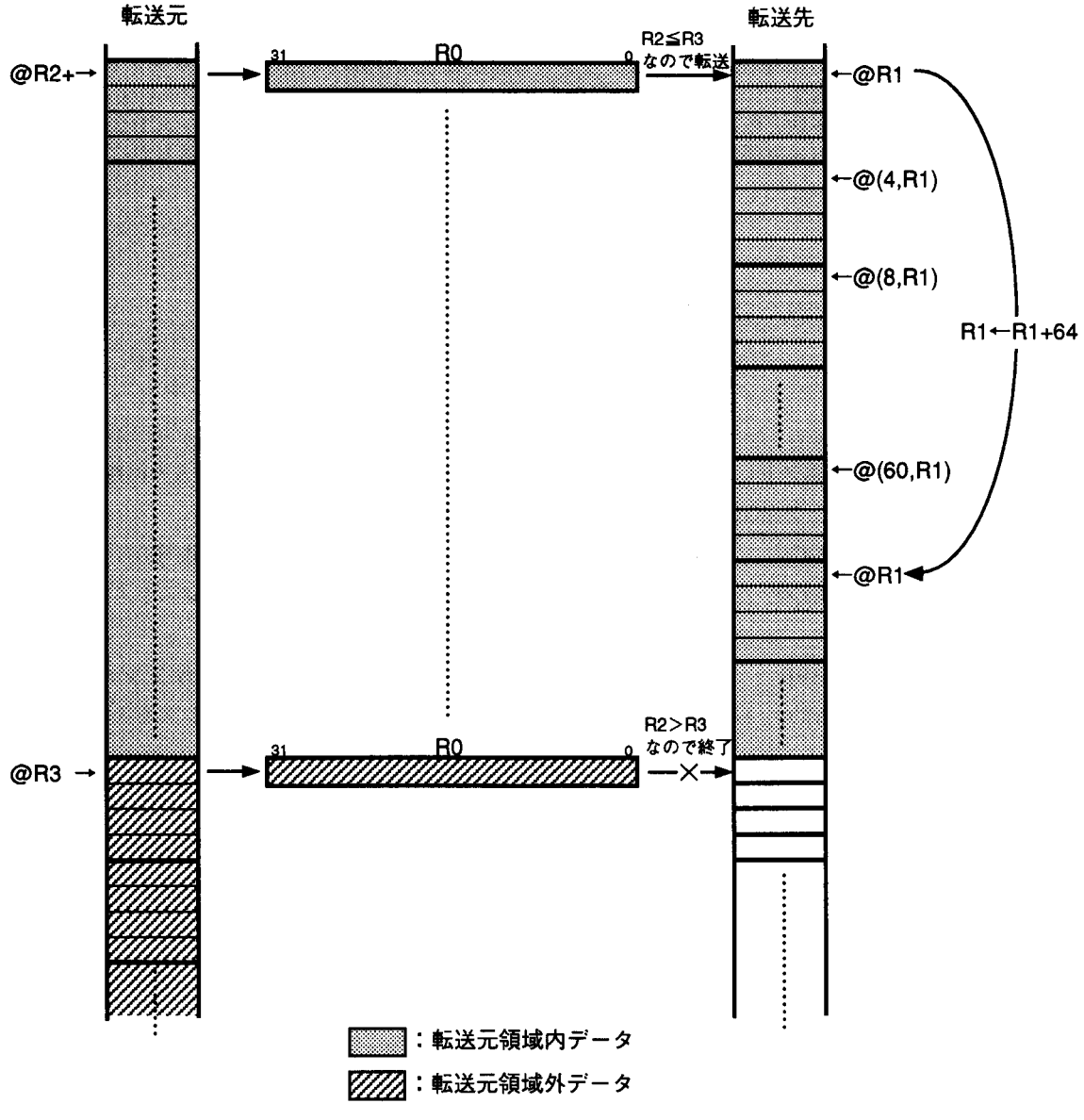
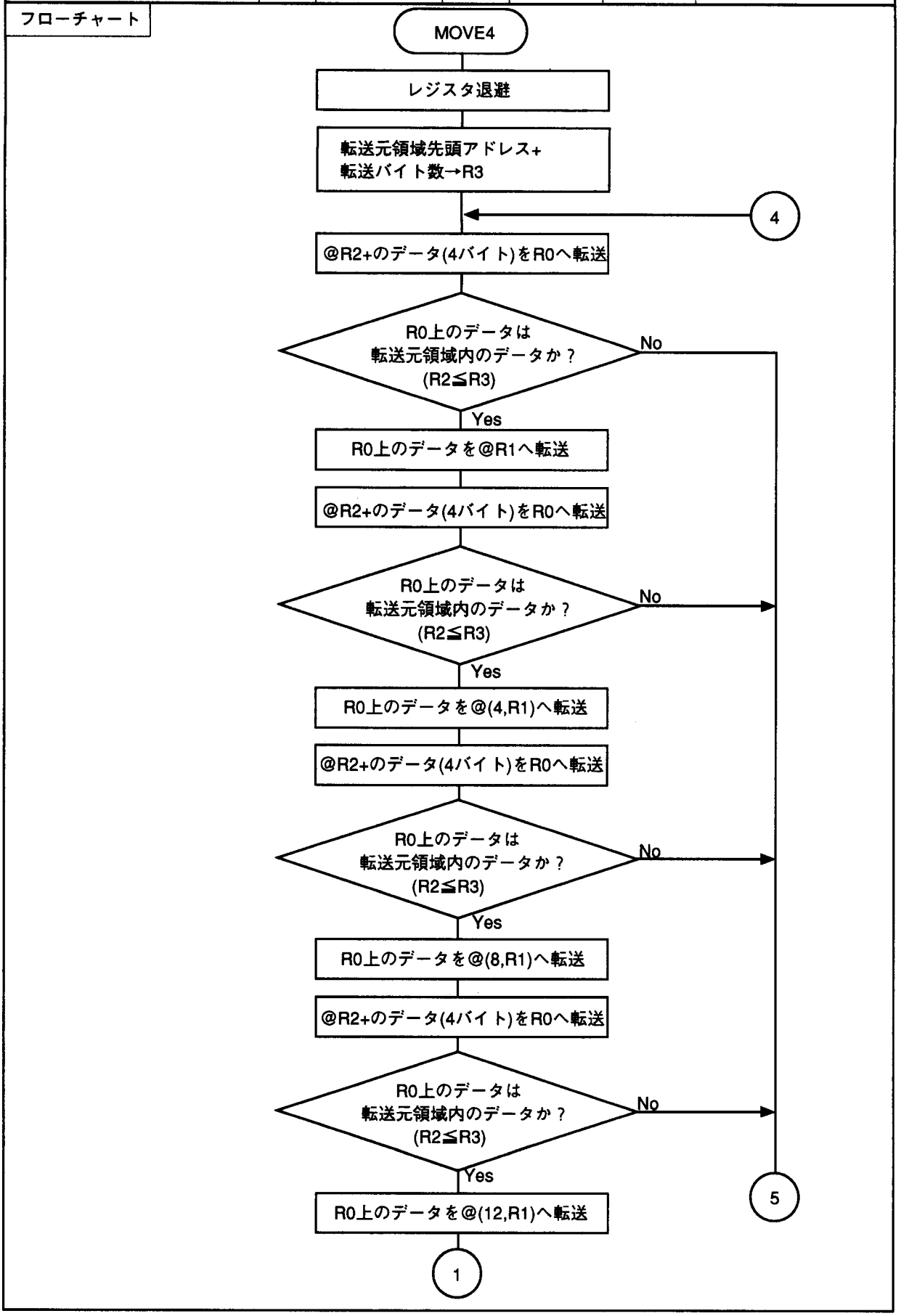


図3.11　データ転送方法

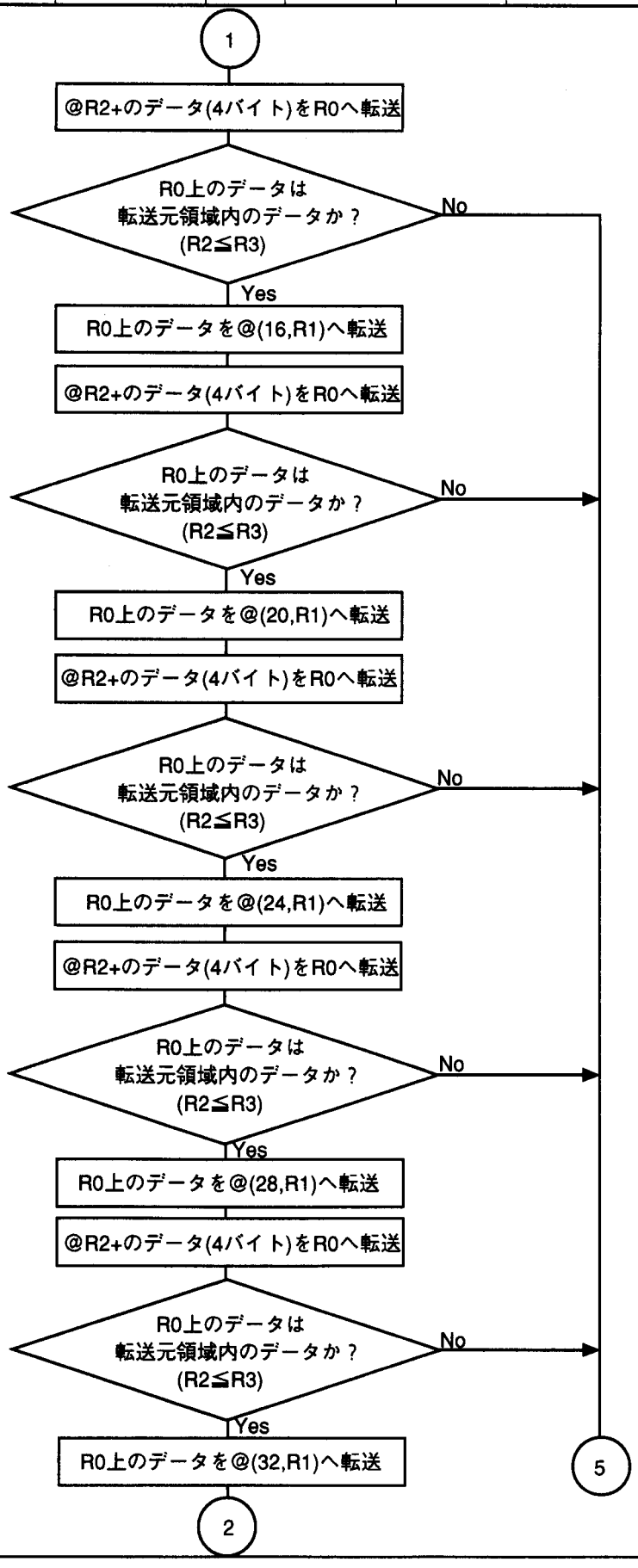
ブロック転送 (4バイト整合されている)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE4	使用機能	MOV．L 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

フローチャート



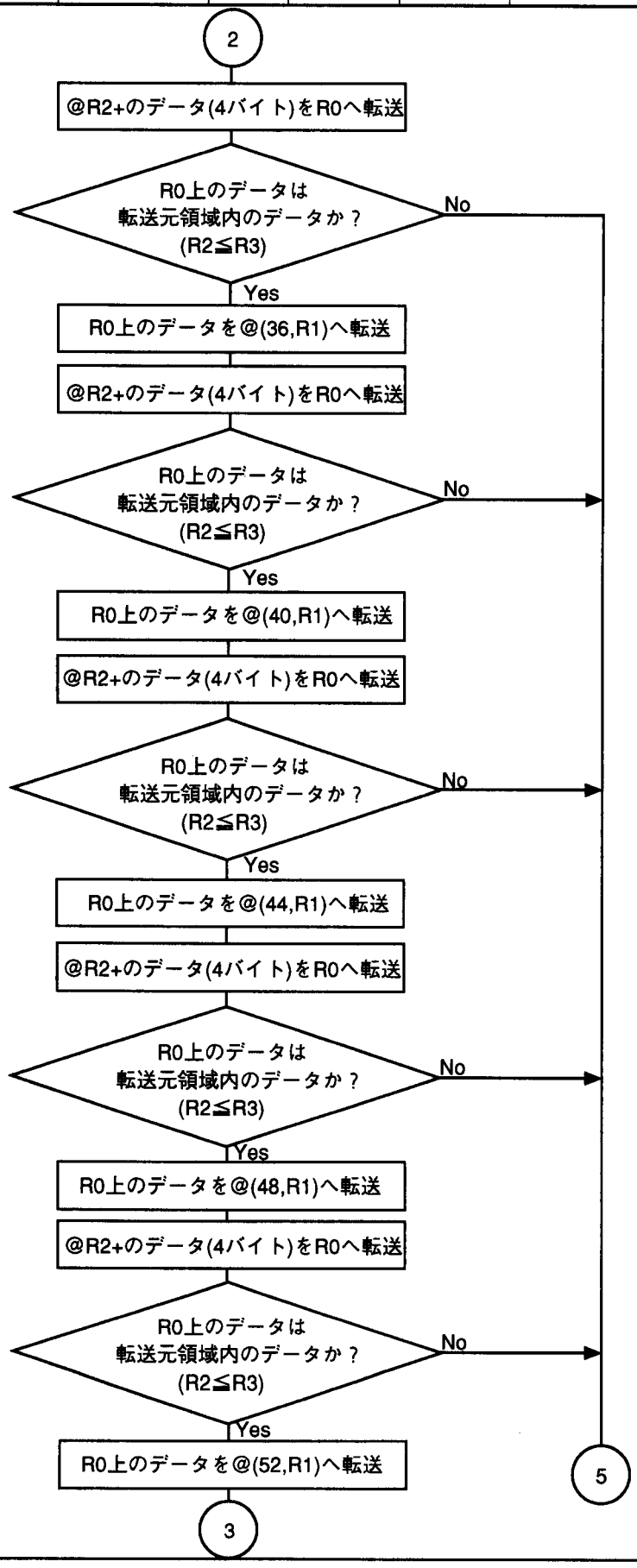
ブロック転送 (4バイト整合されている)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE4	使用機能	MOV．L 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

フローチャート



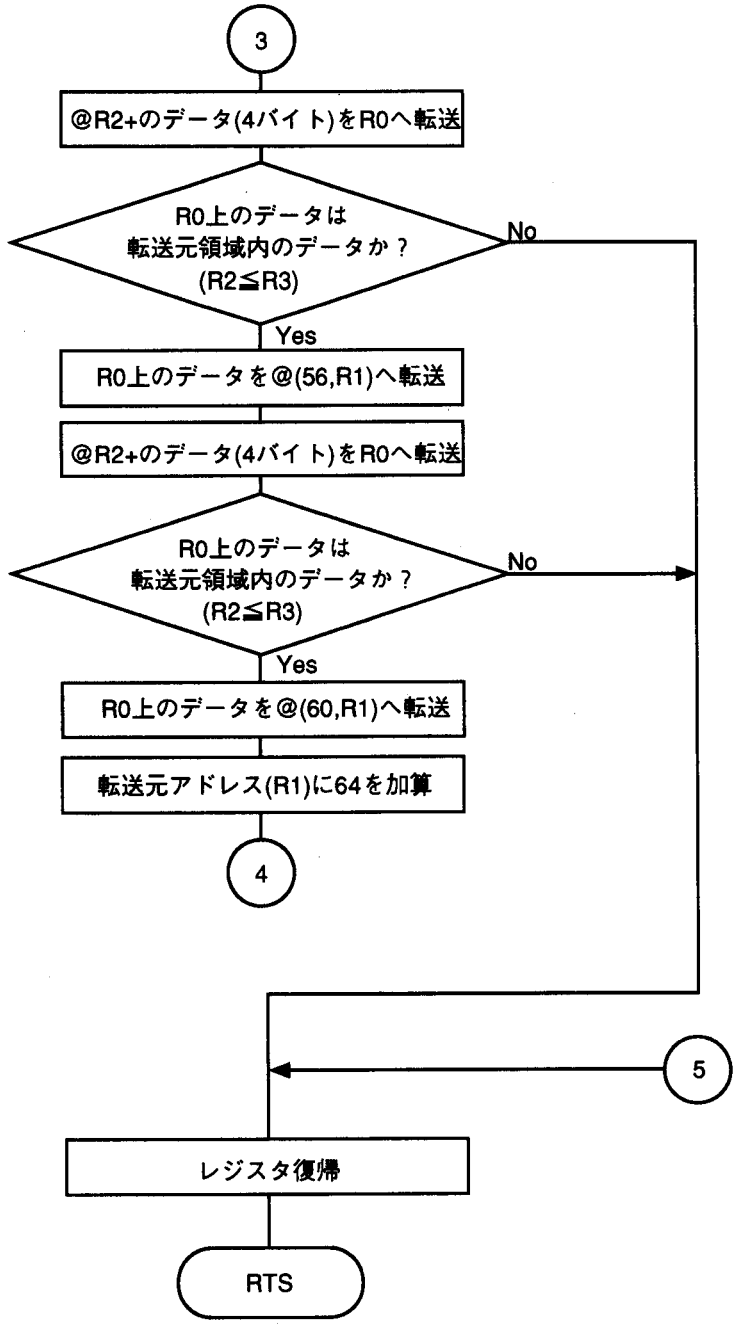
ブロック転送 (4バイト整合されている)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE4	使用機能	MOV．L 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

フローチャート



ブロック転送 (4バイト整合されている)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE4	使用機能	MOV. L 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

フローチャート



ブロック転送 (4バイト整合されている)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE4	使用機能	MOV．B命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

プログラムリスト

```
                                    1  ;*********************************************************************
                                    2  ;*                                                                   *
                                    3  ;*          NAME : MOVING MEMORY BLOCKS (MOVE4)                      *
                                    4  ;*                                                                   *
                                    5  ;*********************************************************************
                                    6  ;*                                                                   *
                                    7  ;*         ENTRY : R0    (NUMBER OF TRANSFER)                       *
                                    8  ;*                 R1    (DESTINATION ADDRESS)                      *
                                    9  ;*                 R2    (SOURCE ADDRESS)                           *
                                   10  ;*       RETURNS : NOTHING                                           *
                                   11  ;*                                                                   *
                                   12  ;*********************************************************************
00001000                           13          .SECTION  A,CODE,LOCATE=H'1000
          00001000                 14  MOVE4   .EQU     $                  ; Entry point
00001000 2F36                      15          MOV.L   R3,@-R15           ; Escape registr
00001002 6323                      16          MOV     R2,R3              ;
00001004 330C                      17          ADD     R0,R3              ;
00001006                           18  MOVE41                             ;
00001006 6026                      19          MOV.L   @R2+,R0            ; Load source data
00001008 3322                      20          CMP/HS  R2,R3              ; R2 <= R3 ?
0000100A 8B3E                      21          BF      MOVE4_END          ; No
0000100C 2102                      22          MOV.L   R0,@R1             ; Yes -> Store source data
0000100E                           23  MOVE42                             ;
0000100E 6026                      24          MOV.L   @R2+,R0            ; Load source data
00001010 3322                      25          CMP/HS  R2,R3              ; R2 <= R3 ?
00001012 8B3A                      26          BF      MOVE4_END          ; No
00001014 1101                      27          MOV.L   R0,@(4,R1)         ; Yes -> Store source data
00001016                           28  MOVE43                             ;
00001016 6026                      29          MOV.L   @R2+,R0            ; Load source data
00001018 3322                      30          CMP/HS  R2,R3              ; R2 <= R3 ?
0000101A 8B36                      31          BF      MOVE4_END          ; No
0000101C 1102                      32          MOV.L   R0,@(8,R1)         ; Yes -> Store source data
0000101E                           33  MOVE44                             ;
0000101E 6026                      34          MOV.L   @R2+,R0            ; Load source data
00001020 3322                      35          CMP/HS  R2,R3              ; R2 <= R3 ?
00001022 8B32                      36          BF      MOVE4_END          ; No
00001024 1103                      37          MOV.L   R0,@(12,R1)        ; Yes -> Store source data
00001026                           38  MOVE45                             ;
00001026 6026                      39          MOV.L   @R2+,R0            ; Load source data
00001028 3322                      40          CMP/HS  R2,R3              ; R2 <= R3 ?
0000102A 8B2E                      41          BF      MOVE4_END          ; No
0000102C 1104                      42          MOV.L   R0,@(16,R1)        ; Yes -> Store source data
0000102E                           43  MOVE46                             ;
0000102E 6026                      44          MOV.L   @R2+,R0            ; Load source data
00001030 3322                      45          CMP/HS  R2,R3              ; R2 <= R3 ?
00001032 8B2A                      46          BF      MOVE4_END          ; No
00001034 1105                      47          MOV.L   R0,@(20,R1)        ; Yes -> Store source data
00001036                           48  MOVE47                             ;
00001036 6026                      49          MOV.L   @R2+,R0            ; Load source data
00001038 3322                      50          CMP/HS  R2,R3              ; R2 <= R3 ?
0000103A 8B26                      51          BF      MOVE4_END          ; No
0000103C 1106                      52          MOV.L   R0,@(24,R1)        ; Yes -> Store source data
0000103E                           53  MOVE48                             ;
0000103E 6026                      54          MOV.L   @R2+,R0            ; Load source data
00001040 3322                      55          CMP/HS  R2,R3              ; R2 <= R3 ?
00001042 8B22                      56          BF      MOVE4_END          ; No
00001044 1107                      57          MOV.L   R0,@(28,R1)        ; Yes -> Store source data
```

ブロック転送 (4バイト整合されている)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE4	使用機能	MOV. B命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

プログラムリスト

```
  58 00001046                    58    MOVE49
  59 00001046 6026               59          MOV.L   @R2+,R0         ; Load source data
  60 00001048 3322               60          CMP/HS  R2,R3           ; R2 <= R3 ?
  61 0000104A 8B1E               61          BF      MOVE4_END       ; No
  62 0000104C 1108               62          MOV.L   R0,@(32,R1)     ; Yes -> Store source data
  63 0000104E                    63    MOVE410                       ;
  64 0000104E 6026               64          MOV.L   @R2+,R0         ; Load source data
  65 00001050 3322               65          CMP/HS  R2,R3           ; R2 <= R3 ?
  66 00001052 8B1A               66          BF      MOVE4_END       ; No
  67 00001054 1109               67          MOV.L   R0,@(36,R1)     ; Yes -> Store source data
  68 00001056                    68    MOVE411                       ;
  69 00001056 6026               69          MOV.L   @R2+,R0         ; Load source data
  70 00001058 3322               70          CMP/HS  R2,R3           ; R2 <= R3 ?
  71 0000105A 8B16               71          BF      MOVE4_END       ; No
  72 0000105C 110A               72          MOV.L   R0,@(40,R1)     ; Yes -> Store source data
  73 0000105E                    73    MOVE412                       ;
  74 0000105E 6026               74          MOV.L   @R2+,R0         ; Load source data
  75 00001060 3322               75          CMP/HS  R2,R3           ; R2 <= R3 ?
  76 00001062 8B12               76          BF      MOVE4_END       ; No
  77 00001064 110B               77          MOV.L   R0,@(44,R1)     ; Yes -> Store source data
  78 00001066                    78    MOVE413                       ;
  79 00001066 6026               79          MOV.L   @R2+,R0         ; Load source data
  80 00001068 3322               80          CMP/HS  R2,R3           ; R2 <= R3 ?
  81 0000106A 8B0E               81          BF      MOVE4_END       ; No
  82 0000106C 110C               82          MOV.L   R0,@(48,R1)     ; Yes -> Store source data
  83 0000106E                    83    MOVE414                       ;
  84 0000106E 6026               84          MOV.L   @R2+,R0         ; Load source data
  85 00001070 3322               85          CMP/HS  R2,R3           ; R2 <= R3 ?
  86 00001072 8B0A               86          BF      MOVE4_END       ; No
  87 00001074 110D               87          MOV.L   R0,@(52,R1)     ; Yes -> Store source data
  88 00001076                    88    MOVE415                       ;
  89 00001076 6026               89          MOV.L   @R2+,R0         ; Load source data
  90 00001078 3322               90          CMP/HS  R2,R3           ; R2 <= R3 ?
  91 0000107A 8B06               91          BF      MOVE4_END       ; No
  92 0000107C 110E               92          MOV.L   R0,@(56,R1)     ; Yes -> Store source data
  93 0000107E                    93    MOVE416                       ;
  94 0000107E 6026               94          MOV.L   @R2+,R0         ; Load source data
  95 00001080 3322               95          CMP/HS  R2,R3           ; R2 <= R3 ?
  96 00001082 8B02               96          BF      MOVE4_END       ; No
  97 00001084 110F               97          MOV.L   R0,@(60,R1)     ; Yes -> Store source data
  98                             98                                  ;
  99 00001086 AFBE               99          BRA     MOVE41          ;
 100 00001088 7140              100          ADD     #D'64,R1        ; R1 <- R1 + 64
 101 0000108A                   101    MOVE4_END                     ;
 102 0000108A 000B              102          RTS                     ;
 103 0000108C 63F6              103          MOV.L   @R15+,R3        ; Return register
 104                            104          .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．3　32ビットデータの多ビットシフト（右算術シフト）

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右算術ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHARN	使用機能	SHLR 2 命令 SHLR 8 命令 SHLR 16 命令

機　能

32ビットデータを多ビット(0～31)右算術シフトします。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	シフト数(0～31)	R0	4
	シフト前32ビットデータ	R1	4
出　力	シフト後32ビットデータ	R1	4

内部レジスタ変化およびフラグ変化

実行前　→　実行後

R0	シフト数　→　変化
R1	シフト前32ビットデータ　→　シフト後32ビットデータ
R2	ワーク
R3	ワーク
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

－：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
74
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
8
ｽﾃｰﾄ数
38
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

プログラミング仕様のステート数は、31ビットシフトさせたときの値です。

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右算術ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHARN	使用機能	SHLR 2 命令 SHLR 8 命令 SHLR 16 命令

説　明

(1) 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数としてシフト数(0～31)をセットします。

R1：入力引数として、シフト前32ビットデータをセットします。

出力引数として、シフト後32ビットデータがセットされます。

(b) 図3.12にソフトウェアSHARNの実行例を示します。

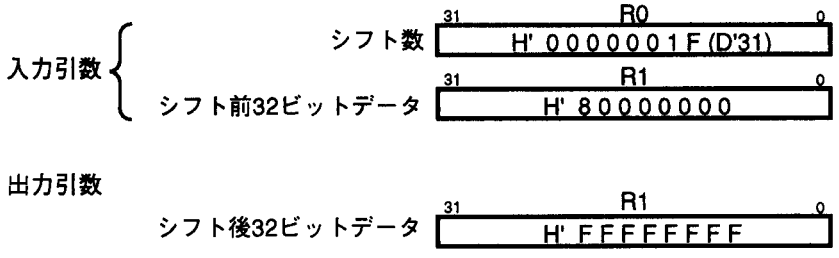


図3.12　ソフトウェアSHARNの実行例

(2) 使用上の注意

シフト前32ビットデータがセットされていたR1にはシフト後32ビットデータがセットされるため、シフト前32ビットデータは破壊されます。また、シフト数がセットされていたR0は、ソフトウェアSHARNの実行により内容が変化します。

ソフトウェアSHARN実行後もシフト前32ビットデータおよびシフト数を必要とする場合、シフト前32ビットデータおよびシフト数をあらかじめ退避してください。

(3) 使用RAM説明

ソフトウェアSHARNではRAMは使用していません。

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右算術ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHARN	使用機能	SHLR 2 命令 SHLR 8 命令 SHLR 16 命令

説　明

（4） 使用例

シフト数およびシフト前32ビットデータを入力引数にセットし、ソフトウェアSHARNをサブルーチンコールします。

```
        MOV     #H'05,R0        ……シフト数を入力引数(R0)にセット
        BSR     SHARN           ……ソフトウェアSHARNをサブルーチンコール
        MOV. L  DATA,R1         ……シフト前32ビットデータを入力引数(R1)にセット
                :
                :
        .align  4
DATA    .data.l H'80000000
```

（5） 動作原理

(a) シフト回数がセットされているR0のビット4～ビット0の各ビットをテストし、“1”ならば表3.1に示す各ビットの重み分のシフトを16ビット右論理シフト命令(SHLR16)、8ビット右論理シフト命令(SHLR8)、2ビット右論理シフト命令(SHLR2)および1ビット右論理シフト命令(SHLR)を使用して行います。

表3.1　各ビットに対するシフト数と使用命令

ビット番号	重み	使用命令
ビット4	2^4=16	SHLR16
ビット3	2^3=8	SHLR8
ビット2	2^2=4	SHLR2(2回)
ビット1	2^1=2	SHLR2
ビット0	2^0=1	SHLR

(b) シフト前32ビットデータは、16ビット、8ビット、2ビットおよび1ビットの右論理シフト命令によりシフトさせているため、シフト前32ビットデータのMSBが“1”の場合、シフトにより空いた上位ビットには1ではなく0が格納されます。

そこで、図3.13に示すように、H'FFFFFFFFを設定したR2をシフト前32ビットデータと同じシフト数分右論理シフトさせ、シフト前32ビットデータのMSBが“1”ならば、シフト終了後、R2を反転させた値と論理和を取ることによりシフト数分の上位ビットを“1”に設定します。

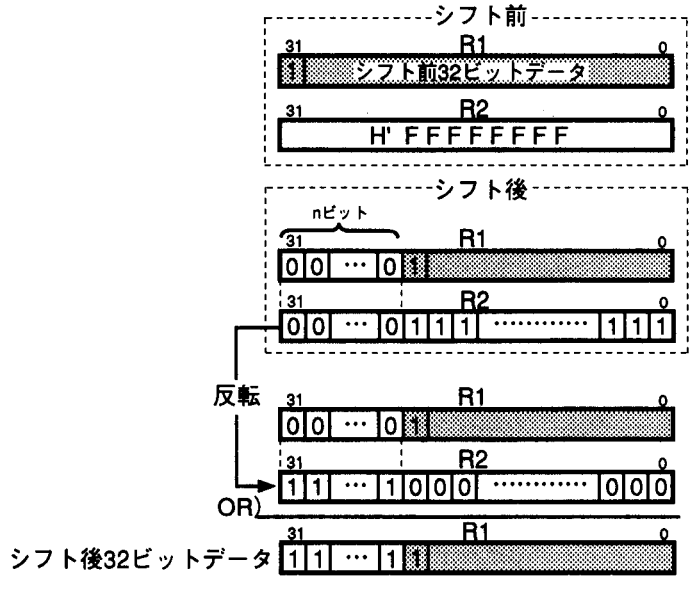
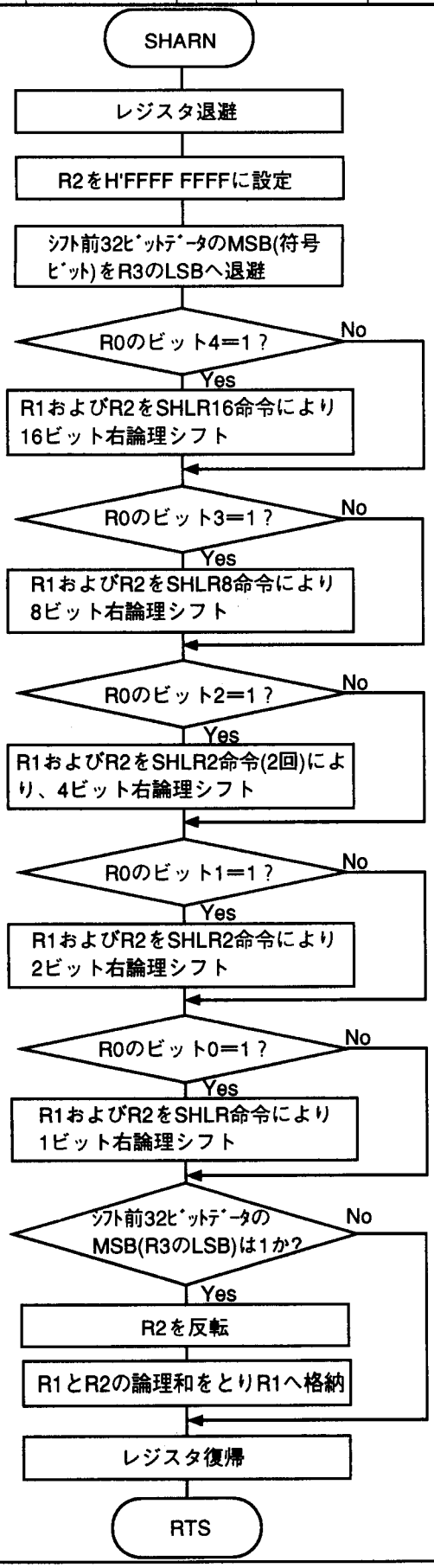


図3.13　多ビットシフト

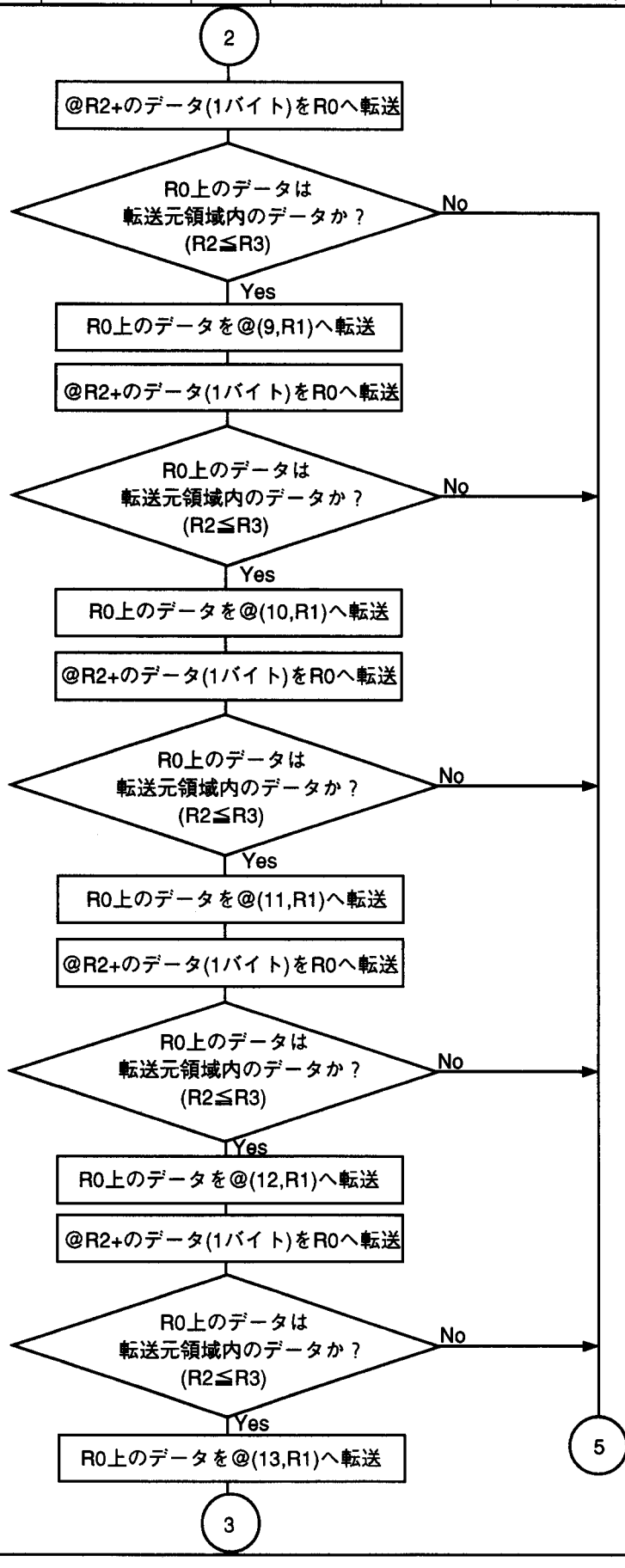
32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右算術ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHARN	使用機能	SHLR 2 命令 SHLR 8 命令 SHLR 16 命令

フローチャート



ブロック転送 (4バイト整合されていない)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MOVE	使用機能	MOV．B 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接 ﾃﾞｨｽﾌﾟﾚｰｽﾒﾝﾄ付きﾚｼﾞｽﾀ間接

フローチャート



32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右算術ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHARN	使用機能	SHLR2 命令 SHLR8 命令 SHLR16 命令

プログラムリスト

```
 58 00001046 000B               58           RTS                     ;
 59 00001048 62F6               59           MOV.L   @R15+,R2        ;
 60                             60           .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．4　32ビットデータの多ビットシフト（右論理シフト）

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLRN	使用機能	SHLR2 命令 SHLR8 命令 SHLR16 命令

機　能

32ビットデータを多ビット(0～31)右論理シフトします。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	シフト数(0～31)	R0	4
	シフト前32ビットデータ	R1	4
出　力	シフト後32ビットデータ	R1	4

内部レジスタ変化およびフラグ変化

実行前　→　実行後

R0	シフト数　→　変化なし
R1	シフト前32ビットデータ　→　シフト後32ビットデータ
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

－：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
36
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
0
ｽﾃｰﾄ数
19
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

プログラミング仕様のステート数は、31ビットシフトさせたときの値です。

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLRN	使用機能	SHLR2 命令 SHLR8 命令 SHLR16 命令

説　明

(1) 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数としてシフト数(0～31)をセットします。

R1：入力引数として、シフト前32ビットデータをセットします。

出力引数として、シフト後32ビットデータがセットされます。

(b) 図3.14にソフトウェアSHLRNの実行例を示します。

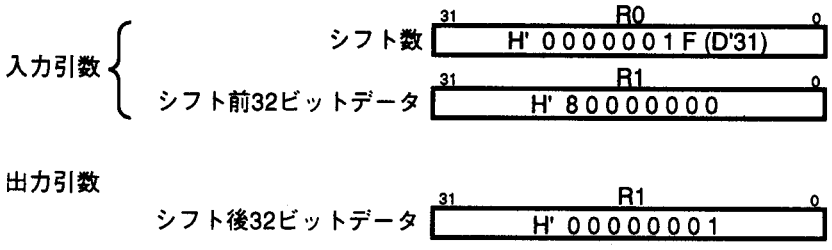


図3.14　ソフトウェアSHLRNの実行例

(2) 使用上の注意

シフト前32ビットデータがセットされていたR1にはシフト後32ビットデータがセットされるため、シフト前32ビットデータは破壊されます。ソフトウェアSHLRN実行後もシフト前32ビットデータを必要とする場合、シフト前32ビットデータをあらかじめ退避してください。

(3) 使用RAM説明

ソフトウェアSHLRNではRAMは使用していません。

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLRN	使用機能	SHLR2 命令 SHLR8 命令 SHLR16 命令

説　明

（4） 使用例

シフト数およびシフト前32ビットデータを入力引数にセットし、ソフトウェアSHLRNをサブルーチンコールします。

```
        MOV     #H'05,R0        ……シフト数を入力引数(R0)にセット
        BSR     SHLRN           ……ソフトウェアSHLRNをサブルーチンコール
        MOV. L  DATA,R1         ……シフト前32ビットデータを入力引数(R1)にセット
                :
                :
        .align  4
DATA    .data.l H'80000000
```

（5） 動作原理

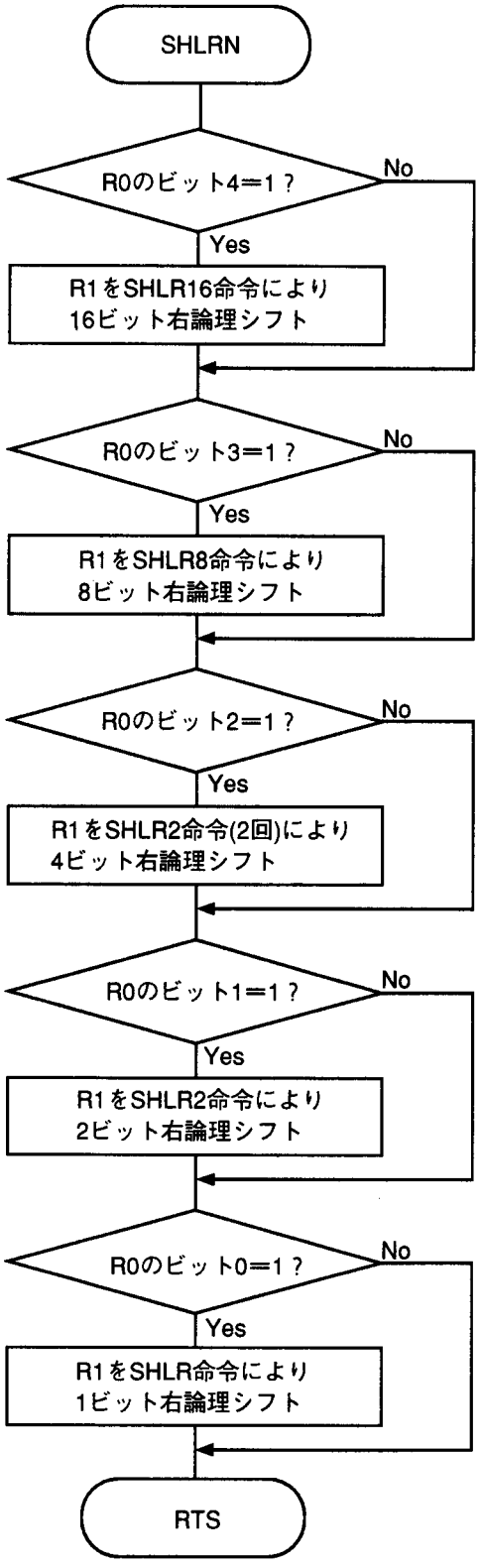
シフト回数がセットされているR0のビット4～ビット0の各ビットをテストし、“1”ならば表3.2に示す各ビットの重み分のシフトを16ビット右論理シフト命令(SHLR16)、8ビット右論理シフト命令(SHLR8)、2ビット右論理シフト命令(SHLR2)および、1ビット右論理シフト命令(SHLR)を使用して行います。

表3.2　各ビットに対するシフト数と使用命令

ビット番号	重み	使用命令
ビット4	2^4=16	SHLR16
ビット3	2^3=8	SHLR8
ビット2	2^2=4	SHLR2(2回)
ビット1	2^1=2	SHLR2
ビット0	2^0=1	SHLR

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLRN	使用機能	SHLR2 命令 SHLR8 命令 SHLR16 命令

フローチャート



32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (右論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLRN	使用機能	SHLR2 命令 SHLR8 命令 SHLR16 命令

プログラムリスト

```
                                    1  ;*******************************************************************************
                                    2  ;*                                                                             *
                                    3  ;*          NAME  :  n BITS SHIFT LOGICAL RIGHT (SHLRN)                        *
                                    4  ;*                                                                             *
                                    5  ;*******************************************************************************
                                    6  ;*                                                                             *
                                    7  ;*          ENTRY  :  R0    (NUMBER OF BIT SHIFTED)                            *
                                    8  ;*                    R1    (32 BIT DATA)                                      *
                                    9  ;*        RETURNS  :  R1    (SHIFT RESULT)                                     *
                                   10  ;*                                                                             *
                                   11  ;*******************************************************************************
00001000                           12          .SECTION A,CODE,LOCATE=H'1000
          00001000                 13  SHLRN   .EQU    $                  ; Entry point
00001000                           14  SHLRN1                             ;
00001000 C810                      15          TST     #B'00010000,R0     ; Bit4 = 1 ?
00001002 8900                      16          BT      SHLRN2             ; No
00001004 4129                      17          SHLR16  R1                 ; 16 bit shift logical right
00001006                           18  SHLRN2                             ;
00001006 C808                      19          TST     #B'00001000,R0     ; Bit3 = 1 ?
00001008 8900                      20          BT      SHLRN3             ; No
0000100A 4119                      21          SHLR8   R1                 ; 8 bit shift logical right
0000100C                           22  SHLRN3                             ;
0000100C C804                      23          TST     #B'00000100,R0     ; Bit2 = 1 ?
0000100E 8901                      24          BT      SHLRN4             ; No
00001010 4109                      25          SHLR2   R1                 ; 4 bit shift logical right
00001012 4109                      26          SHLR2   R1                 ;
00001014                           27  SHLRN4                             ;
00001014 C802                      28          TST     #B'00000010,R0     ; Bit1 = 1 ?
00001016 8900                      29          BT      SHLRN5             ; No
00001018 4109                      30          SHLR2   R1                 ; 2 bit shift logical right
0000101A                           31  SHLRN5                             ;
0000101A C801                      32          TST     #B'00000001,R0     ; Bit0 = 1 ?
0000101C 8900                      33          BT      SHLRN_END          ; No
0000101E 4101                      34          SHLR    R1                 ; 1 bit shift logical right
00001020                           35  SHLRN_END                          ;
00001020 000B                      36          RTS                        ;
00001022 0009                      37          NOP                        ;
                                   38          .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．5　32ビットデータの多ビットシフト（左論理シフト）

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (左論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLLN	使用機能	SHLL2 命令 SHLL8 命令 SHLL16 命令

機　能

32ビットデータを多ビット(0～31)左論理シフトします。

引　数

	内　容	格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	シフト数(0～31)	R0	4
	シフト前32ビットデータ	R1	4
出　力	シフト後32ビットデータ	R1	4

内部レジスタ変化およびフラグ変化

	実行前　→　実行後
R0	シフト数　→　変化なし
R1	シフト前32ビットデータ　→　シフト後32ビットデータ
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

—：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ（ﾊﾞｲﾄ）
36
ﾃﾞｰﾀﾒﾓﾘ（ﾊﾞｲﾄ）
0
ｽﾀｯｸ（ﾊﾞｲﾄ）
0
ｽﾃｰﾄ数
19
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

プログラミング仕様のステート数は、31ビットシフトさせたときの値です。

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (左論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLLN	使用機能	SHLL2 命令 SHLL8 命令 SHLL16 命令

説　明

（1） 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数としてシフト数(0～31)をセットします。

R1：入力引数として、シフト前32ビットデータをセットします。

出力引数として、シフト後32ビットデータがセットされます。

(b) 図3.15にソフトウェアSHLLNの実行例を示します。

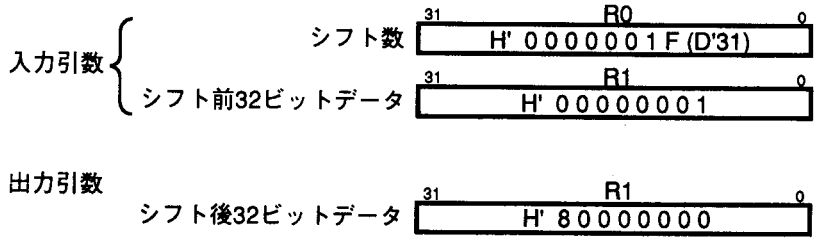


図3.15　ソフトウェアSHLLNの実行例

（2） 使用上の注意

シフト前32ビットデータがセットされていたR1にはシフト後32ビットデータがセットされるため、シフト前32ビットデータは破壊されます。ソフトウェアSHLLN実行後もシフト前32ビットデータを必要とする場合、シフト前32ビットデータをあらかじめ退避してください。

（3） 使用RAM説明

ソフトウェアSHLLNではRAMは使用していません。

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (左論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLLN	使用機能	SHLL2 命令 SHLL8 命令 SHLL16 命令

説　明

（4） 使用例

シフト数およびシフト前32ビットデータを入力引数にセットし、ソフトウェアSHLLNをサブルーチンコールします。

```
        MOV     #H'05,R0        ……シフト数を入力引数(R0)にセット
        BSR     SHLLN           ……ソフトウェアSHLLNをサブルーチンコール
        MOV. L  DATA,R1         ……シフト前32ビットデータを入力引数(R1)にセット
                :
                :
        .align  4
DATA    .data.l H'00000001
```

（5） 動作原理

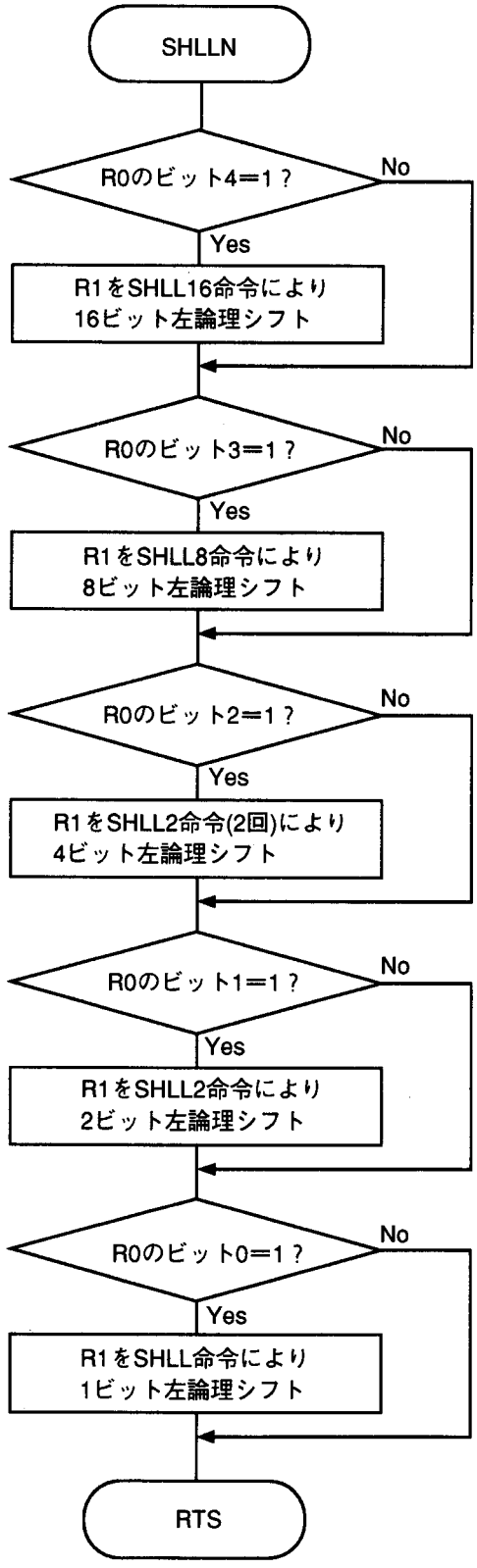
(a) シフト回数がセットされているR0のビット4～ビット0の各ビットをテストし、“1”ならば表3.3に示す各ビットの重み分のシフトを16ビット左論理シフト命令(SHLL16)、8ビット左論理シフト命令(SHLL8)、2ビット左論理シフト命令(SHLL2)および、1ビット右論理シフト命令(SHLL)を使用して行います。

表3.3　各ビットに対するシフト数と使用命令

ビット番号	重み	使用命令
ビット4	2^4=16	SHLL16
ビット3	2^3=8	SHLL8
ビット2	2^2=4	SHLL2(2回)
ビット1	2^1=2	SHLL2
ビット0	2^0=1	SHLL

32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (左論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLLN	使用機能	SHLL2 命令 SHLL8 命令 SHLL16 命令

フローチャート



32ﾋﾞｯﾄﾃﾞｰﾀの多ﾋﾞｯﾄｼﾌﾄ (左論理ｼﾌﾄ)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	SHLLN	使用機能	SHLL2 命令 SHLL8 命令 SHLL16 命令

プログラムリスト

```
                                    1  ;*******************************************************************************
                                    2  ;*                                                                             *
                                    3  ;*          NAME  :  n BITS SHIFT LOGICAL LEFT (SHLLN)                         *
                                    4  ;*                                                                             *
                                    5  ;*******************************************************************************
                                    6  ;*                                                                             *
                                    7  ;*         ENTRY  :  R0    (NUMBER OF BIT SHIFTED)                             *
                                    8  ;*                   R1    (32 BIT DATA)                                       *
                                    9  ;*       RETURNS  :  R1    (SHIFT RESULT)                                      *
                                   10  ;*                                                                             *
                                   11  ;*******************************************************************************
 12 00001000                       12          .SECTION A,CODE,LOCATE=H'1000
 13          00001000              13  SHLLN   .EQU     $               ; Entry point
 14 00001000                       14  SHLLN1                          ;
 15 00001000 C810                  15          TST     #B'00010000,R0  ; Bit4 = 1 ?
 16 00001002 8900                  16          BT      SHLLN2          ; No
 17 00001004 4128                  17          SHLL16  R1              ; 16 bit shift logical left
 18 00001006                       18  SHLLN2                          ;
 19 00001006 C808                  19          TST     #B'00001000,R0  ; Bit3 = 1 ?
 20 00001008 8900                  20          BT      SHLLN3          ; No
 21 0000100A 4118                  21          SHLL8   R1              ; 8 bit shift logical left
 22 0000100C                       22  SHLLN3                          ;
 23 0000100C C804                  23          TST     #B'00000100,R0  ; Bit2 = 1 ?
 24 0000100E 8901                  24          BT      SHLLN4          ; No
 25 00001010 4108                  25          SHLL2   R1              ; 4 bit shift logical left
 26 00001012 4108                  26          SHLL2   R1              ;
 27 00001014                       27  SHLLN4                          ;
 28 00001014 C802                  28          TST     #B'00000010,R0  ; Bit1 = 1 ?
 29 00001016 8900                  29          BT      SHLLN5          ; No
 30 00001018 4108                  30          SHLL2   R1              ; 2 bit shift logical left
 31 0000101A                       31  SHLLN5                          ;
 32 0000101A C801                  32          TST     #B'00000001,R0  ; Bit0 = 1 ?
 33 0000101C 8900                  33          BT      SHLLN_END       ; No
 34 0000101E 4100                  34          SHLL    R1              ; 1 bit shift logical left
 35 00001020                       35  SHLLN_END                       ;
 36 00001020 000B                  36          RTS                     ;
 37 00001022 0009                  37          NOP                     ;
 38                                38          .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．6　32ビットデータの最初の1の検出（Find First 1）

32ﾋﾞｯﾄﾃﾞｰﾀの最初の1の検出 (Find First 1)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	FIND1	使用機能	SHLL 命令

機　能

32ビットデータのMSBから順に各ビットの判定を行い、最初に“1”が検出されたビットの番号(0～31)を求めます。

引　数

	内　容	格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	検出を行う32ビットデータ	R0	4
出　力	最初に“1”が検出されたビットの番号(0～31)	R1	4

内部レジスタ変化およびフラグ変化

実行前　→　実行後

R0	検出を行う32ビットデータ　→　変化
R1	不定　→　最初に“1”が検出たれたビットの番号
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

－：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ（ﾊﾞｲﾄ）
16
ﾃﾞｰﾀﾒﾓﾘ（ﾊﾞｲﾄ）
0
ｽﾀｯｸ（ﾊﾞｲﾄ）
0
ｽﾃｰﾄ数
29
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

プログラミング仕様のステート数は、32ビットデータがH'10000000のときの値です。

32ﾋﾞｯﾄﾃﾞｰﾀの最初の1の検出 (Find First 1)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	FIND1	使用機能	SHLL 命令

説　明

（1） 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数として、検出を行う32ビットデータをセットします。

R1：出力引数として、“1”が最初に検出されたビットの番号(0～31)がセットされます。

(b) 図3.16にソフトウェアFIND1の実行例を示します。

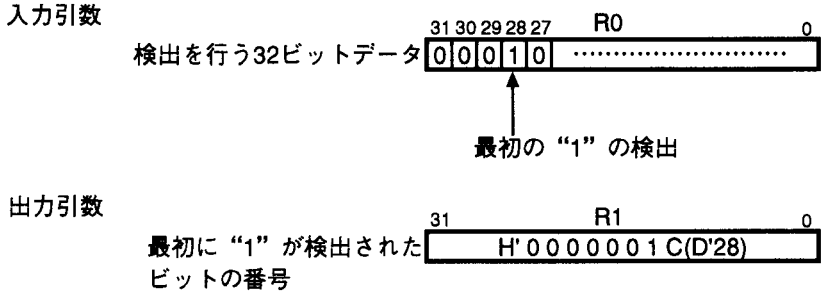


図3.16　ソフトウェアFIND1の実行例

（2） 使用上の注意

検出を行う32ビットデータがセットされていたR0は、ソフトウェアFIND1の実行により内容が変化します。ソフトウェアFIND1実行後も検出を行う32ビットデータを必要とする場合、検出を行う32ビットデータをあらかじめ退避してください。

（3） 使用RAM説明

ソフトウェアFIND1では、RAMは使用していません。

32ﾋﾞｯﾄﾃﾞｰﾀの最初の1の検出 (Find First 1)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	FIND1	使用機能	SHLL 命令

説　明

(4) 使用例

検出を行う32ビットデータを入力引数にセットしてからソフトウェアFIND1 をサブルーチンコールします。

```
        BSR       FIND1          ……ソフトウェアFIND1をサブルーチンコール
        MOV. L    DATA,R0        ……検出を行う32ビットデータを入力引数(R0)
                                     にセット
                  ⋮
        .align    4
DATA    .data.l   H'12345678
```

(5) 動作原理

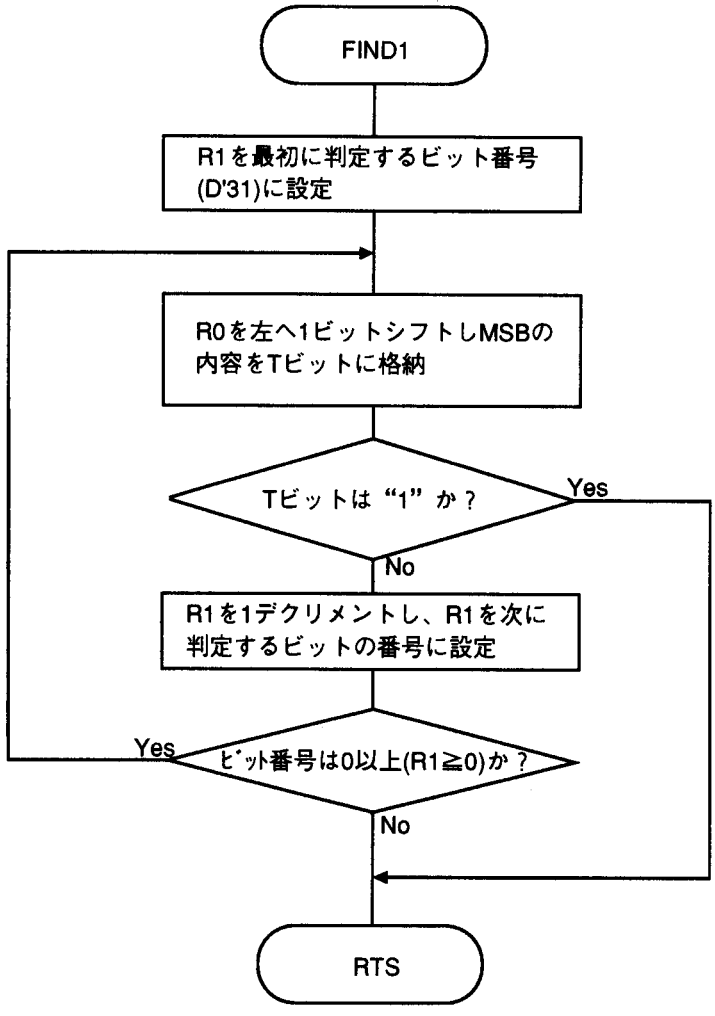
(a) SHLL命令によって、検出を行う32ビットデータのビット31から順にTビットにセットし、ビット判定を行います。

(b) R1はビット判定を行うビット番号のポインタとして使用します。R1には初期値として、最初にビット判定を行うビット番号の31を設定します。R1はビット判定後、1デクリメントされ、次にビット判定を行うビット番号を示します。

(c) ソフトウェアFIND1は、“1”が検出されたとき、または、ビット番号(R1)<0になったときに終了します。“1”の検出により終了した場合、R1は“1”が検出されたビットの番号を示しています。ビット番号(R1)<0により終了した場合、R1はH'FFFFFFFFとなります。

32ﾋﾞｯﾄﾃﾞｰﾀの最初の1の検出 (Find First 1)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	FIND1	使用機能	SHLL 命令

フローチャート



32ﾋﾞｯﾄﾃﾞｰﾀの最初の1の検出 (Find First 1)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	FIND1	使用機能	SHLL 命令

プログラムリスト

```
  1                                     1    ;*******************************************************************************
  2                                     2    ;*                                                                             *
  3                                     3    ;*          NAME  :  FIND FIRST 1 (FIND1)                                      *
  4                                     4    ;*                                                                             *
  5                                     5    ;*******************************************************************************
  6                                     6    ;*                                                                             *
  7                                     7    ;*          ENTRY  :  R0    (32 BIT DATA)                                      *
  8                                     8    ;*        RETURNS  :  R1    (BIT NUMBER)                                       *
  9                                     9    ;*                                                                             *
 10                                    10    ;*******************************************************************************
 11 00001000                           11            .SECTION A,CODE,LOCATE=H'1000
 12          00001000                  12    FIND1   .EQU    $               ; Entry point
 13 00001000 E11F                      13            MOV     #D'31,R1        ; Initialize R1
 14 00001002                           14    FIND11                          ;
 15 00001002 4000                      15            SHLL    R0              ; T bit = 1 ?
 16 00001004 8902                      16            BT      FIND_END        ; Yes
 17 00001006 71FF                      17            ADD     #H'FF,R1        ; Decrement bit number
 18 00001008 4111                      18            CMP/PZ  R1              ; Bit number >= 0 ?
 19 0000100A 89FA                      19            BT      FIND11          ; Yes
 20 0000100C                           20    FIND_END                        ;
 21 0000100C 000B                      21            RTS                     ;
 22 0000100E 0009                      22            NOP                     ;
 23                                    23            .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．7　64ビット＋64ビット＝64ビット（符号なし）

64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDU64	使用機能	ADDC 命令

機　能

被加数(符号なし64ビット)と加数(符号なし64ビット)の加算を行い、加算結果(符号なし64ビット)を求めます。このとき桁上がりの有無をTビットへ設定します。

引　数

	内　容	格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	被加数(符号なし64ビット)の上位32ビット	R0	4
	被加数(符号なし64ビット)の下位32ビット	R1	4
	加数(符号なし64ビット)の上位32ビット	R2	4
	加数(符号なし64ビット)の下位32ビット	R3	4
出　力	加算結果(符号なし64ビット)の上位32ビット	R0	4
	加算結果(符号なし64ビット)の下位32ビット	R1	4
	桁上がりの有無(有:T=1,無:T=0)	Tビット(SR)	4

内部レジスタ変化およびフラグ変化

	実行前 → 実行後
R0	被加数の上位32ﾋﾞｯﾄ → 加算結果の上位32ﾋﾞｯﾄ
R1	被加数の下位32ﾋﾞｯﾄ → 加算結果の下位32ﾋﾞｯﾄ
R2	加数の上位32ﾋﾞｯﾄ → 変化なし
R3	加数の下位32ﾋﾞｯﾄ → 変化なし
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

T [※]　－：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
8
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
0
ｽﾃｰﾄ数
5
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

64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDU64	使用機能	ADDC 命令

説　明

(1) 機能説明

(a) 引数の詳細は以下の通りです。

R0： 入力引数として、被加数(符号なし64ビット)の上位32ビットをセットします。
　　出力引数として、加算結果(符号なし64ビット)の上位32ビットがセットされます。

R1： 入力引数として、被加数(符号なし64ビット)の下位32ビットをセットします。
　　出力引数として、加算結果(符号なし64ビット)の下位32ビットがセットされます。

R2： 入力引数として、加数(符号なし64ビット)の上位32ビットをセットします。

R3： 入力引数として、加数(符号なし64ビット)の下位32ビットをセットします。

Tビット(SR)：ソフトウェアADDU64実行後の桁上がりの有無を示します。

Tビット=1　桁上がりが発生したことを示します。

Tビット=0　桁上がりが発生しなかったことを示します。

(b) 図3.17にソフトウェアADDU64の実行例を示します。

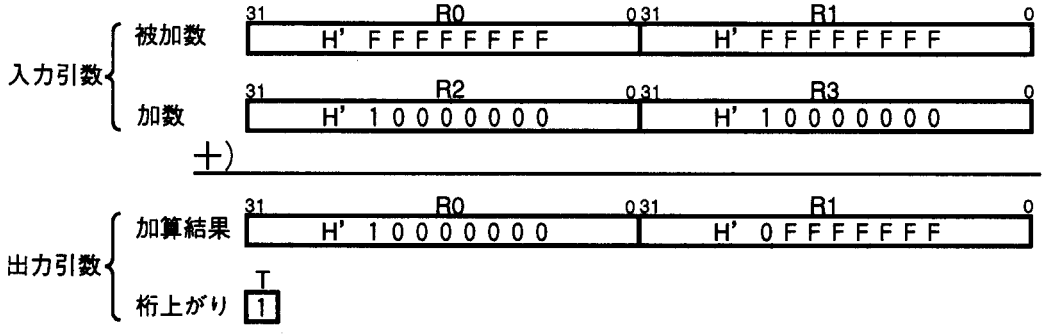


図3.17　ソフトウェアADDU64の実行例

(2) 使用上の注意

被加数のセットされていたR1およびR2には、加算結果がセットされるため、被加数は破壊されます。
ソフトウェアADDU64実行後も、被加数を必要とする場合、被加数をあらかじめ退避してください。

(3) 使用RAM説明

ソフトウェアADDU64ではRAMは使用していません。

64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDU64	使用機能	ADDC 命令

説　明

(4) 使用例

被加数および加数を入力引数にセットしてからソフトウェアADDU64をサブルーチンコールします。

```
        MOV. L  DATA1,R0        ……被加数(上位32ビット)を入力引数にセット
        MOV. L  DATA2,R1        ……被加数(下位32ビット)を入力引数にセット
        MOV. L  DATA3,R2        ……加数(上位32ビット)を入力引数にセット
        BSR     ADDU64          ……ソフトウェアADDU64をサブルーチンコール
        MOV. L  DATA4,R3        ……加数(下位32ビット)を入力引数にセット
        BT      ERROR           ……桁上がりが発生した場合、エラー処理
                                    ルーチンへ分岐
          :
        .align  4
DATA1   .data.l H'FFFFFFFF
DATA2   .data.l H'FFFFFFFF
DATA3   .data.l H'10000000
DATA4   .data.l H'10000000
```

(5) 動作原理

図3.18に示すように、下位から32ビット単位でキャリ付き加算命令(ADDC)により加算を繰り返します。

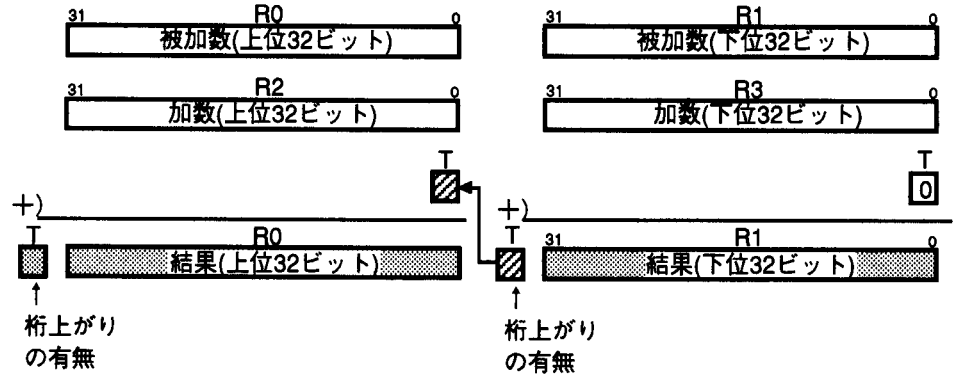
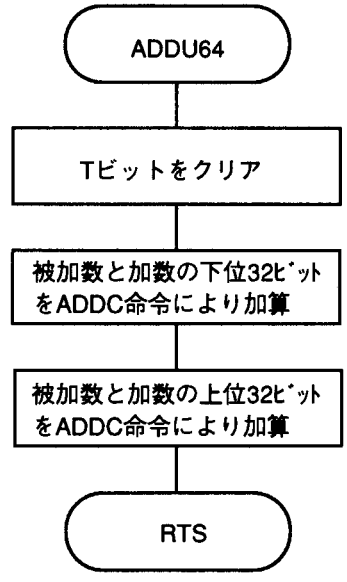


図3.18　符号なし加算

64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDU64	使用機能	ADDC 命令

フローチャート



64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDU64	使用機能	ADDC 命令

プログラムリスト

```
  1                                      1    ;*************************************************************************
  2                                      2    ;*                                                                       *
  3                                      3    ;*          NAME  :  64 BIT UNSIGNED BINARY ADDITION (ADDU64)            *
  4                                      4    ;*                                                                       *
  5                                      5    ;*************************************************************************
  6                                      6    ;*                                                                       *
  7                                      7    ;*         ENTRY  :  R0    (UPPER 32 BIT AUGEND)                         *
  8                                      8    ;*                   R1    (LOWER 32 BIT AUGEND)                         *
  9                                      9    ;*                   R2    (UPPER 32 BIT ADDEND)                         *
 10                                     10    ;*                   R3    (LOWER 32 BIT ADDEND)                         *
 11                                     11    ;*       RETURNS  :  R0    (UPPER 32 BIT SUM)                            *
 12                                     12    ;*                   R1    (LOWER 32 BIT SUM)                            *
 13                                     13    ;*                   T BIT (CARRY -> TRUE;T=1,FALSE;T=0)                 *
 14                                     14    ;*                                                                       *
 15                                     15    ;*************************************************************************
 16 00001000                            16            .SECTION A,CODE,LOCATE=H'1000
 17          00001000                   17    ADDU64  .EQU    $               ; Entry point
 18 00001000 0008                       18            CLRT                    ; Clear T bit
 19 00001002 313E                       19            ADDC    R3,R1           ; Lower 32 bit augend + Lower 32 bit addend
 20 00001004 000B                       20            RTS                     ;
 21 00001006 302E                       21            ADDC    R2,R0           ; Upper 32 bit augend + Upper 32 bit addend
 22                                     22            .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．8　64ビット＋64ビット＝64ビット（符号付き）

64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDS64	使用機能	ADDV 命令

機　能

被加数(符号付き64ビット)と加数(符号付き64ビット)の加算を行い、加算結果(符号付き64ビット)を求めます。このとき、オーバフローまたはアンダフローの有無をTビットへ設定します。オーバフローとアンダフローの区別は行っておりません。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	被加数(符号付き64ビット)の上位32ビット	R0	4
	被加数(符号付き64ビット)の下位32ビット	R1	4
	加数(符号付き64ビット)の上位32ビット	R2	4
	加数(符号付き64ビット)の下位32ビット	R3	4
出　力	加算結果(符号付き64ビット)の上位32ビット	R0	4
	加算結果(符号付き64ビット)の下位32ビット	R1	4
	ｵｰﾊﾞﾌﾛｰ,ｱﾝﾀﾞﾌﾛｰの有無(有:T=1,無:T=0)	Tビット(SR)	4

内部レジスタ変化およびフラグ変化

	実行前 → 実行後
R0	被加数の上位32ﾋﾞｯﾄ → 加算結果の上位32ﾋﾞｯﾄ
R1	被加数の下位32ﾋﾞｯﾄ → 加算結果の下位32ﾋﾞｯﾄ
R2	加数の上位32ﾋﾞｯﾄ → 変化なし
R3	加数の下位32ﾋﾞｯﾄ → 変化なし
R4	ワーク
R5	ワーク
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

T ※　－：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
28
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
8
ｽﾃｰﾄ数
15
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

64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDS64	使用機能	ADDV 命令

説　明

（1） 機能説明

(a) 引数の詳細は以下の通りです。

R0： 入力引数として、被加数(符号付き64ビット)の上位32ビットをセットします。
　　出力引数として、加算結果(符号付き64ビット)の上位32ビットがセットされます。

R1： 入力引数として、被加数(符号付き64ビット)の下位32ビットをセットします。
　　出力引数として、加算結果(符号付き64ビット)の下位32ビットがセットされます。

R2： 入力引数として、加数(符号付き64ビット)の上位32ビットをセットします。

R3： 入力引数として、加数(符号付き64ビット)の下位32ビットをセットします。

Tビット(SR)：ソフトウェアADDS64実行後のオーバフロー,アンダフローの有無を示します。

Tビット=1　オーバフロー,アンダフローが発生したことを示します。

Tビット=0　オーバフロー,アンダフローが発生しなかったことを示します。

(b) 図3.19にソフトウェアADDS64の実行例を示します。

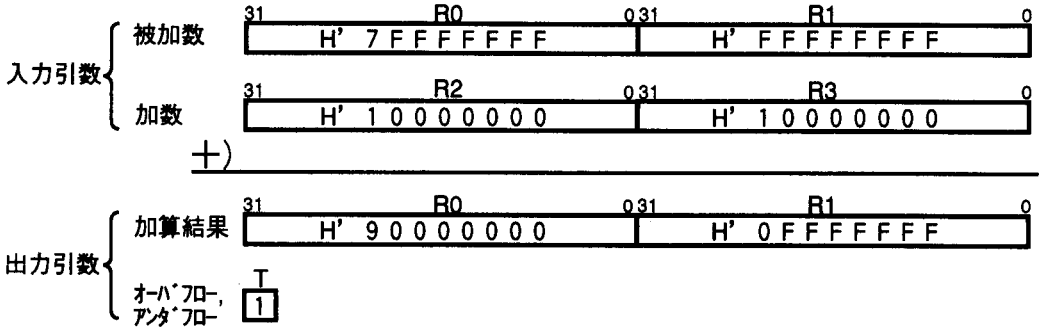


図3.19　ソフトウェアADDS64の実行例

（2） 使用上の注意

被加数のセットされていたR1およびR2には、加算結果がセットされるため、被加数は破壊されます。
ソフトウェアADDS64実行後も、被加数を必要とする場合，被加数をあらかじめ退避してください。

（3） 使用RAM説明

ソフトウェアADDS64ではRAMは使用していません。

64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDS64	使用機能	ADDV 命令

説　明

（4） 使用例

被加数および加数を入力引数にセットしてからソフトウェアADDS64をサブルーチンコールします。

```
        MOV. L  DATA1,R0        ……被加数(上位32ビット)を入力引数にセット
        MOV. L  DATA2,R1        ……被加数(下位32ビット)を入力引数にセット
        MOV. L  DATA3,R2        ……加数(上位32ビット)を入力引数にセット
        BSR     ADDS64          ……ソフトウェアADDS64をサブルーチンコール
        MOV. L  DATA4,R3        ……加数(下位32ビット)を入力引数にセット
        BT      ERROR           ……オーバフロー,アンダフローを発生した場合、
                                    エラー処理ルーチンへ分岐
                :
        .align  4
DATA1   .data.l H'7FFFFFFF
DATA2   .data.l H'FFFFFFFF
DATA3   .data.l H'10000000
DATA4   .data.l H'10000000
```

（5） 動作原理

(a) 被加数と加数の下位32ビットをキャリ付加算命令(ADDC)により加算します。加算後、桁上がりの有無がTビットに示されます。（図3.20-①）

(b) 桁上がりを被加数の上位32ビットに加算します。(a)のTビットの内容をR4に格納し、被加数の上位32ビットとオーバフロー,アンダフローのチェック付加算命令(ADDV)により加算します。桁上がりが発生していた場合には、被加数の上位32ビットに“1”が加算され、発生していなかった場合には、被加数の上位32ビットに0が加算されます。加算後、オーバフロー,アンダフローの有無がTビットに示されます。（図3.20-②）

(c) (b)の加算結果と加数の上位32ビットをオーバフロー,アンダフローのチェック付加算命令(ADDV)により加算します。加算後、オーバフロー,アンダフローの有無がTビットに示されます。（図3.20-③）

(d) (b)および(c)のTビットの内容の論理和をとり、結果をTビットに格納します。Tビット=1のとき、オーバフロー,アンダフローが発生したことを示し、Tビット=0のとき、オーバフロー,アンダフローが発生しなかったことを示します。

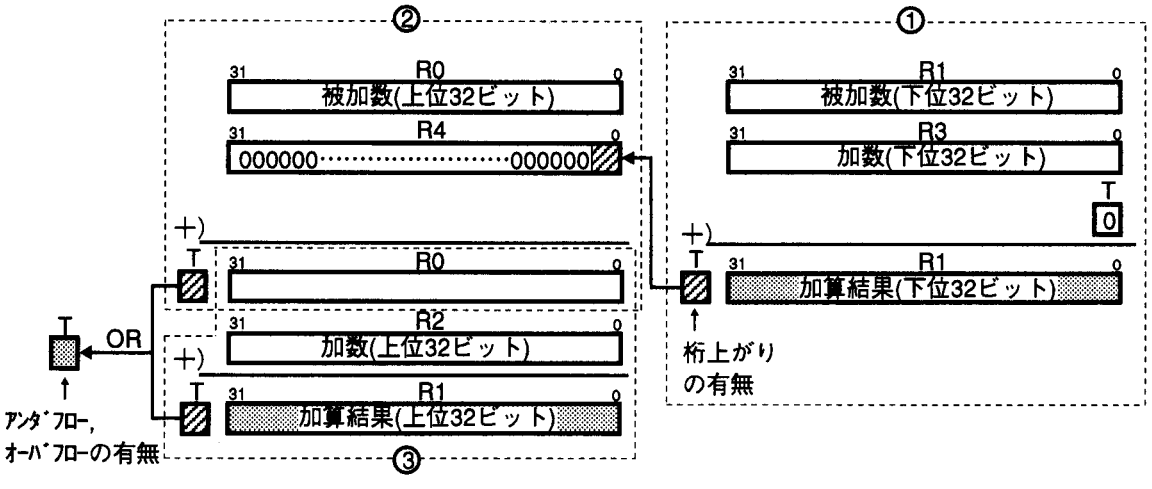
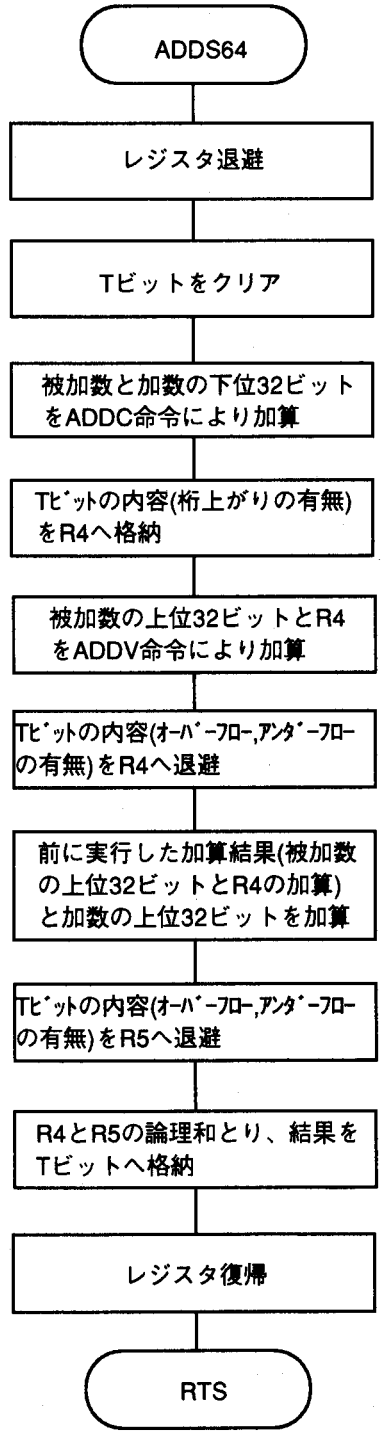


図3.20　符号付き加算

64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDS64	使用機能	ADDV 命令

フローチャート



64ﾋﾞｯﾄ+64ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	ADDS64	使用機能	ADDV命令

プログラムリスト

```
  1                                     1   ;*********************************************************************
  2                                     2   ;*                                                                   *
  3                                     3   ;*          NAME  :  64 BIT SIGNED BINARY ADDITION (ADDS64)          *
  4                                     4   ;*                                                                   *
  5                                     5   ;*********************************************************************
  6                                     6   ;*                                                                   *
  7                                     7   ;*         ENTRY  :  R0    (UPPER 32 BIT AUGEND)                     *
  8                                     8   ;*                   R1    (LOWER 32 BIT AUGEND)                     *
  9                                     9   ;*                   R2    (UPPER 32 BIT ADDEND)                     *
 10                                    10   ;*                   R3    (LOWER 32 BIT ADDEND)                     *
 11                                    11   ;*       RETURNS  :  R0    (UPPER 32 BIT SUM)                        *
 12                                    12   ;*                   R1    (LOWER 32 BIT SUM)                        *
 13                                    13   ;*                   T BIT (OVERFLOW/UNDERFLOW -> TRUE;T=1,FALSE;T=0)*
 14                                    14   ;*                                                                   *
 15                                    15   ;*********************************************************************
 16 00001000                           16           .SECTION A,CODE,LOCATE=H'1000
 17          00001000                  17   ADDS64  .EQU    $               ; Entry point
 18 00001000 2F46                      18           MOV.L   R4,@-R15        ; Escape register
 19 00001002 2F56                      19           MOV.L   R5,@-R15        ;
 20 00001004 0008                      20           CLRT                    ; Clear T bit
 21 00001006 313E                      21           ADDC    R3,R1           ; Lower 32 bit augend + Lower 32 bit addend
 22 00001008 0429                      22           MOVT    R4              ; R4 <- Carry
 23 0000100A 304F                      23           ADDV    R4,R0           ; Upper 32 bit augend + Carry
 24 0000100C 0429                      24           MOVT    R4              ; R4 <- Overflow / Underflow
 25 0000100E 302F                      25           ADDV    R2,R0           ; Upper 32 bit augend + Upper 32 bit addend
 26 00001010 0529                      26           MOVT    R5              ; R5 <- Overflow / Underflow
 27 00001012 245B                      27           OR      R5,R4           ; R4 <- R5 or R4
 28 00001014 4401                      28           SHLR    R4              ; T bit <- Overflow / Underflow
 29 00001016 65F6                      29           MOV.L   @R15+,R5        ; Return register
 30 00001018 000B                      30           RTS                     ;
 31 0000101A 64F6                      31           MOV.L   @R15+,R4        ;
 32                                    32           .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．9　32ビット×32ビット=64ビット（符号なし）

32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULU32	使用機能	MULU 命令 SWAP 命令

機　能

被乗数(符号なし32ビット)と乗数(符号なし32ビット)の乗算を行い、積(符号なし64ビット)を求めます。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	被乗数(符号なし32ビット)	R0	4
	乗　数(符号なし32ビット)	R1	4
出　力	積(符号なし64ビット)の上位32ビット	R2	4
	積(符号なし64ビット)の下位32ビット	R3	4

内部レジスタ変化およびフラグ変化

	実行前 → 実行後
R0	被乗数(符号なし32ﾋﾞｯﾄ) → 変化なし
R1	乗　数(符号なし32ﾋﾞｯﾄ) → 変化なし
R2	不定 → 乗算結果(上位32ﾋﾞｯﾄ)
R3	不定 → 乗算結果(下位32ﾋﾞｯﾄ)
R4	ワーク
R5	ワーク
R6	ワーク
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

ー：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
76
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
24
ｽﾃｰﾄ数
35
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

プログラミング仕様のステート数は、H'FFFFFFFF×H'FFFFFFFFを計算した場合の値です。

32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULU32	使用機能	MULU 命令 SWAP 命令

説　明

(1) 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数として、被乗数(符号なし32ビット)をセットします。

R1：入力引数として、乗数(符号なし32ビット)をセットします。

R2：出力引数として、積(符号なし64ビット)の上位32ビットがセットされます。

R3：出力引数として、積(符号なし64ビット)の下位32ビットがセットされます。

(b) 図3.21にソフトウェアMULU32の実行例を示します。

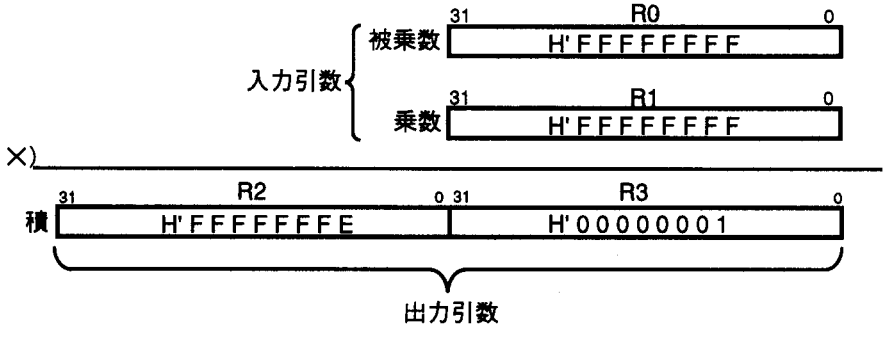


図3.21　ソフトウェアMULU32の実行例

(2) 使用上の注意

ソフトウェアMULU32では、使用上の注意は特にありません。

(3) 使用RAM説明

ソフトウェアMULU32ではRAMは使用していません。

32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULU32	使用機能	MULU 命令 SWAP 命令

説　明

（4） 使用例

被乗数および乗数をセットしてからソフトウェアMULU32をサブルーチンコールします。

```
                MOV. L   DATA1,R0        ……被乗数を入力引数(R0)にセット
                BSR      MULU32          ……ソフトウェアMULU32をサブルーチンコール
                MOV. L   DATA2,R1        ……乗数を入力引数(R1)にセット
                  ⋮
                .align   4
DATA1           .data.l  H'FFFFFFFF
DATA2           .data.l  H'FFFFFFFF
```

（5） 動作原理

図3.22に示すように、16ビット単位で乗算を行い、部分積(1～4)を求め、これらを加算した結果が積(64ビット)となります。部分積の乗算は16ビット符号なし乗算命令(MULU)を使用します。

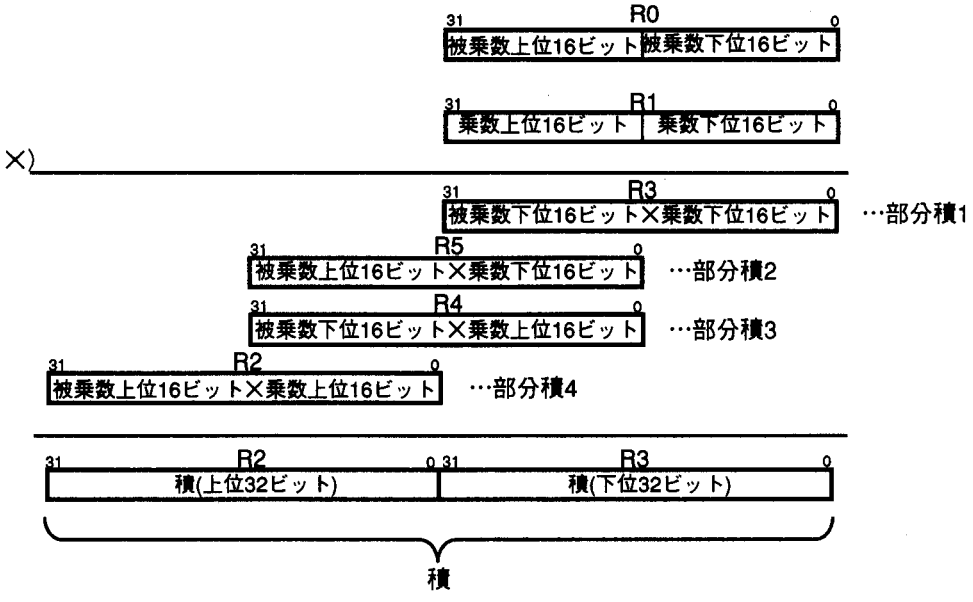
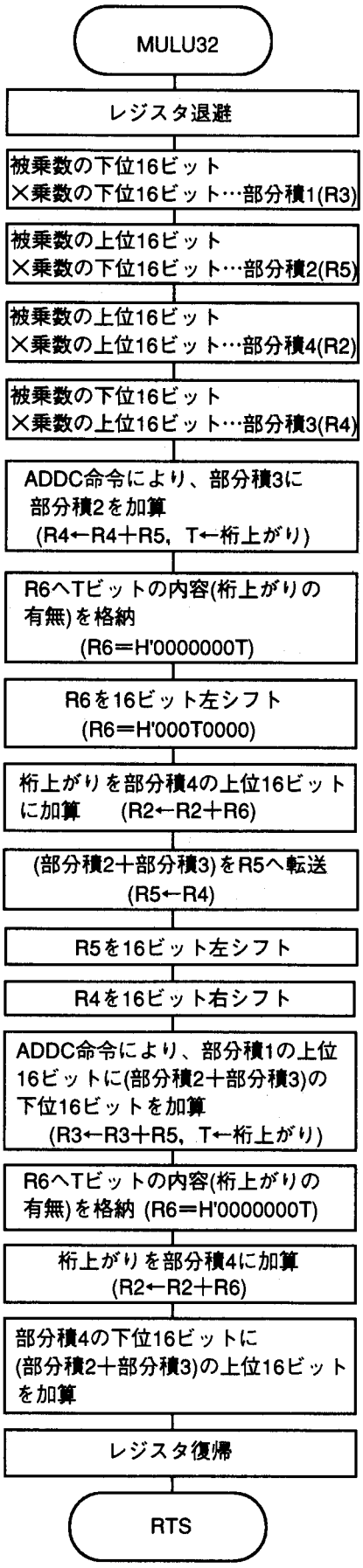


図3.22　乗算

32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULU32	使用機能	MULU 命令 SWAP 命令

フローチャート



32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULU32	使用機能	MULU 命令 SWAP 命令

プログラムリスト

```
                                    1  ;*******************************************************************************
                                    2  ;*                                                                             *
                                    3  ;*          NAME  :  32 BIT UNSIGNED MULTIPLATION (MULU32)                     *
                                    4  ;*                                                                             *
                                    5  ;*******************************************************************************
                                    6  ;*                                                                             *
                                    7  ;*          ENTRY  :  R0    (MULTIPLICAND)                                     *
                                    8  ;*                    R1    (MULTIPLIER)                                       *
                                    9  ;*        RETURNS  :  R2    (UPPER 32 BIT PRODUCT)                             *
                                   10  ;*                    R3    (LOWER 32 BIT PRODUCT)                             *
                                   11  ;*                                                                             *
                                   12  ;*******************************************************************************
00001000                           13          .SECTION A,CODE,LOCATE=H'1000
          00001000                 14  MULU32  .EQU    $                   ; Entry point
00001000 4F12                      15          STS.L   MACL,@-R15          ; Escape register
00001002 2F46                      16          MOV.L   R4,@-R15            ;
00001004 2F56                      17          MOV.L   R5,@-R15            ;
00001006 2F66                      18          MOV.L   R6,@-R15            ;
                                   19                                      ;
00001008 201E                      20          MULU    R1,R0               ; Lower 16 bit * lower 16 bit -> R3
0000100A 6009                      21          SWAP.W  R0,R0               ;
0000100C 031A                      22          STS     MACL,R3             ;
0000100E 201E                      23          MULU    R1,R0               ; Upper 16 bit * lower 16 bit -> R5
00001010 6119                      24          SWAP.W  R1,R1               ;
00001012 051A                      25          STS     MACL,R5             ;
00001014 201E                      26          MULU    R1,R0               ; Upper 16 bit * upper 16 bit -> R2
00001016 6009                      27          SWAP.W  R0,R0               ;
00001018 021A                      28          STS     MACL,R2             ;
0000101A 201E                      29          MULU    R1,R0               ; Lower 16 bit * upper 16 bit -> R4
0000101C 6119                      30          SWAP.W  R1,R1               ;
0000101E 041A                      31          STS     MACL,R4             ;
                                   32                                      ;
00001020 0008                      33          CLRT                        ;
00001022 345E                      34          ADDC    R5,R4               ;
00001024 0629                      35          MOVT    R6                  ; R6 <- Carry
00001026 4628                      36          SHLL16  R6                  ;
00001028 326C                      37          ADD     R6,R2               ; Carry = 1  R2 <- R2 + H'00010000
                                   38                                      ; Carry = 0  R2 <- R2 + H'00000000
0000102A 6543                      39          MOV     R4,R5               ;
0000102C 4528                      40          SHLL16  R5                  ;
0000102E 4429                      41          SHLR16  R4                  ;
                                   42                                      ;
00001030 0008                      43          CLRT                        ;
00001032 335E                      44          ADDC    R5,R3               ;
00001034 0629                      45          MOVT    R6                  ; R6 <- Carry
00001036 326C                      46          ADD     R6,R2               ; Carry = 1  R2 <- R2 + H'00000001
                                   47                                      ; Carry = 0  R2 <- R2 + H'00000000
00001038 324C                      48          ADD     R4,R2               ;
                                   49                                      ;
0000103A 66F6                      50          MOV.L   @R15+,R6            ; Return register
0000103C 65F6                      51          MOV.L   @R15+,R5            ;
0000103E 64F6                      52          MOV.L   @R15+,R4            ;
00001040 000B                      53          RTS                         ;
00001042 4F16                      54          LDS.L   @R15+,MACL          ;
                                   55          .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．10　32ビット×32ビット＝64ビット（符号付き）

32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULS32	使用機能	MULU 命令 SWAP 命令 NEGC 命令

機　能

被乗数(符号付き32ビット)と乗数(符号付き32ビット)の乗算を行い、積(符号付き64ビット)を求めます。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	被乗数(符号付き32ビット)	R0	4
	乗　数(符号付き32ビット)	R1	4
出　力	積(符号付き64ビット)の上位32ビット	R2	4
	積(符号付き64ビット)の下位32ビット	R3	4

内部レジスタ変化およびフラグ変化

	実行前 → 実行後
R0	被乗数(符号なし32ﾋﾞｯﾄ) → 変化なし
R1	乗　数(符号なし32ﾋﾞｯﾄ) → 変化
R2	不定 → 乗算結果(上位32ﾋﾞｯﾄ)
R3	不定 → 乗算結果(下位32ﾋﾞｯﾄ)
R4	ワーク
R5	ワーク
R6	ワーク
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※
ー：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
92
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
16
ｽﾃｰﾄ数
48
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

プログラミング仕様のステート数は、H'7FFFFFFF×H'80000000を計算した場合の値です。

32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULS32	使用機能	MULU 命令 SWAP 命令 NEGC 命令

説　明

(1) 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数として、被乗数(符号付き32ビット)をセットします。

R1：入力引数として、乗数(符号付き32ビット)をセットします。

R2：出力引数として、積(符号付き64ビット)の上位32ビットがセットされます。

R3：出力引数として、積(符号付き64ビット)の下位32ビットがセットされます。

(b) 図3.2 3にソフトウェアMULS32の実行例を示します。

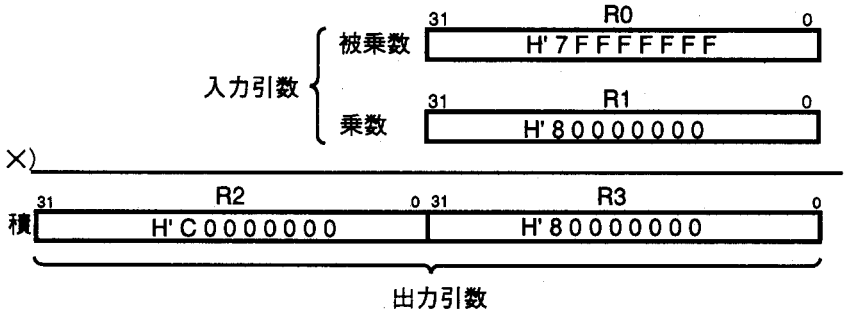


図3.2 3　ソフトウェアMULS32の実行例

(2) 使用上の注意

乗数がセットされていたR1は、ソフトウェアMULS32の実行により内容が変化します。ソフトウェアMULS32実行後も、乗数を必要とする場合、乗数をあらかじめ退避してください。

(3) 使用RAM説明

ソフトウェアMULS32ではRAMは使用していません。

32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULS32	使用機能	MULU 命令 SWAP 命令 NEGC 命令

説　明

（4） 使用例

被乗数および乗数をセットしてからソフトウェアMULS32をサブルーチンコールします。

```
        MOV. L  DATA1,R0        ……被乗数を入力引数(R0)にセット
        BSR     MULS32          ……ソフトウェアMULS32をサブルーチンコール
        MOV. L  DATA2,R1        ……乗数を入力引数(R1)にセット
                :
                :
        .align  4
DATA1   .data.l H'7FFFFFFF
DATA2   .data.l H'80000000
```

（5） 動作原理

(a) 図3.24に示すように、16ビット単位で乗算を行い、部分積(1～4)を求め、これらを加算して64ビットの積を求めます。部分積の乗算は、16ビット符号なし乗算命令(MULU)を使用するので、被乗数および乗数が負の場合は、正に変換してから乗算を行います。

(b) 積は正で求められるので、表3.4に示すように、積が負か正の判定は被除数と乗数の最上位ビットの排他的論理和をとり行います。

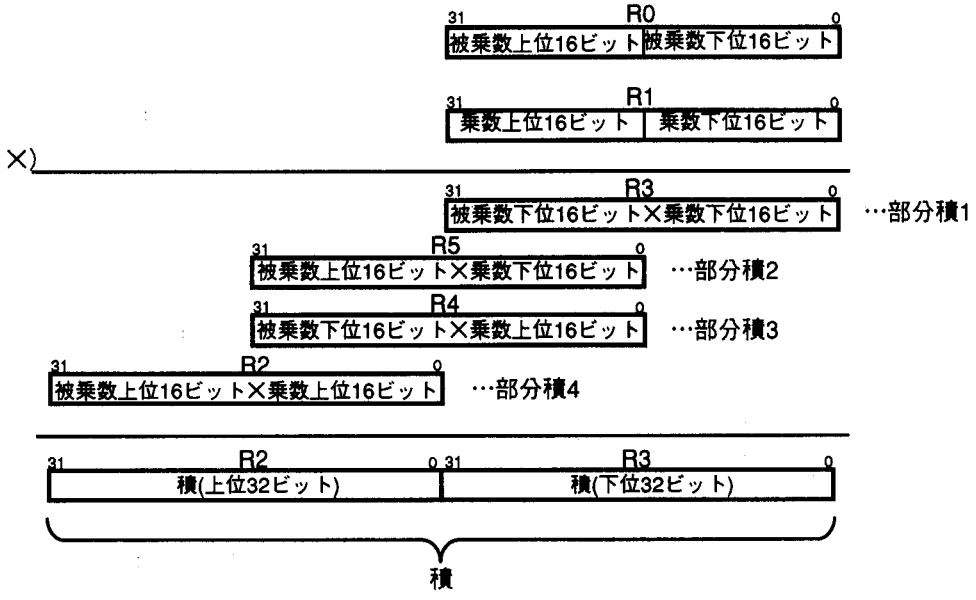


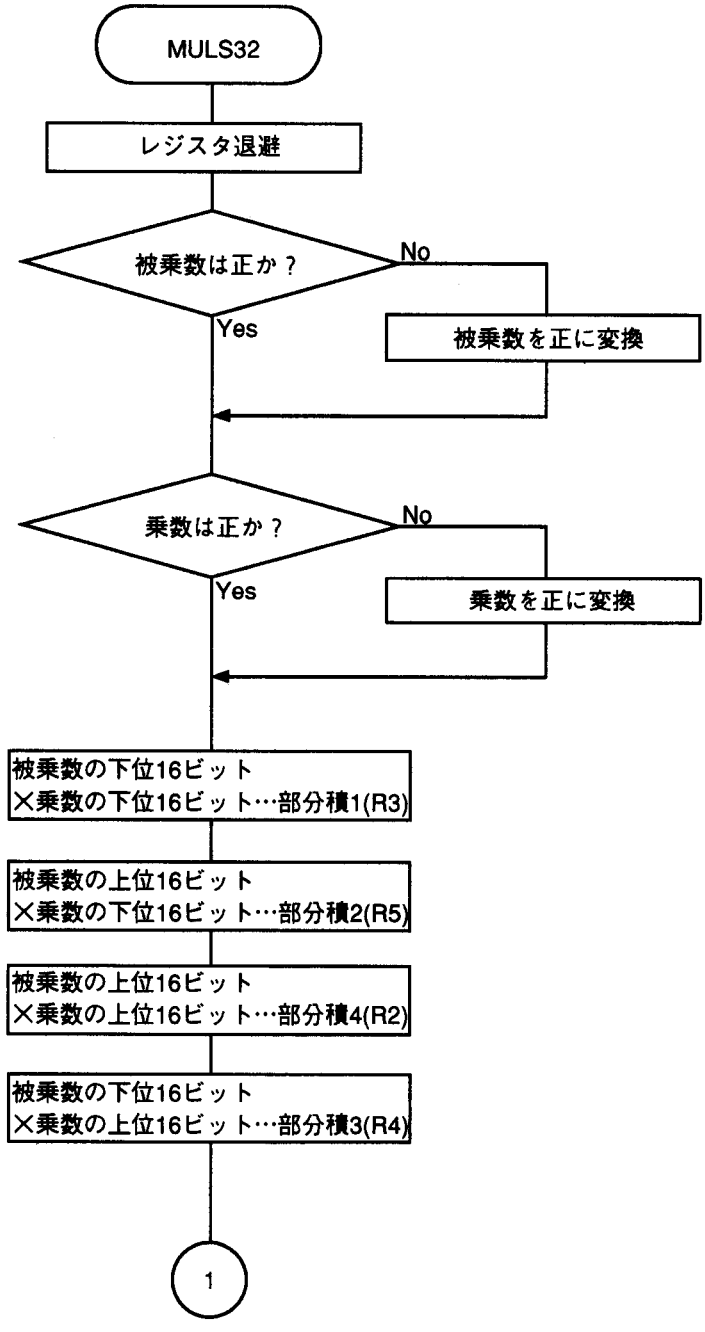
図3.24　乗算

表3.4　積の符号変換

被乗数のMSB	乗数のMSB	積の符号変換
正	正	正
正	負	負
負	正	負
負	負	正

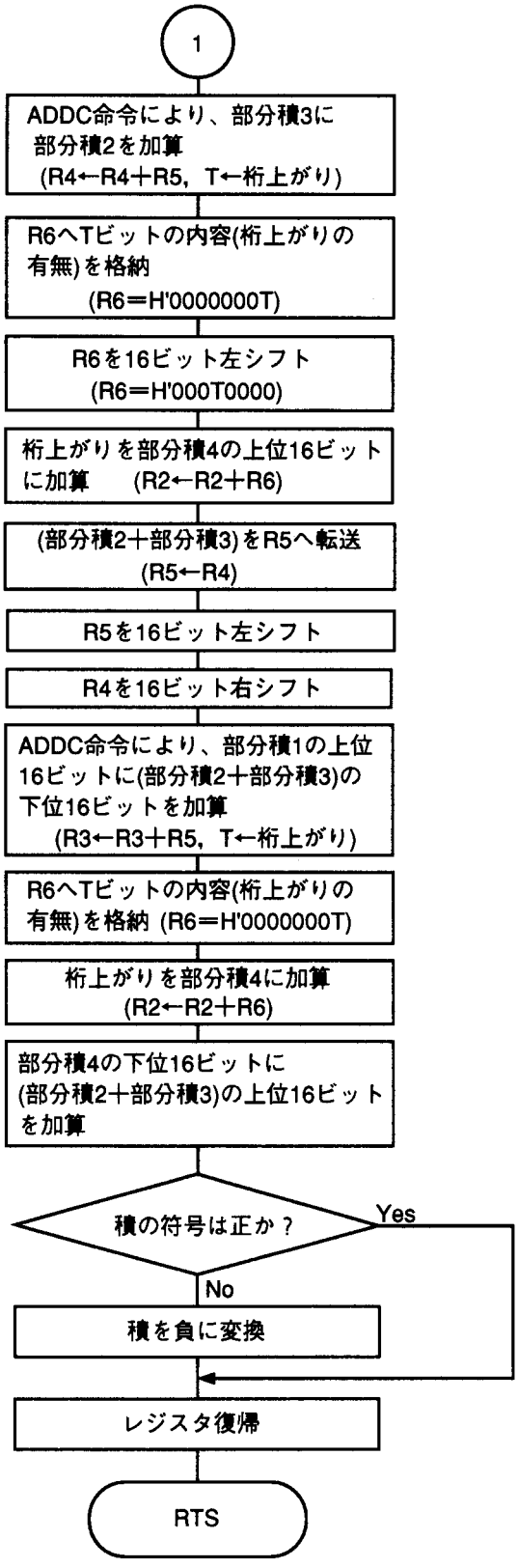
32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULS32	使用機能	MULU 命令 SWAP 命令 NEGC 命令

フローチャート



32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULS32	使用機能	MULU 命令 SWAP 命令 NEGC 命令

フローチャート



32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULS32	使用機能	MULU 命令 SWAP 命令 NEGC 命令

プログラムリスト

```
                                                ;*******************************************************************************
                                                ;*                                                                             *
                                                ;*          NAME   :  32 BIT SINGNED MULTIPLATION (MULS32)                     *
                                                ;*                                                                             *
                                                ;*******************************************************************************
                                                ;*                                                                             *
                                                ;*         ENTRY   :  R0    (MULTIPLICAND)                                    *
                                                ;*                    R1    (MULTIPLIER)                                      *
                                                ;*       RETURNS   :  R2    (UPPER 32 BIT PRODUCT)                            *
                                                ;*                    R3    (LOWER 32 BIT PRODUCT)                            *
                                                ;*                                                                             *
                                                ;*******************************************************************************
00001000                                        .SECTION A,CODE,LOCATE=H'1000
          00001000                  MULS32      .EQU    $                 ; Entry point
00001000 4F12                                   STS.L   MACL,@-R15        ; Escape register
00001002 2F46                                   MOV.L   R4,@-R15          ;
00001004 2F56                                   MOV.L   R5,@-R15          ;
00001006 2F66                                   MOV.L   R6,@-R15          ;
                                                ;
00001008 4011                                   CMP/PZ  R0                ; Multiplicand >= 0 ?
0000100A 8900                                   BT      MULS321           ; Yes
0000100C 600B                                   NEG     R0,R0             ; Change plus
0000100E                                MULS321 ;
0000100E 4111                                   CMP/PZ  R1                ; Multiplier >= 0 ?
00001010 8900                                   BT      MULS322           ; Yes
00001012 611B                                   NEG     R1,R1             ; Change puls
00001014                                MULS322 ;
00001014 201E                                   MULU    R1,R0             ; Lower 16 bit * lower 16 bit -> R3
00001016 6009                                   SWAP.W  R0,R0             ;
00001018 031A                                   STS     MACL,R3           ;
0000101A 201E                                   MULU    R1,R0             ; Upper 16 bit * lower 16 bit -> R5
0000101C 6119                                   SWAP.W  R1,R1             ;
0000101E 051A                                   STS     MACL,R5           ;
00001020 201E                                   MULU    R1,R0             ; Upper 16 bit * upper 16 bit -> R2
00001022 6009                                   SWAP.W  R0,R0             ;
00001024 021A                                   STS     MACL,R2           ;
00001026 201E                                   MULU    R1,R0             ; Lower 16 bit * upper 16 bit -> R4
00001028 6119                                   SWAP.W  R1,R1             ;
0000102A 041A                                   STS     MACL,R4           ;
                                                ;
0000102C 0008                                   CLRT                      ;
0000102E 345E                                   ADDC    R5,R4             ;
00001030 0629                                   MOVT    R6                ; R6 <- Carry
00001032 4628                                   SHLL16  R6                ;
00001034 326C                                   ADD     R6,R2             ; Carry = 1  R2 <- R2 + H'00010000
                                                ;                         ; Carry = 0  R2 <- R2 + H'00000000
00001036 6543                                   MOV     R4,R5             ;
00001038 4528                                   SHLL16  R5                ;
0000103A 4429                                   SHLR16  R4                ;
                                                ;
0000103C 0008                                   CLRT                      ;
0000103E 335E                                   ADDC    R5,R3             ;
00001040 0629                                   MOVT    R6                ; R6 <- Carry
00001042 326C                                   ADD     R6,R2             ; Carry = 1  R2 <- R2 + H'00000001
                                                ;                         ; Carry = 1  R2 <- R2 + H'00000000
00001044 324C                                   ADD     R4,R2             ;
                                                ;
```

32ﾋﾞｯﾄ×32ﾋﾞｯﾄ=64ﾋﾞｯﾄ (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	MULS32	使用機能	MULU 命令 SWAP 命令 NEGC 命令

プログラムリスト

```
  58 00001046 210A                58              XOR     R0,R1           ; Product < 0 ?
  59 00001048 4100                59              SHLL    R1              ;
  60                              60                              ;
  61 0000104A 8B02                61              BF      MULS32_END      ; No
  62 0000104C 0008                62              CLRT                    ; Change minus
  63 0000104E 633A                63              NEGC    R3,R3           ;
  64 00001050 622A                64              NEGC    R2,R2           ;
  65 00001052                     65      MULS32_END                      ;
  66 00001052 66F6                66              MOV.L   @R15+,R6        ; Return register
  67 00001054 65F6                67              MOV.L   @R15+,R5        ;
  68 00001056 64F6                68              MOV.L   @R15+,R4        ;
  69 00001058 000B                69              RTS                     ;
  70 0000105A 4F16                70              LDS.L   @R15+,MACL      ;
  71                              71              .END
*****TOTAL ERRORS        0
*****TOTAL WARNINGS      0
```

3．2．11　32ビット÷32ビットの商（符号なし）

32ビット÷32ビットの商 (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32Q	使用機能	DIV0U 命令 DIV1 命令

機　能

被除数(符号なし32ビット)と除数(符号なし32ビット)の除算を行い、商(符号なし32ビット)を求めます。また、エラー(0による除算)の有無をTビットに示します。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	被除数(符号なし32ビット)	R1	4
	除数(符号なし32ビット)	R0	4
出　力	商(符号なし32ビット)	R1	4
	エラー(0による除算)の有無(有:T=1，無:T=0)	Tビット(SR)	4

内部レジスタ変化およびフラグ変化

実行前　→　実行後

R0	除数(符号なし32ビット)　→　変化なし
R1	被除数(符号なし32ビット)　→　商(符号なし32ビット)
R2	ワーク
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

—：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
152
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
4
ｽﾃｰﾄ数
74
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

プログラミング仕様のステート数は、H'FFFFFFFF÷H'FFFFFFFEのときの値です。

32ビット÷32ビットの商 (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32Q	使用機能	DIV0U 命令 DIV1 命令

説　明

(1) 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数として、除数(符号なし32ビット)をセットします。

R1：入力引数として、被除数(符号なし32ビット)をセットします。

出力引数として、商(符号なし32ビット)がセットされます。

Tビット(SR)：エラー(0による除算)の有無を示します。

Tビット=1　実行した除算にエラー(0による除算)が発生したことを示します。

Tビット=0　実行した除算にエラー(0による除算)が発生しなかったことを示します。

(b) 図3.25にソフトウェアDIVU32Qの実行例を示します。

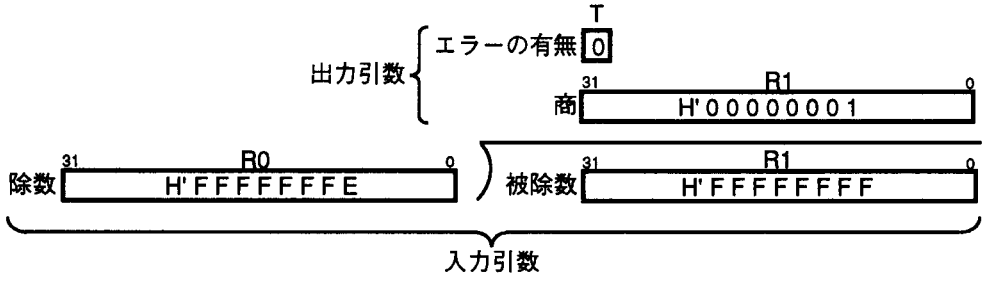


図3.25　ソフトウェアDIVU32Qの実行例

(2) 使用上の注意

ソフトウェアDIVU32Q実行後、被除数がセットされていた、R1には商がセットされるため、被除数は破壊されます。ソフトウェアDIVU32Q実行後も被除数を必要とする場合、被除数をあらかじめ退避してください。

(3) 使用RAM説明

ソフトウェアDIVU32QではRAMは使用していません。

(4) 使用例

被除数および除数 を入力引数にセットしてからソフトウェアDIVSU32Qをサブルーチンコールします。

```
                MOV. L  DATA1,R1      ……被除数(符号なし32ビット)を入力引数(R1)にセット
                BSR     DIVU32Q       ……ソフトウェアDIVU32Qをサブルーチンコール
                MOV. L  DATA2,R0      ……除数(符号なし32ビット)を入力引数(R0)にセット
                BT      ERROR         ……エラー(0による除算)が発生した場合、エラー処理
                                            ルーチンへ分岐
                        ⋮
                .align  4
DATA1           .data.l H'FFFFFFFF
DATA2           .data.l H'FFFFFFFE
```

32ビット÷32ビットの商 (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32Q	使用機能	DIV0U 命令 DIV1 命令

説　明

(5) 動作原理

(a) 除算前に以下の初期設定を行ないます。

(i) R2を上位32ビットとし、被除数を64ビットにゼロ拡張します。　(図3.26-①)

(ii) 1ステップ除算で使用するM、QおよびTビットを符号なし除算の値に設定します。
(図3.26-②)

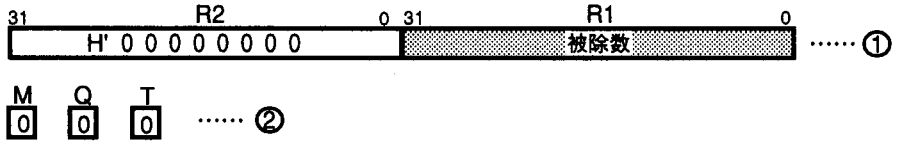


図3.26　初期設定

(b) 図3.27に示すように、除算はROTCL命令、DIV1命令を除数のビット数(32回)分繰り返し行います。

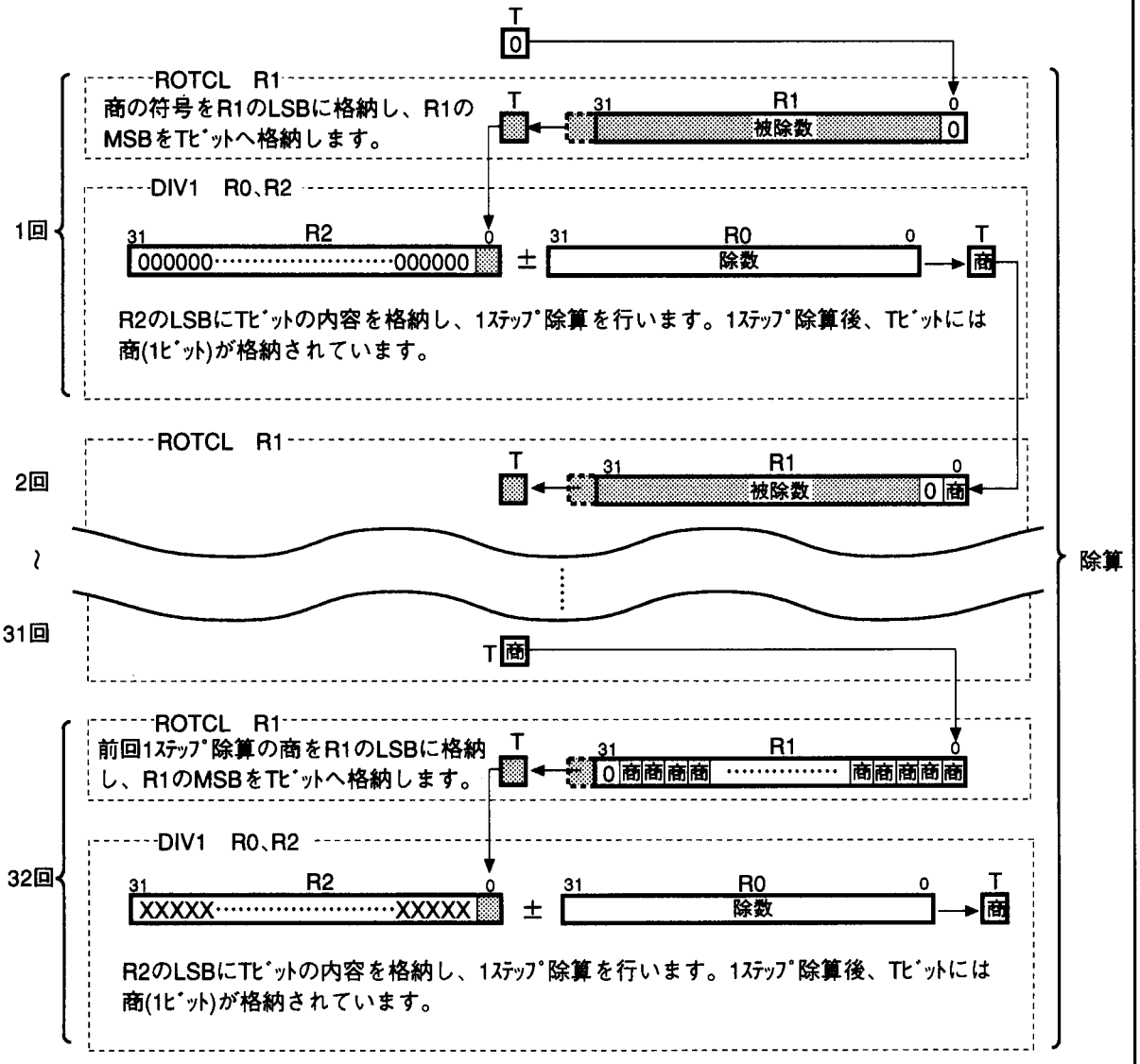


図3.27　除算

32ビット÷32ビットの商 (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32Q	使用機能	DIV0U 命令 DIV1 命令

説　明

(c)　図３．２８に示すように、除算終了時、Tビットには32回目の1ステップ除算の商が格納されています。Tビットの32回目の1ステップ除算の商をR1のLSBへ格納したR1の内容が商となります。

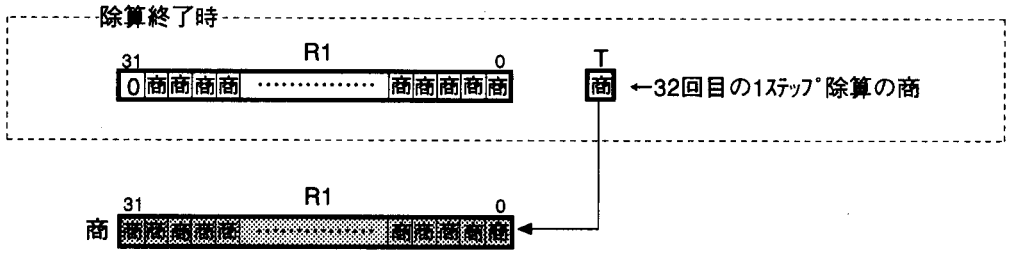
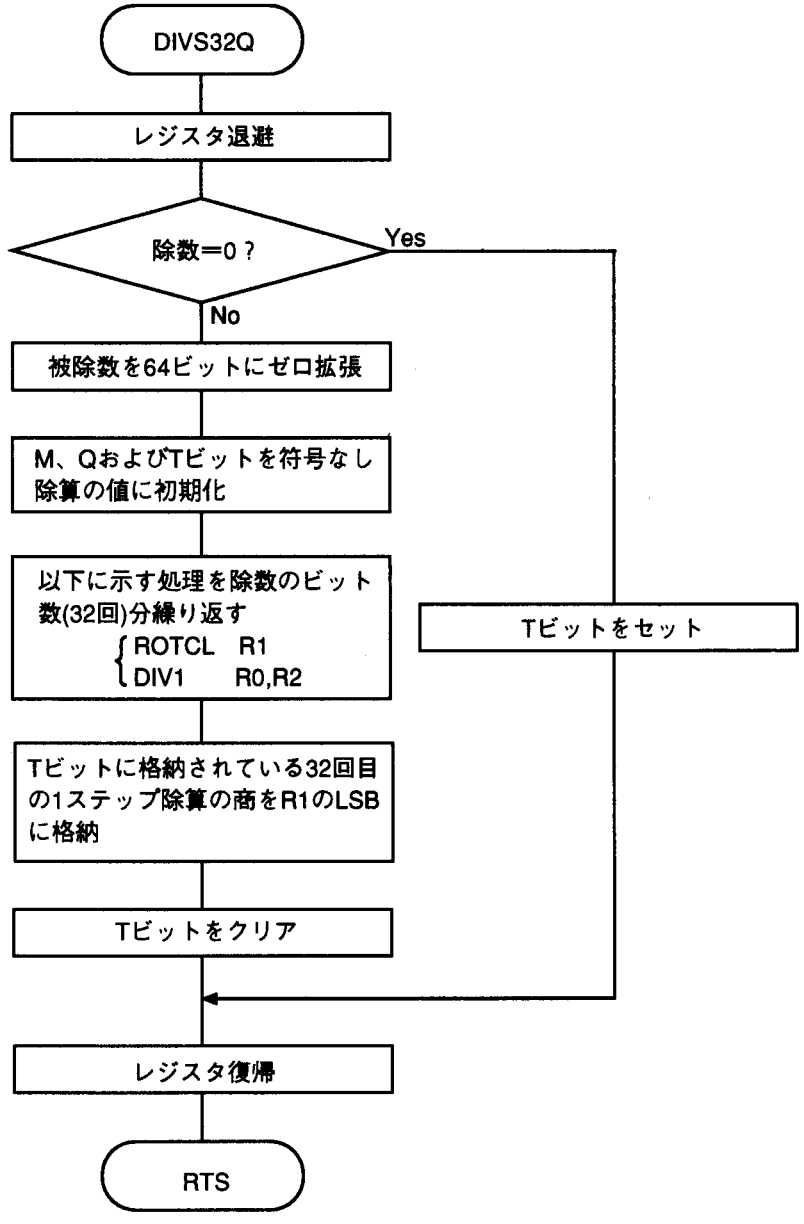


図３．２８　商

32ビット÷32ビットの商 (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32Q	使用機能	DIV0U 命令 DIV1 命令

フローチャート



32ビット÷32ビットの商 (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32Q	使用機能	DIV0U 命令 DIV1 命令

プログラムリスト

```
                                    1  ;*******************************************************************************
                                    2  ;*                                                                             *
                                    3  ;*          NAME  :  QUOTIENT OF 32 BIT UNSIGNED DIVISION (DIVU32Q)            *
                                    4  ;*                                                                             *
                                    5  ;*******************************************************************************
                                    6  ;*                                                                             *
                                    7  ;*          ENTRY  :  R1    (DIVIDEND)                                         *
                                    8  ;*                    R0    (DIVISOR)                                          *
                                    9  ;*        RETURNS  :  R1    (QUOTIENT)                                         *
                                   10  ;*                    T BIT (ERROR -> TRUE;T=1,FALSE;T=0)                      *
                                   11  ;*                                                                             *
                                   12  ;*******************************************************************************
00001000                           13          .SECTION A,CODE,LOCATE=H'1000
          00001000                 14  DIVU32Q .EQU    $                   ; Entry point
00001000 2F26                      15          MOV.L   R2,@-R15            ; Escape register
00001002 2008                      16          TST     R0,R0               ; Divisor = 0 ?
00001004 8945                      17          BT      DIVU32Q1            ; Yes
00001006 222A                      18          XOR     R2,R2               ; R2 <- H'00000000
00001008 0019                      19          DIV0U                       ; Divide as unsigned
                                   20                                      ;
0000100A 4124                      21          ROTCL   R1                  ; Divide 1 step
0000100C 3204                      22          DIV1    R0,R2               ;
0000100E 4124                      23          ROTCL   R1                  ;
00001010 3204                      24          DIV1    R0,R2               ;
00001012 4124                      25          ROTCL   R1                  ;
00001014 3204                      26          DIV1    R0,R2               ;
00001016 4124                      27          ROTCL   R1                  ;
00001018 3204                      28          DIV1    R0,R2               ;
0000101A 4124                      29          ROTCL   R1                  ;
0000101C 3204                      30          DIV1    R0,R2               ;
0000101E 4124                      31          ROTCL   R1                  ;
00001020 3204                      32          DIV1    R0,R2               ;
00001022 4124                      33          ROTCL   R1                  ;
00001024 3204                      34          DIV1    R0,R2               ;
00001026 4124                      35          ROTCL   R1                  ;
00001028 3204                      36          DIV1    R0,R2               ;
                                   37                                      ;
0000102A 4124                      38          ROTCL   R1                  ;
0000102C 3204                      39          DIV1    R0,R2               ;
0000102E 4124                      40          ROTCL   R1                  ;
00001030 3204                      41          DIV1    R0,R2               ;
00001032 4124                      42          ROTCL   R1                  ;
00001034 3204                      43          DIV1    R0,R2               ;
00001036 4124                      44          ROTCL   R1                  ;
00001038 3204                      45          DIV1    R0,R2               ;
0000103A 4124                      46          ROTCL   R1                  ;
0000103C 3204                      47          DIV1    R0,R2               ;
0000103E 4124                      48          ROTCL   R1                  ;
00001040 3204                      49          DIV1    R0,R2               ;
00001042 4124                      50          ROTCL   R1                  ;
00001044 3204                      51          DIV1    R0,R2               ;
00001046 4124                      52          ROTCL   R1                  ;
00001048 3204                      53          DIV1    R0,R2               ;
                                   54                                      ;
0000104A 4124                      55          ROTCL   R1                  ;
0000104C 3204                      56          DIV1    R0,R2               ;
0000104E 4124                      57          ROTCL   R1                  ;
```

32ビット÷32ビットの商 (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32Q	使用機能	DIV0U 命令 DIV1 命令

プログラムリスト

```
  58 00001050 3204                58              DIV1    R0,R2           ;
  59 00001052 4124                59              ROTCL   R1              ;
  60 00001054 3204                60              DIV1    R0,R2           ;
  61 00001056 4124                61              ROTCL   R1              ;
  62 00001058 3204                62              DIV1    R0,R2           ;
  63 0000105A 4124                63              ROTCL   R1              ;
  64 0000105C 3204                64              DIV1    R0,R2           ;
  65 0000105E 4124                65              ROTCL   R1              ;
  66 00001060 3204                66              DIV1    R0,R2           ;
  67 00001062 4124                67              ROTCL   R1              ;
  68 00001064 3204                68              DIV1    R0,R2           ;
  69 00001066 4124                69              ROTCL   R1              ;
  70 00001068 3204                70              DIV1    R0,R2           ;
  71                              71                                      ;
  72 0000106A 4124                72              ROTCL   R1              ;
  73 0000106C 3204                73              DIV1    R0,R2           ;
  74 0000106E 4124                74              ROTCL   R1              ;
  75 00001070 3204                75              DIV1    R0,R2           ;
  76 00001072 4124                76              ROTCL   R1              ;
  77 00001074 3204                77              DIV1    R0,R2           ;
  78 00001076 4124                78              ROTCL   R1              ;
  79 00001078 3204                79              DIV1    R0,R2           ;
  80 0000107A 4124                80              ROTCL   R1              ;
  81 0000107C 3204                81              DIV1    R0,R2           ;
  82 0000107E 4124                82              ROTCL   R1              ;
  83 00001080 3204                83              DIV1    R0,R2           ;
  84 00001082 4124                84              ROTCL   R1              ;
  85 00001084 3204                85              DIV1    R0,R2           ;
  86 00001086 4124                86              ROTCL   R1              ;
  87 00001088 3204                87              DIV1    R0,R2           ;
  88                              88                                      ;
  89 0000108A 4124                89              ROTCL   R1              ;
  90 0000108C 0008                90              CLRT                    ; T bit <- No error
  91 0000108E 000B                91              RTS                     ;
  92 00001090 62F6                92              MOV.L   @R15+,R2        ; Return register
  93 00001092                     93      DIVU32Q1                        ;
  94 00001092 0018                94              SETT                    ; T bit <- Error
  95 00001094 000B                95              RTS                     ;
  96 00001096 62F6                96              MOV.L   @R15+,R2        ; Return register
  97                              97              .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．12　32ビット÷32ビット剰余（符号なし）

32ビット÷32ビットの剰余(符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32R	使用機能	DIV0U 命令 DIV1 命令

機　能

被除数(符号なし32ビット)と除数(符号なし32ビット)の除算を行ない、剰余(符号なし32ビット)を求めます。また、エラー(0による除算)の有無をTビットに示します。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	被除数(符号なし32ビット)	R1	4
	除数(符号なし32ビット)	R0	4
出　力	剰余(符号なし32ビット)	R2	4
	エラー(0による除算)の有無(有:T=1，無:T=0)	Tビット(SR)	4

内部レジスタ変化およびフラグ変化

	実行前 → 実行後
R0	除数(符号なし32ﾋﾞｯﾄ) → 変化なし
R1	被除数(符号なし32ﾋﾞｯﾄ) → 変化
R2	不定 → 剰余(符号なし32ﾋﾞｯﾄ)
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

—：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
148
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
0
ｽﾃｰﾄ数
74
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

プログラミング仕様のステート数は、H'FFFFFFFF÷H'FFFFFFFEのときの値です。

32ビット÷32ビットの剰余(符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32R	使用機能	DIV0U 命令 DIV1 命令

説　明

(1) 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数として、除数(符号なし32ビット)をセットします。

R1：入力引数として、被除数(符号なし32ビット)をセットします。

R2：出力引数として、剰余(符号なし32ビット)がセットされます。

Tビット(SR)：エラー(0による除算)の有無を示します。

Tビット=1　実行した除算にエラー(0による除算)が発生したことを示します。

Tビット=0　実行した除算にエラー(0による除算)が発生しなかったことを示します。

(b) 図3.29にソフトウェアDIVU32Rの実行例を示します。

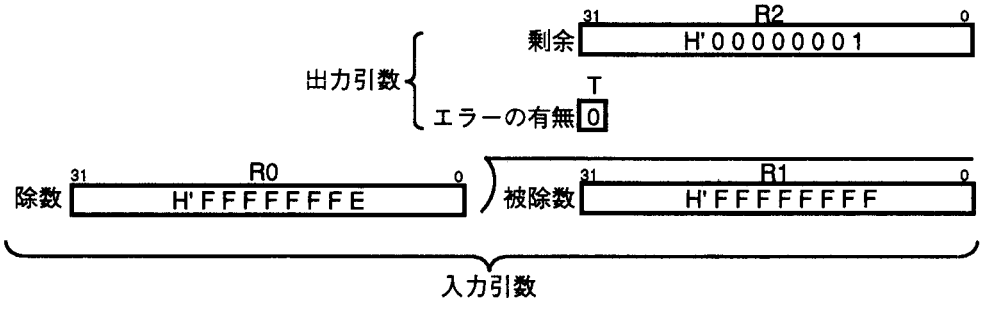


図3.29　ソフトウェアDIVU32Rの実行例

(2) 使用上の注意

被除数がセットされていたR1は、ソフトウェアDIVU32Rの実行により内容が変化します。ソフトウェアDIVU32R実行後も被除数を必要とする場合、被除数をあらかじめ退避してください。

(3) 使用RAM説明

ソフトウェアDIVU32RではRAMは使用していません。

(4) 使用例

被除数および除数 を入力引数にセットしてからソフトウェアDIVU32Rをサブルーチンコールします。

```
        MOV. L  DATA1,R1     ……被除数(符号なし32ビット)を入力引数(R1)にセット
        BSR     DIVU32R      ……ソフトウェアDIVU32Rをサブルーチンコール
        MOV. L  DATA2,R0     ……除数(符号なし32ビット)を入力引数(R0)にセット
        BT      ERROR        ……エラー(0による除算)が発生した場合、エラー処理
                                 ルーチンへ分岐
                :
        .align  4
DATA1   .data.l H'FFFFFFFF
DATA2   .data.l H'FFFFFFFE
```

32ビット÷32ビットの剰余(符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32R	使用機能	DIV0U 命令 DIV1 命令

説　明

（5） 動作原理

(a) 除算前に以下の初期設定を行ないます。

(i) R2を上位32ビットとし、被除数を64ビットにゼロ拡張します。 (図3.30-①)

(ii) 1ステップ除算で使用するM、QおよびTビットを符号なし除算の値に設定します。 (図3.30-②)

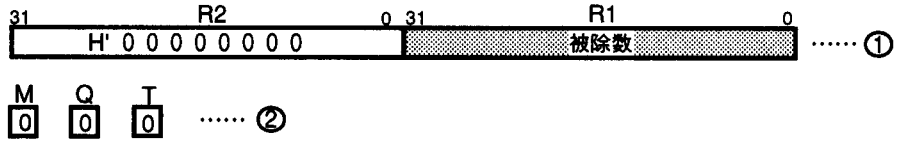


図3.30　初期設定

(b) 図3.31に示すように、除算はROTCL命令、DIV1命令を除数のビット数(32回)分繰り返し行います。

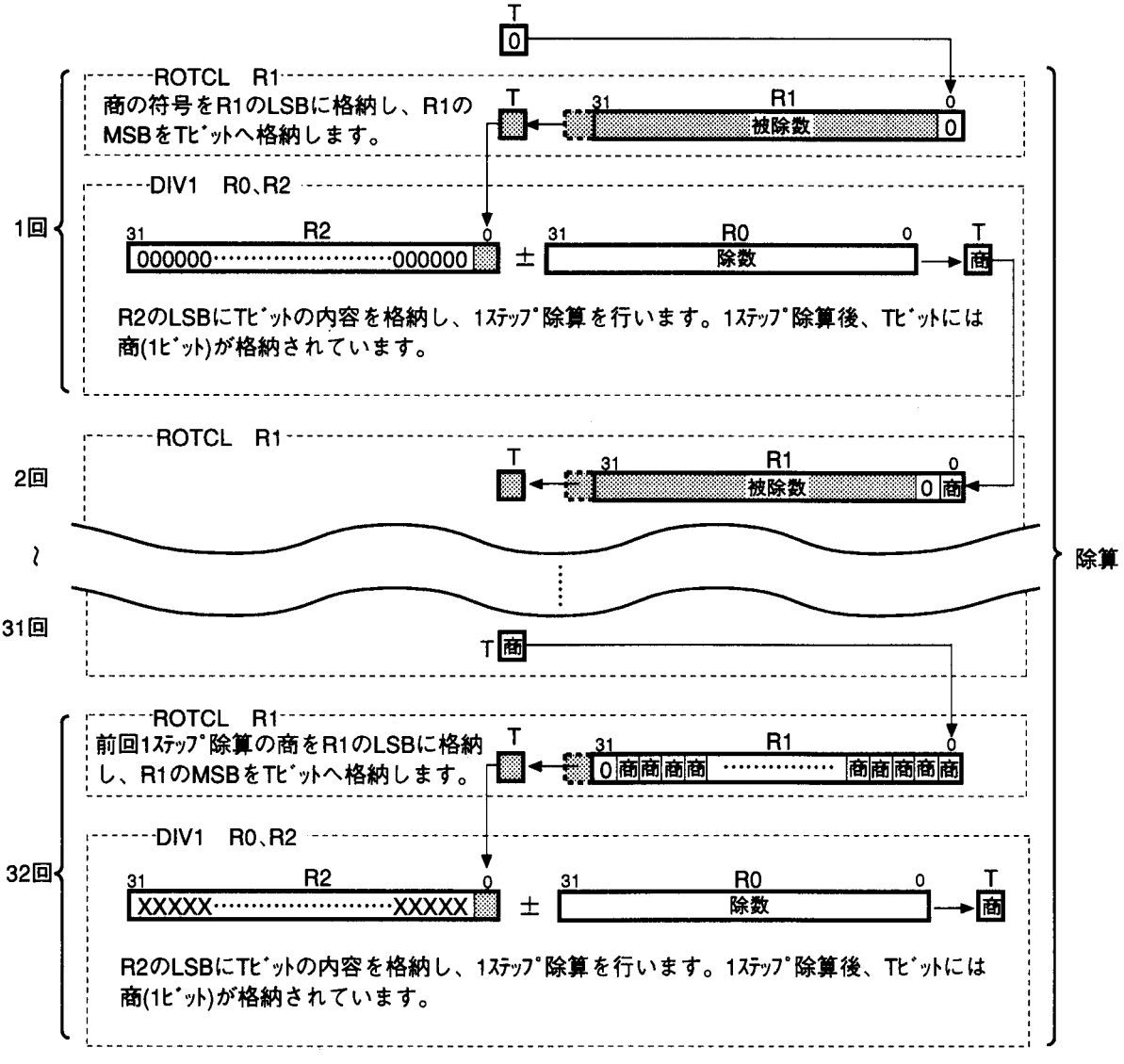


図3.31　除算

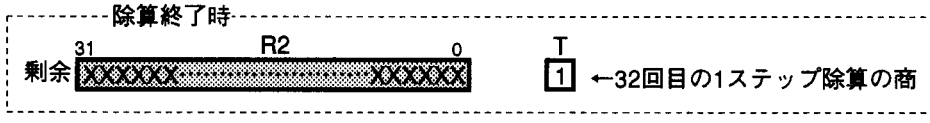
32ビット÷32ビットの剰余(符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32R	使用機能	DIV0U 命令 DIV1 命令

説　明

(c)　図３.３２に示すように、除算終了時のTビット(32回目の1ｽﾃｯﾌﾟ除算の商)の内容により剰余の求め方が異なります。

・Tビット(32回目の1ｽﾃｯﾌﾟ除算の商)＝1のとき

除算終了時のR2の内容が剰余となります。



・Tビット(32回目の1ｽﾃｯﾌﾟ除算の商)＝0のとき

R2は32回目の1ステップ除算の内部処理により除数を1回減算し過ぎた値になっているため、除算終了時のR2に除数を加算した値が剰余となります。

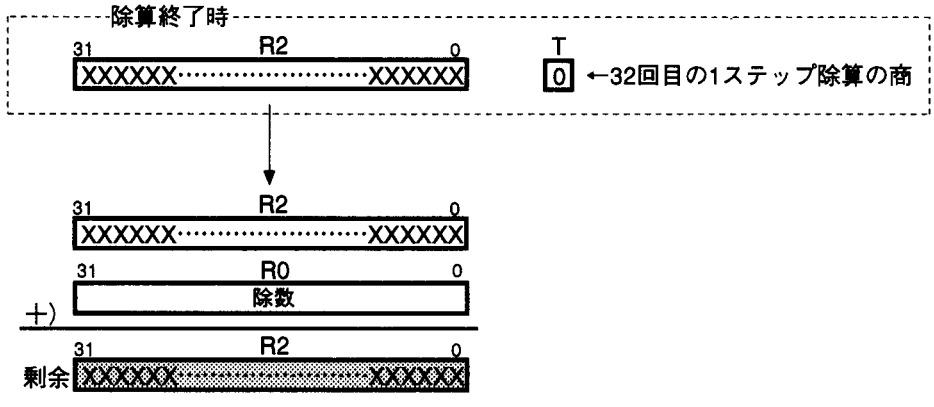
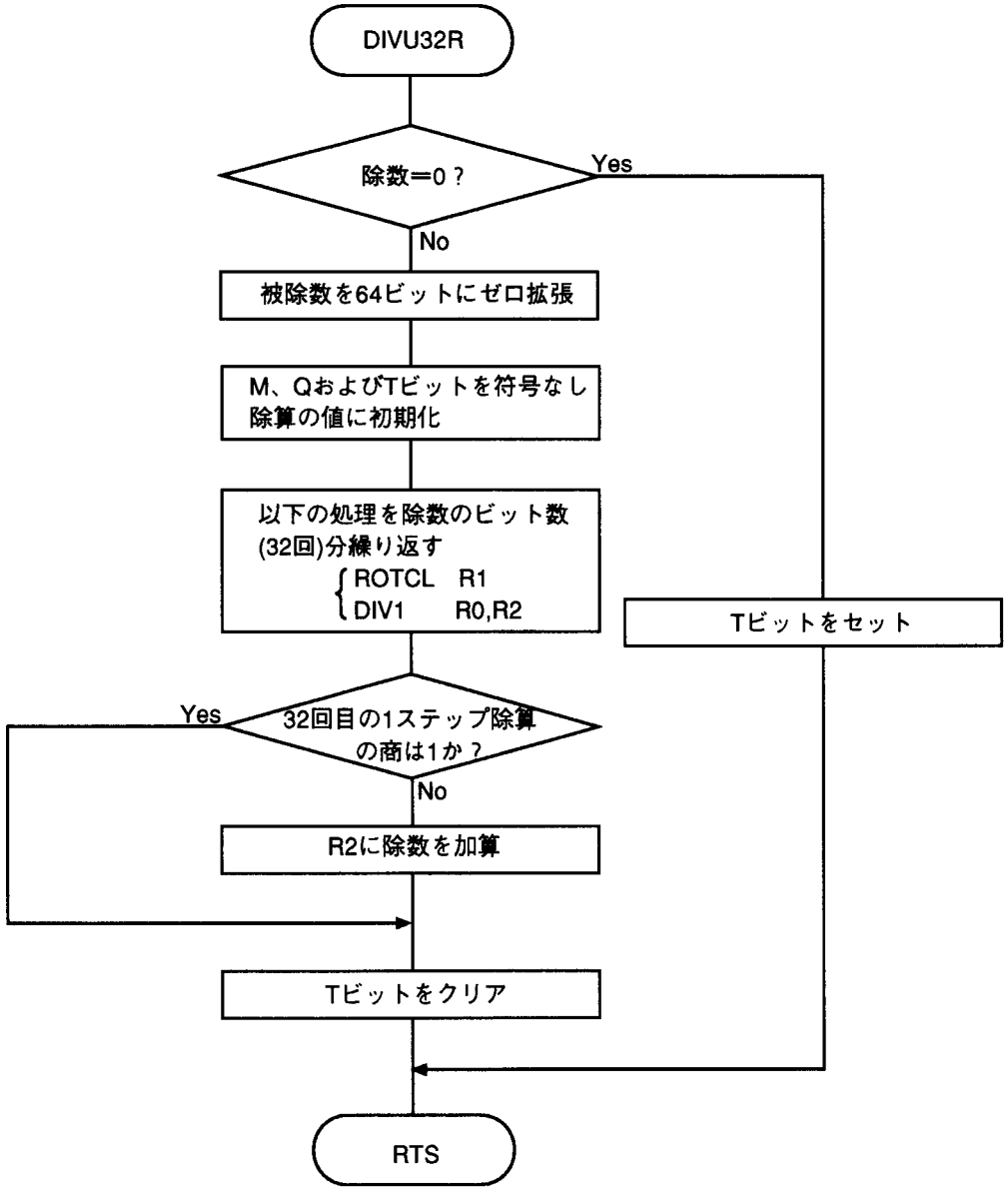


図３.３２　剰余

32ビット÷32ビットの剰余 (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32R	使用機能	DIV0U 命令 DIV1 命令

フローチャート



32ビット÷32ビットの剰余(符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32R	使用機能	DIV0U 命令 DIV1 命令

プログラムリスト

```
                                    1  ;*******************************************************************************
                                    2  ;*                                                                             *
                                    3  ;*          NAME  :  RESIDUAL OF 32 BIT UNSIGNED DIVISION (DIVU32R)            *
                                    4  ;*                                                                             *
                                    5  ;*******************************************************************************
                                    6  ;*                                                                             *
                                    7  ;*          ENTRY  :  R1    (DIVIDEND)                                         *
                                    8  ;*                    R0    (DIVISOR)                                          *
                                    9  ;*        RETURNS  :  R2    (RESIDUAL)                                         *
                                   10  ;*                    T BIT (ERROR -> TRUE;T=1,FALSE;T=0)                      *
                                   11  ;*                                                                             *
                                   12  ;*******************************************************************************
00001000                           13          .SECTION A,CODE,LOCATE=H'1000
          00001000                 14  DIVU32R .EQU    $               ; Entry point
00001000 2008                      15          TST     R0,R0           ; Divisor = 0 ?
00001002 8945                      16          BT      DIVU32R2        ; Yes
00001004 222A                      17          XOR     R2,R2           ; R2 <- H'00000000
00001006 0019                      18          DIV0U                   ; Divide as unsigned
                                   19                                  ;
00001008 4124                      20          ROTCL   R1              ; Divide 1 step
0000100A 3204                      21          DIV1    R0,R2           ;
0000100C 4124                      22          ROTCL   R1              ;
0000100E 3204                      23          DIV1    R0,R2           ;
00001010 4124                      24          ROTCL   R1              ;
00001012 3204                      25          DIV1    R0,R2           ;
00001014 4124                      26          ROTCL   R1              ;
00001016 3204                      27          DIV1    R0,R2           ;
00001018 4124                      28          ROTCL   R1              ;
0000101A 3204                      29          DIV1    R0,R2           ;
0000101C 4124                      30          ROTCL   R1              ;
0000101E 3204                      31          DIV1    R0,R2           ;
00001020 4124                      32          ROTCL   R1              ;
00001022 3204                      33          DIV1    R0,R2           ;
00001024 4124                      34          ROTCL   R1              ;
00001026 3204                      35          DIV1    R0,R2           ;
                                   36                                  ;
00001028 4124                      37          ROTCL   R1              ;
0000102A 3204                      38          DIV1    R0,R2           ;
0000102C 4124                      39          ROTCL   R1              ;
0000102E 3204                      40          DIV1    R0,R2           ;
00001030 4124                      41          ROTCL   R1              ;
00001032 3204                      42          DIV1    R0,R2           ;
00001034 4124                      43          ROTCL   R1              ;
00001036 3204                      44          DIV1    R0,R2           ;
00001038 4124                      45          ROTCL   R1              ;
0000103A 3204                      46          DIV1    R0,R2           ;
0000103C 4124                      47          ROTCL   R1              ;
0000103E 3204                      48          DIV1    R0,R2           ;
00001040 4124                      49          ROTCL   R1              ;
00001042 3204                      50          DIV1    R0,R2           ;
00001044 4124                      51          ROTCL   R1              ;
00001046 3204                      52          DIV1    R0,R2           ;
                                   53                                  ;
00001048 4124                      54          ROTCL   R1              ;
0000104A 3204                      55          DIV1    R0,R2           ;
0000104C 4124                      56          ROTCL   R1              ;
0000104E 3204                      57          DIV1    R0,R2           ;
```

32ビット÷32ビットの剰余 (符号なし)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32R	使用機能	DIV0U 命令 DIV1 命令

プログラムリスト

```
  58 00001050 4124                58            ROTCL   R1              ;
  59 00001052 3204                59            DIV1    R0,R2           ;
  60 00001054 4124                60            ROTCL   R1              ;
  61 00001056 3204                61            DIV1    R0,R2           ;
  62 00001058 4124                62            ROTCL   R1              ;
  63 0000105A 3204                63            DIV1    R0,R2           ;
  64 0000105C 4124                64            ROTCL   R1              ;
  65 0000105E 3204                65            DIV1    R0,R2           ;
  66 00001060 4124                66            ROTCL   R1              ;
  67 00001062 3204                67            DIV1    R0,R2           ;
  68 00001064 4124                68            ROTCL   R1              ;
  69 00001066 3204                69            DIV1    R0,R2           ;
  70                              70                                    ;
  71 00001068 4124                71            ROTCL   R1              ;
  72 0000106A 3204                72            DIV1    R0,R2           ;
  73 0000106C 4124                73            ROTCL   R1              ;
  74 0000106E 3204                74            DIV1    R0,R2           ;
  75 00001070 4124                75            ROTCL   R1              ;
  76 00001072 3204                76            DIV1    R0,R2           ;
  77 00001074 4124                77            ROTCL   R1              ;
  78 00001076 3204                78            DIV1    R0,R2           ;
  79 00001078 4124                79            ROTCL   R1              ;
  80 0000107A 3204                80            DIV1    R0,R2           ;
  81 0000107C 4124                81            ROTCL   R1              ;
  82 0000107E 3204                82            DIV1    R0,R2           ;
  83 00001080 4124                83            ROTCL   R1              ;
  84 00001082 3204                84            DIV1    R0,R2           ;
  85 00001084 4124                85            ROTCL   R1              ;
  86 00001086 3204                86            DIV1    R0,R2           ;
  87                              87                                    ;
  88 00001088 8900                88            BT      DIVU32R1        ; T bit = 1 ?
  89 0000108A 320C                89            ADD     R0,R2           ; Clear oversub
  90 0000108C                     90    DIVU32R1                        ;
  91 0000108C 000B                91            RTS                     ;
  92 0000108E 0008                92            CLRT                    ; T bit <- No error
  93 00001090                     93    DIVU32R2                        ;
  94 00001090 000B                94            RTS                     ;
  95 00001092 0018                95            SETT                    ; T bit <- Error
  96                              96            .END
*****TOTAL ERRORS        0
*****TOTAL WARNINGS      0
```

3．2．13　32ビット÷32ビットの商（符号付き）

32ビット÷32ビットの商 (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32Q	使用機能	DIV0S 命令 DIV1 命令

機　能

被除数(符号付き32ビット)と除数(符号付き32ビット)の除算を行い、商(符号付き32ビット)を求めます。
また、エラー(0による除算)の有無をTビットに示します。

引　数

内　　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	被除数(符号付き32ビット)	R1	4
	除数(符号付き32ビット)	R0	4
出　力	商(符号付き32ビット)	R1	4
	エラー(0による除算)の有無(有:T=1，無:T=0)	Tビット(SR)	4

内部レジスタ変化およびフラグ変化

実行前　→　実行後

R0	除数(符号付き32ビット)　→　変化なし
R1	被除数(符号付き32ビット)　→　商(符号付き32ビット)
R2	ワーク
R3	ワーク
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

—：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
166
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
8
ｽﾃｰﾄ数
80
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

プログラミング仕様のステート数は、H'80000000÷H'7FFFFFFFのときの値です。

32ビット÷32ビットの商 (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32Q	使用機能	DIV0S 命令 DIV1 命令

説　明

(1) 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数として、除数(符号付き32ビット)をセットします。

R1：入力引数として、被除数(符号付き32ビット)をセットします。

出力引数として、商(符号付き32ビット)がセットされます。

Tビット(SR)：エラー(0による除算)の有無を示します。

Tビット=1　実行した除算にエラー(0による除算)が発生したことを示します。

Tビット=0　実行した除算にエラー(0による除算)が発生しなかったことを示します。

(b) 図3.3 3にソフトウェアDIVS32Qの実行例を示します。

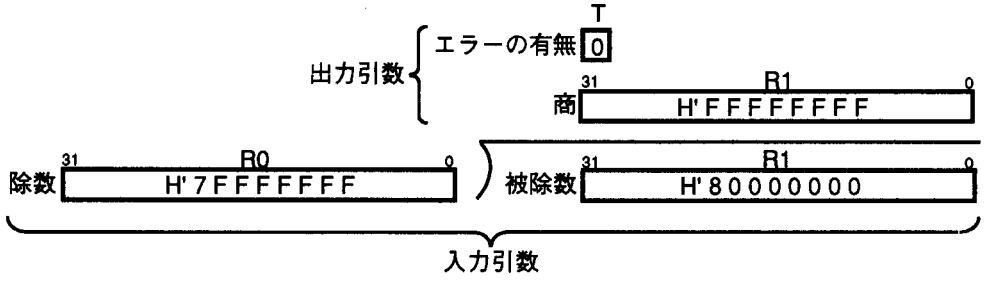


図3.3 3　ソフトウェアDIVS32Qの実行例

(2) 使用上の注意

ソフトウェアDIVS32Q実行後、被除数がセットされていたR1には商がセットされるため、被除数は破壊されます。ソフトウェアDIVS32Q実行後も被除数を必要とする場合、被除数をあらかじめ退避してください。

また、H'80000000÷H'FFFFFFFFのときはオーバフローとなりますが、ソフトウェアDIVS32Qでは、このオーバフローの検出を行っていません。

(3) 使用RAM説明

ソフトウェアDIVS32QではRAMは使用していません。

(4) 使用例

被除数および除数 をセットしてからソフトウェアDIVS32Qをサブルーチンコールします。

```
        MOV. L  DATA1,R1    ……被除数(符号付き32ビット)を入力引数(R1)にセット
        BSR     DIVS32Q     ……ソフトウェアDIVS32Qをサブルーチンコール
        MOV. L  DATA2,R0    ……除数(符号付き32ビット)を入力引数(R0)にセット
        BT      ERROR       ……エラー(0による除算)が発生した場合、エラー処理
                                ルーチンへ分岐
                :
        .align  4
DATA1   .data.l H'80000000
DATA2   .data.l H'7FFFFFFF
```

32ビット÷32ビットの商 (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32Q	使用機能	DIV0S 命令 DIV1 命令

説　明

(5) 動作原理

(a) 除算前に以下の初期設定を行ないます。

(i) R2を上位32ビットとし、被除数を64ビットに符号拡張します。(図3.3 4-①)

(ii) 被除数が負の場合、1ステップ除算命令で扱えるように1の補数に変換します。(図3.3 4-②)

(iii) 1ステップ除算で使用するM、QおよびTビットを符号付き除算の値(M＝除数の符号、Q＝被除数の符号、T＝商の符号)に設定します。(図3.3 4-③)

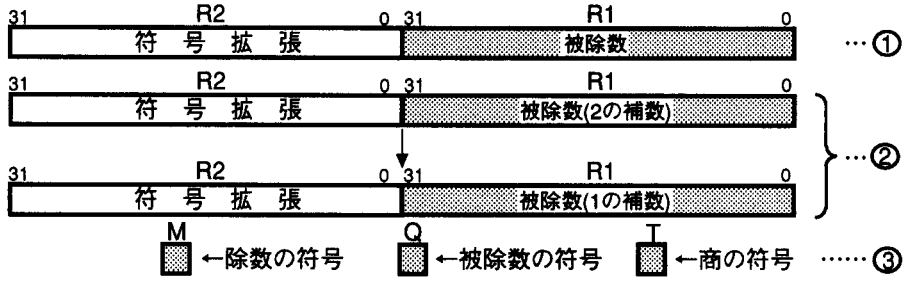


図3.3 4　初期設定

(b) 図3.3 5に示すように、除算はROTCL命令、DIV1を除数のビット数(32回)分繰り返して行います。

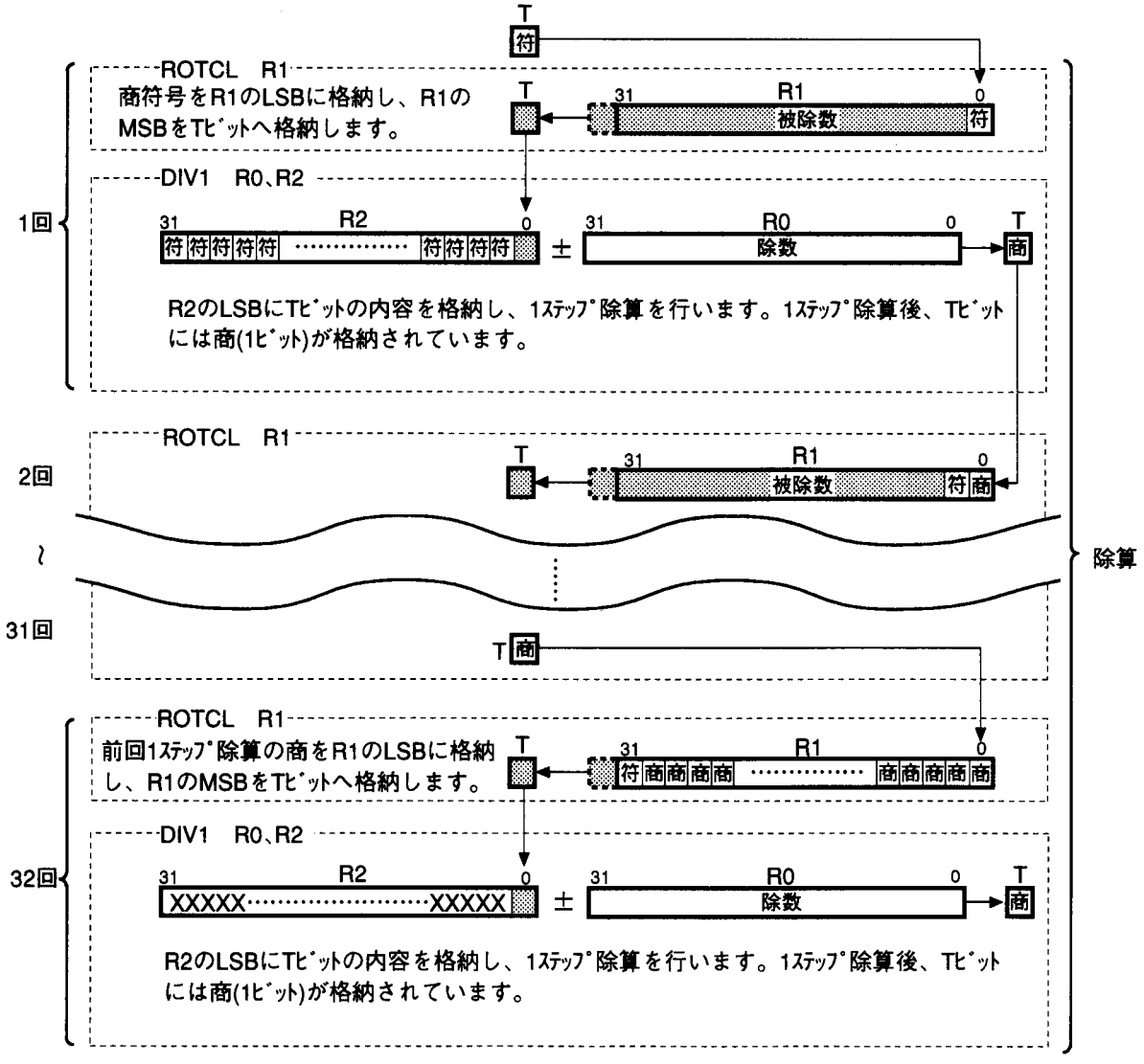


図3.3 5　動作例

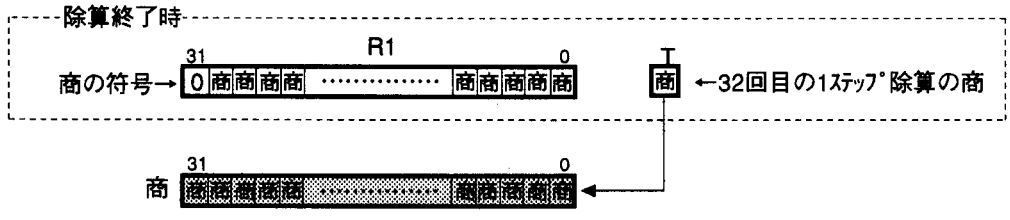
32ビット÷32ビットの商 (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32Q	使用機能	DIV0S 命令 DIV1 命令

説　明

(c)

(i) 図3.36に示すように、除算終了時、Tビットには32回目の1ステップ除算の商、R1のMSBには商の符号が格納されています。商が正のときは、Tビット(32回目の1ステップ除算の商)をR1のLSBへ格納したR1の内容が商となりす。商が負のときは、Tビット(32回目の1ステップ除算の商)をR1のLSBへ格納したR1の内容は1の補数表現の商となっているので、2の補数表現に変換します。

・商が正(除算終了時のR1のMSB=0)のとき



・商が負(除算終了時のR1のMSB=1)のとき

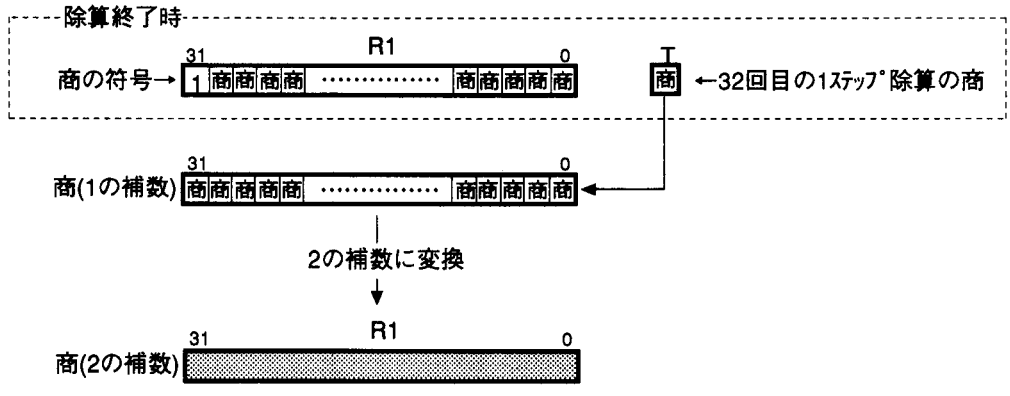


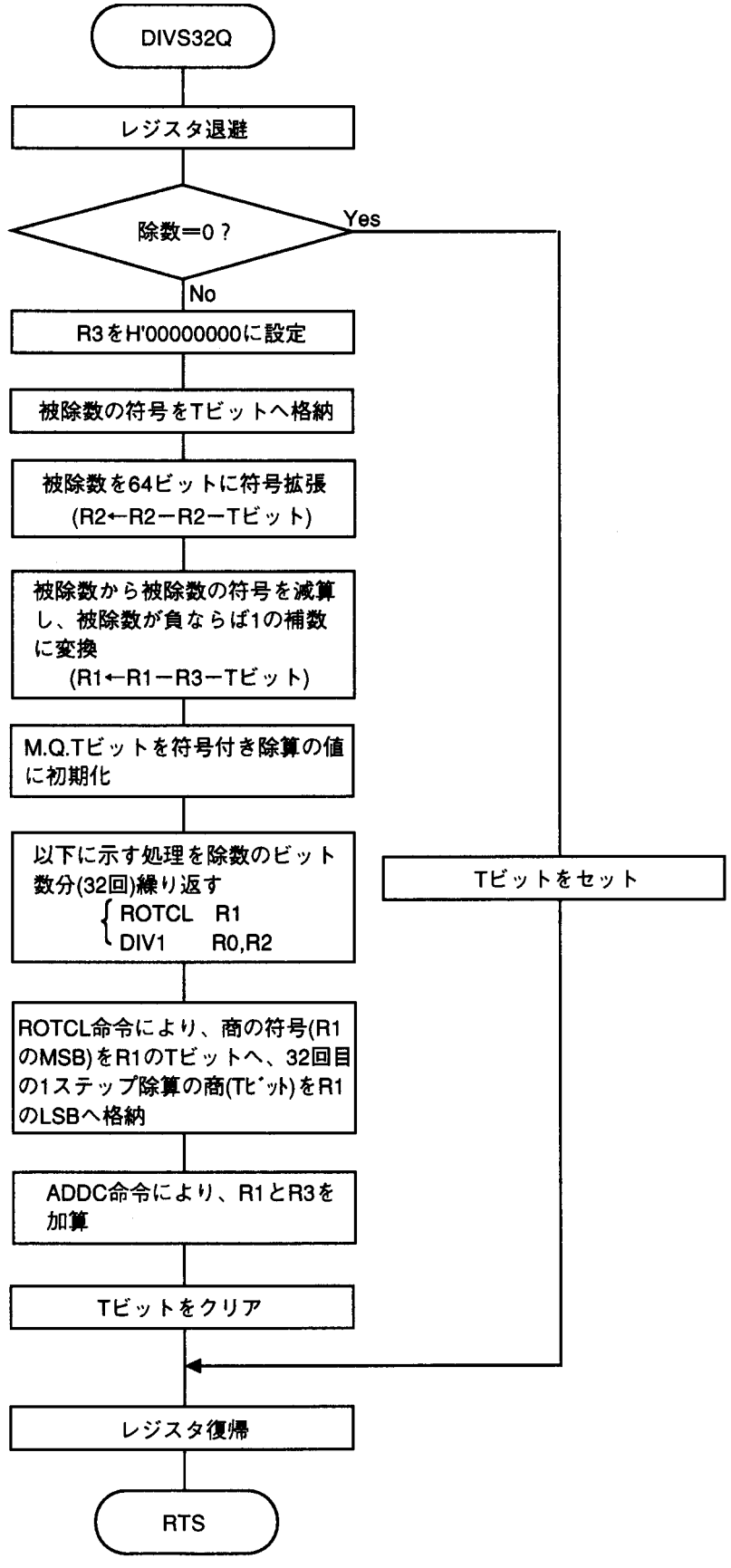
図3.36　商

(ii) ソフトウェアDIVS32Qでは、(i)の処理を以下のようにして行なっています。但し、R3にはH'00000000が格納されています。

```
ROTCL   R1          ←商の符号をTビットへ格納し、Tビットの商をR1の
                      LSBへ格納します。
ADDC    R3, R1      ←商が正ならば、Tビット=0なので商の値に変化はなく、
                      商が負ならば、Tビット=1なので1が加算されて2の補数
                      となります。
```

32ビット÷32ビットの商 (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32Q	使用機能	DIV0S 命令 DIV1 命令

フローチャート



32ビット÷32ビットの商 (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32Q	使用機能	DIV0S 命令 DIV1 命令

プログラムリスト

```
                                    ;*******************************************************************************
                                    ;*                                                                             *
                                    ;*          NAME  :  QUOTIENT OF 32 BIT SIGNED DIVISION (DIVS32Q)              *
                                    ;*                                                                             *
                                    ;*******************************************************************************
                                    ;*                                                                             *
                                    ;*         ENTRY  :  R1    (DIVIDEND)                                          *
                                    ;*                   R0    (DIVISOR)                                           *
                                    ;*       RETURNS  :  R1    (QUOTIENT)                                          *
                                    ;*                   T BIT (ERROR -> TRUE;T=1,FALSE;T=0)                       *
                                    ;*                                                                             *
                                    ;*******************************************************************************
00001000                            .SECTION A,CODE,LOCATE=H'1000
          00001000                  DIVS32Q .EQU    $               ; Entry point
00001000 2F26                       MOV.L   R2,@-R15        ; Escape register
00001002 2F36                       MOV.L   R3,@-R15        ;
00001004 2008                       TST     R0,R0           ; Divisor= 0 ?
00001006 894A                       BT      DIVS32Q1        ; Yes
00001008 233A                       XOR     R3,R3           ; R3 <- H'00000000
0000100A 2137                       DIV0S   R3,R1           ; T bit <- Sign of dividend
0000100C 322A                       SUBC    R2,R2           ; R2 sign extend
0000100E 313A                       SUBC    R3,R1           ;
00001010 2207                       DIV0S   R0,R2           ; Divide as signed
                                    ;
00001012 4124                       ROTCL   R1              ; Divide 1 step
00001014 3204                       DIV1    R0,R2           ;
00001016 4124                       ROTCL   R1              ;
00001018 3204                       DIV1    R0,R2           ;
0000101A 4124                       ROTCL   R1              ;
0000101C 3204                       DIV1    R0,R2           ;
0000101E 4124                       ROTCL   R1              ;
00001020 3204                       DIV1    R0,R2           ;
00001022 4124                       ROTCL   R1              ;
00001024 3204                       DIV1    R0,R2           ;
00001026 4124                       ROTCL   R1              ;
00001028 3204                       DIV1    R0,R2           ;
0000102A 4124                       ROTCL   R1              ;
0000102C 3204                       DIV1    R0,R2           ;
0000102E 4124                       ROTCL   R1              ;
00001030 3204                       DIV1    R0,R2           ;
                                    ;
00001032 4124                       ROTCL   R1              ;
00001034 3204                       DIV1    R0,R2           ;
00001036 4124                       ROTCL   R1              ;
00001038 3204                       DIV1    R0,R2           ;
0000103A 4124                       ROTCL   R1              ;
0000103C 3204                       DIV1    R0,R2           ;
0000103E 4124                       ROTCL   R1              ;
00001040 3204                       DIV1    R0,R2           ;
00001042 4124                       ROTCL   R1              ;
00001044 3204                       DIV1    R0,R2           ;
00001046 4124                       ROTCL   R1              ;
00001048 3204                       DIV1    R0,R2           ;
0000104A 4124                       ROTCL   R1              ;
0000104C 3204                       DIV1    R0,R2           ;
0000104E 4124                       ROTCL   R1              ;
00001050 3204                       DIV1    R0,R2           ;
```

32ビット÷32ビットの商 (符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVU32Q	使用機能	DIV0S 命令 DIV1 命令

プログラムリスト

```
  58                                    58
  59 00001052 4124                      59            ROTCL   R1              ;
  60 00001054 3204                      60            DIV1    R0,R2           ;
  61 00001056 4124                      61            ROTCL   R1              ;
  62 00001058 3204                      62            DIV1    R0,R2           ;
  63 0000105A 4124                      63            ROTCL   R1              ;
  64 0000105C 3204                      64            DIV1    R0,R2           ;
  65 0000105E 4124                      65            ROTCL   R1              ;
  66 00001060 3204                      66            DIV1    R0,R2           ;
  67 00001062 4124                      67            ROTCL   R1              ;
  68 00001064 3204                      68            DIV1    R0,R2           ;
  69 00001066 4124                      69            ROTCL   R1              ;
  70 00001068 3204                      70            DIV1    R0,R2           ;
  71 0000106A 4124                      71            ROTCL   R1              ;
  72 0000106C 3204                      72            DIV1    R0,R2           ;
  73 0000106E 4124                      73            ROTCL   R1              ;
  74 00001070 3204                      74            DIV1    R0,R2           ;
  75                                    75                                    ;
  76 00001072 4124                      76            ROTCL   R1              ;
  77 00001074 3204                      77            DIV1    R0,R2           ;
  78 00001076 4124                      78            ROTCL   R1              ;
  79 00001078 3204                      79            DIV1    R0,R2           ;
  80 0000107A 4124                      80            ROTCL   R1              ;
  81 0000107C 3204                      81            DIV1    R0,R2           ;
  82 0000107E 4124                      82            ROTCL   R1              ;
  83 00001080 3204                      83            DIV1    R0,R2           ;
  84 00001082 4124                      84            ROTCL   R1              ;
  85 00001084 3204                      85            DIV1    R0,R2           ;
  86 00001086 4124                      86            ROTCL   R1              ;
  87 00001088 3204                      87            DIV1    R0,R2           ;
  88 0000108A 4124                      88            ROTCL   R1              ;
  89 0000108C 3204                      89            DIV1    R0,R2           ;
  90 0000108E 4124                      90            ROTCL   R1              ;
  91 00001090 3204                      91            DIV1    R0,R2           ;
  92                                    92                                    ;
  93 00001092 4124                      93            ROTCL   R1              ;
  94 00001094 313E                      94            ADDC    R3,R1           ;
  95 00001096 0008                      95            CLRT                    ; T bit <- No error
  96 00001098 63F6                      96            MOV.L   @R15+,R3        ; Return register
  97 0000109A 000B                      97            RTS                     ;
  98 0000109C 62F6                      98            MOV.L   @R15+,R2        ;
  99 0000109E                           99    DIVS32Q1                        ;
 100 0000109E 0018                     100            SETT                    ; T bit <- Error
 101 000010A0 63F6                     101            MOV.L   @R15+,R3        ; Return register
 102 000010A2 000B                     102            RTS                     ;
 103 000010A4 62F6                     103            MOV.L   @R15+,R2        ;
 104                                   104            .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

3．2．14　32ビット÷32ビットの剰余（符号付き）

32ビット÷32ビットの剰余(符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32R	使用機能	DIV0S 命令 DIV1 命令

機　能

被除数(符号付き32ビット)と除数(符号付き32ビット)の除算を行い、剰余(符号付き32ビット)を求めます。また、エラー(0による除算)の有無をTビットに示します。

引　数

内　容		格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入　力	被除数(符号付き32ビット)	R1	4
	除数(符号付き32ビット)	R0	4
出　力	剰余(符号付き32ビット)	R2	4
	エラー(0による除算)の有無(有:T=1，無:T=0)	Tビット(SR)	4

内部レジスタ変化およびフラグ変化

実行前　→　実行後

R0	除数(符号付き32ﾋﾞｯﾄ)　→　変化なし
R1	被除数(符号付き32ﾋﾞｯﾄ)　→　変化
R2	不定　→　剰余(符号付き32ﾋﾞｯﾄ)
R3	ワーク
R4	ワーク
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

Tビット ※

—：不変
※：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ (ﾊﾞｲﾄ)
182
ﾃﾞｰﾀﾒﾓﾘ (ﾊﾞｲﾄ)
0
ｽﾀｯｸ (ﾊﾞｲﾄ)
8
ｽﾃｰﾄ数
87
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

プログラミング仕様のステート数は、H'80000000÷H'7FFFFFFFのときの値です。

32ビット÷32ビットの剰余(符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32R	使用機能	DIV0S 命令 DIV1 命令

説　明

(1) 機能説明

(a) 引数の詳細は以下の通りです。

R0：入力引数として、除数(符号付き32ビット)をセットします。

R1：入力引数として、被除数(符号付き32ビット)をセットします。

R2：出力引数として、剰余(符号付き32ビット)がセットされます。

Tビット(SR)：エラー(0による除算)の有無を示します。

Tビット=1　実行した除算にエラー(0による除算)が発生したことを示します。

Tビット=0　実行した除算にエラー(0による除算)が発生しなかったことを示します。

(b) 図3.37にソフトウェアDIVS32Rの実行例を示します。

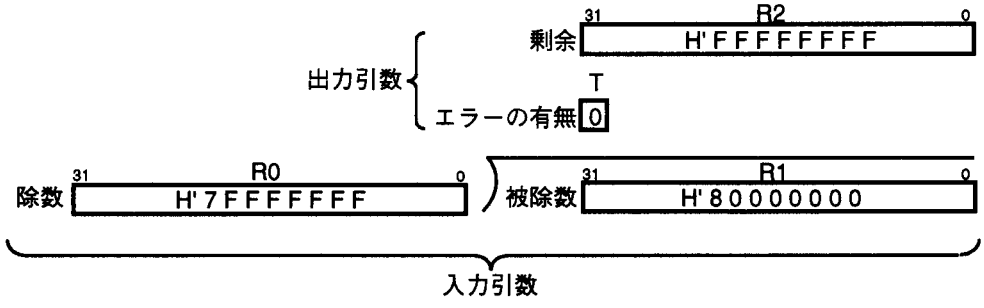


図3.37　ソフトウェアDIVS32Rの実行例

(2) 使用上の注意

被除数がセットされていたR1は、ソフトウェアDIVS32Rの実行により内容が変化します。

ソフトウェアDIVS32R実行後も被除数を必要とする場合、被除数をあらかじめ退避してください。

(3) 使用RAM説明

ソフトウェアDIVS32RではRAMは使用していません。

(4) 使用例

被除数および除数 をセットしてからソフトウェアDIVS32Rをサブルーチンコールします。

```
        MOV. L  DATA1,R1     ……被除数(符号なし32ビット)を入力引数(R1)にセット
        BSR     DIVS32R      ……ソフトウェアDIVS32Rをサブルーチンコール
        MOV. L  DATA2,R0     ……除数(符号なし32ビット)を入力引数(R0)にセット
        BT      ERROR        ……エラー(0による除算)が発生した場合、エラー処理
                                 ルーチンへ分岐
                :
        .align  4
DATA1   .data.l H'80000000
DATA2   .data.l H'7FFFFFFF
```

32ビット÷32ビットの剰余(符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32R	使用機能	DIV0S 命令 DIV1 命令

説　明

(5) 動作原理

(a) 除算前に以下の初期設定を行ないます。

(i) R2を上位32ビットとし、被除数を64ビットに符号拡張します。(図3.38-①)

(ii) 被除数が負の場合、1ステップ除算命令で扱えるように1の補数に変換します。(図3.38-②)

(iii) 1ステップ除算で使用するM、QおよびTビットを符号付き除算の値(M=除数の符号、Q=被除数の符号、T=商の符号)に設定します。(図3.38-③)

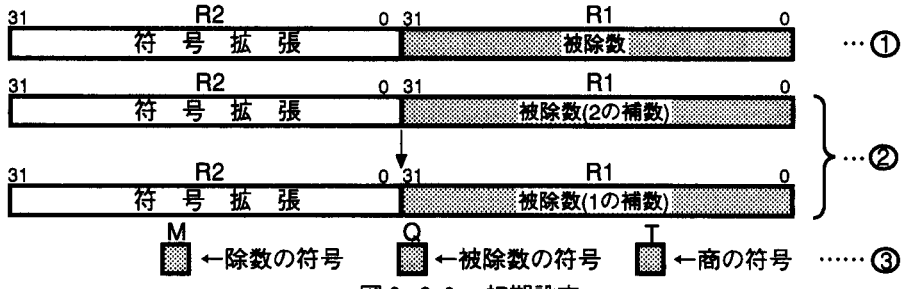


図3.38　初期設定

(b) 図3.39に示すように、除算はROTCL命令、DIV1を除数のビット数(32回)分繰り返して行います。

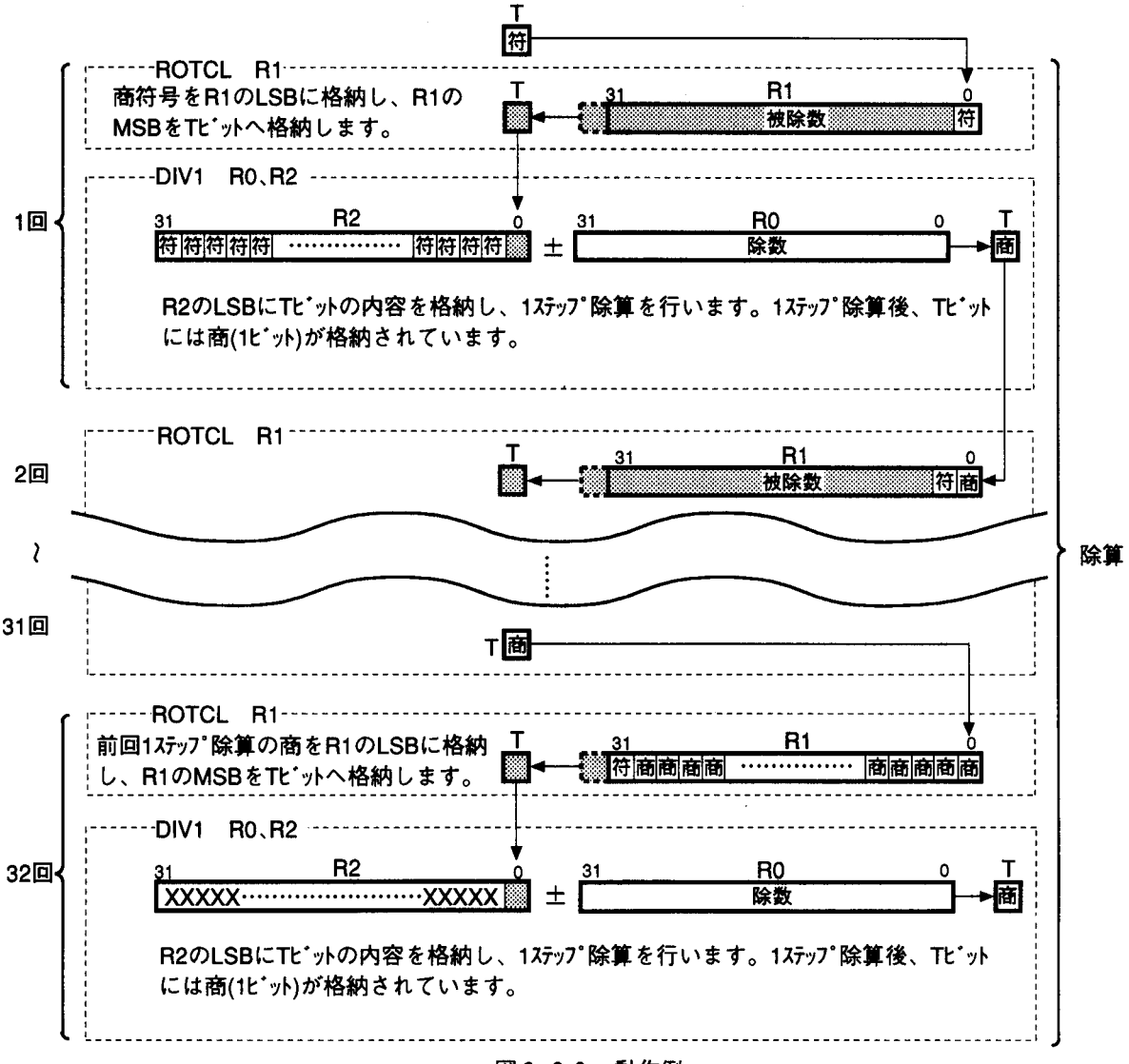


図3.39　動作例

32ビット÷32ビットの剰余(符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32R	使用機能	DIV0S 命令 DIV1 命令

説　明

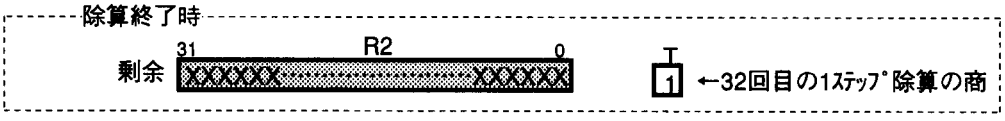
(c)

(i)　図3.40に示すように、被除数の符号およびTビット(32回目の1ステップ除算の商)の内容により、剰余の求め方が異なります。

・被除数が正のとき

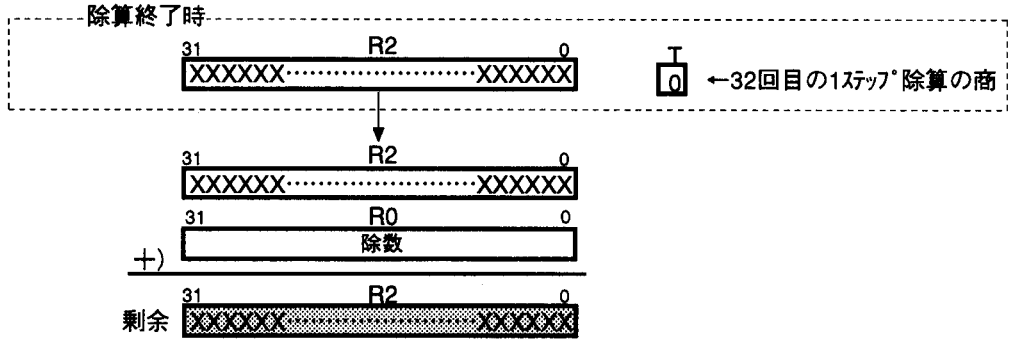
〈T=0のとき〉

除算終了時のR2の内容が剰余となります。



〈T=1のとき〉

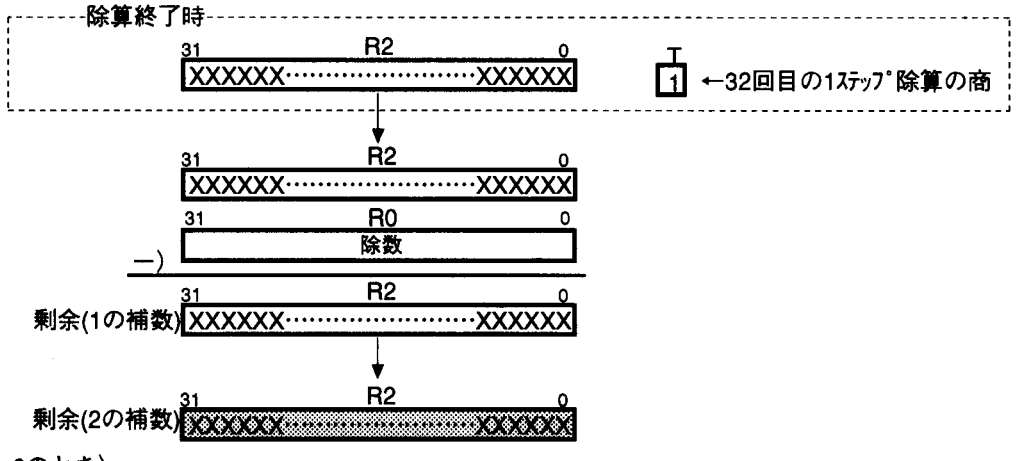
除算終了時のR2の内容は、32回目の1ｽﾃｯﾌﾟ除算の内部処理により、剰余から除数を1回減算し過ぎた値になっているので、R2に除数を加算します。



・被除数が負のとき

〈T=1のとき〉

除算終了時のR2の内容は、32回目の1ｽﾃｯﾌﾟ除算の内部処理により、剰余から除数を1回加算し過ぎた値になっているので、R2から除数を減算します。さらにR2は、1の補数表現の剰余となっているので、2の補数表現に変換します。



〈T=0のとき〉

除算終了時のR2の内容は、1の補数表現の剰余となっているので、2の補数表現に変換します。

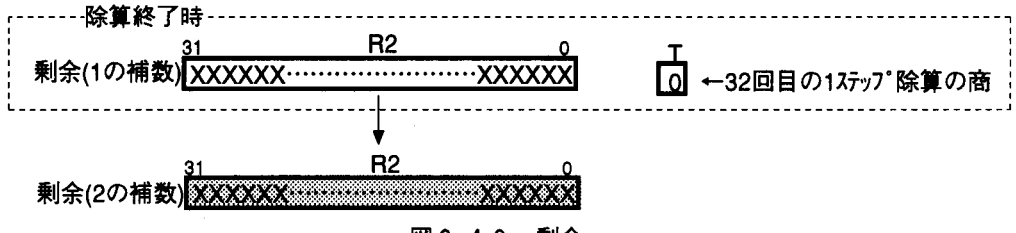


図3.40　剰余

32ビット÷32ビットの剰余(符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32R	使用機能	DIV0S 命令 DIV1 命令

説　明

(ii) ソフトウェアDIVS32Rでは、(i)の処理を以下に示すようにして行なっています。但し、R3にはH'00000000、R4のLSBには被除数の符号ビットが格納されています。

・符号なし除算の初期化命令(DIV0S)を使用して、剰余の符号ビットをTビットへ格納し、Tビット上の剰余の符号ビットをR3へ格納します。

```
DIV0S   R3, R2
MOVT    R3
```
} 剰余の符号ビット→R3

・表3.5に示すように、被除数の符号と剰余の符号は同符号になるので、被除数の符号(R4)と剰余の符号(R3)の排他的論理和をとり、被除数の符号と剰余の符号が同符号かどうかを調べます。異符号の場合、剰余は除数を減算し過ぎまたは加算し過ぎた値となっています。

表3.5　剰余の符号

被除数の符号	剰余の符号
正	正
負	負

```
XOR     R4, R3
ROTCR   R3
BF      DIVS32R1
```
} 被除数の符号＝剰余の符号？

・減算し過ぎまたは加算し過ぎの補正は、DIV1命令を使用して行なっています。

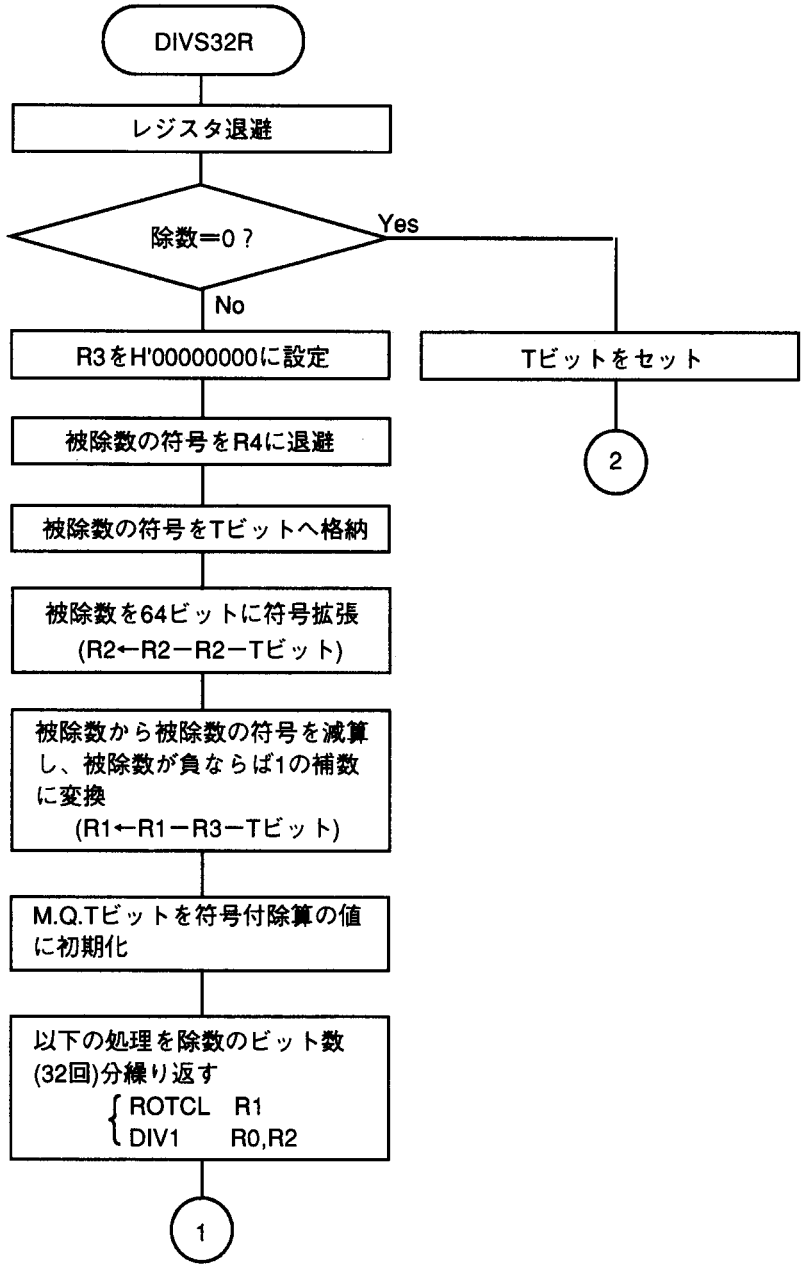
```
DIV0S   R0, R2  ←符号付き除算の初期化
SHAR    R2      ←次のDIV1命令の実行により、剰余(R2)は1/2されるので
                  R2を2倍しておきます。
DIV1    R0, R2  ←DIV1命令の内部処理により、減算し過ぎのときは
                  剰余を加算し、加算し過ぎのときは剰余を減算します。
```

・剰余とLSBに被除数の符号ビットが格納されているR4と剰余(R2)の加算を行ないます。剰余と被除数の符号は同符号なので、剰余が正ならば0が加算され剰余の値に変化はなく、剰余が負ならば1が加算されて2の補数となります。

```
ADD     R4, R2
```

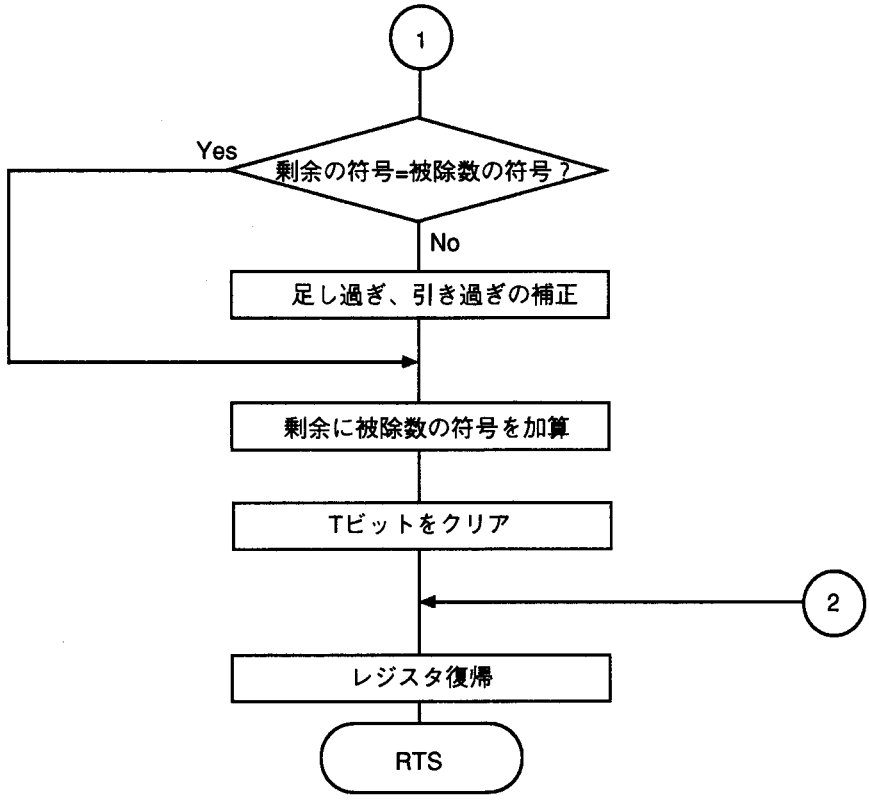
32ビット÷32ビットの剰余(符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32R	使用機能	DIV0S 命令 DIV1 命令

フローチャート



32ビット÷32ビットの剰余(符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32R	使用機能	DIV0S 命令 DIV1 命令

フローチャート



32ビット÷32ビットの剰余(符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32R	使用機能	DIV0S 命令 DIV1 命令

プログラムリスト

```
                                    1  ;*******************************************************************************
                                    2  ;*                                                                             *
                                    3  ;*          NAME  :  RESIDUAL OF 32 BIT SIGNED DIVISION (DIVS32R)              *
                                    4  ;*                                                                             *
                                    5  ;*******************************************************************************
                                    6  ;*                                                                             *
                                    7  ;*          ENTRY  :  R1    (DIVIDEND)                                         *
                                    8  ;*                    R0    (DIVISOR)                                          *
                                    9  ;*        RETURNS  :  R2    (RESIDUAL)                                         *
                                   10  ;*                    T BIT (ERROR -> TRUE;T=1,FALSE;T=0)                    *
                                   11  ;*                                                                             *
                                   12  ;*******************************************************************************
00001000                           13          .SECTION A,CODE,LOCATE=H'1000
         00001000                  14  DIVS32R .EQU    $                   ; Entry point
00001000 2F36                      15          MOV.L   R3,@-R15            ; Escape register
00001002 2F46                      16          MOV.L   R4,@-R15            ;
00001004 2008                      17          TST     R0,R0               ; Divisor = 0 ?
00001006 8952                      18          BT      DIVS32R2            ; Yes
00001008 233A                      19          XOR     R3,R3               ; R3 <- H'00000000
0000100A 2137                      20          DIV0S   R3,R1               ; T bit <- Sign of Dividend
0000100C 0429                      21          MOVT    R4                  ; R4 <- T bit
0000100E 322A                      22          SUBC    R2,R2               ; R2 sign extend
00001010 313A                      23          SUBC    R3,R1               ;
00001012 2207                      24          DIV0S   R0,R2               ; Divide as siged
                                   25                                      ;
00001014 4124                      26          ROTCL   R1                  ; Divide 1 step
00001016 3204                      27          DIV1    R0,R2               ;
00001018 4124                      28          ROTCL   R1                  ;
0000101A 3204                      29          DIV1    R0,R2               ;
0000101C 4124                      30          ROTCL   R1                  ;
0000101E 3204                      31          DIV1    R0,R2               ;
00001020 4124                      32          ROTCL   R1                  ;
00001022 3204                      33          DIV1    R0,R2               ;
00001024 4124                      34          ROTCL   R1                  ;
00001026 3204                      35          DIV1    R0,R2               ;
00001028 4124                      36          ROTCL   R1                  ;
0000102A 3204                      37          DIV1    R0,R2               ;
0000102C 4124                      38          ROTCL   R1                  ;
0000102E 3204                      39          DIV1    R0,R2               ;
00001030 4124                      40          ROTCL   R1                  ;
00001032 3204                      41          DIV1    R0,R2               ;
                                   42                                      ;
00001034 4124                      43          ROTCL   R1                  ;
00001036 3204                      44          DIV1    R0,R2               ;
00001038 4124                      45          ROTCL   R1                  ;
0000103A 3204                      46          DIV1    R0,R2               ;
0000103C 4124                      47          ROTCL   R1                  ;
0000103E 3204                      48          DIV1    R0,R2               ;
00001040 4124                      49          ROTCL   R1                  ;
00001042 3204                      50          DIV1    R0,R2               ;
00001044 4124                      51          ROTCL   R1                  ;
00001046 3204                      52          DIV1    R0,R2               ;
00001048 4124                      53          ROTCL   R1                  ;
0000104A 3204                      54          DIV1    R0,R2               ;
0000104C 4124                      55          ROTCL   R1                  ;
0000104E 3204                      56          DIV1    R0,R2               ;
00001050 4124                      57          ROTCL   R1                  ;
```

32ビット÷32ビットの剰余(符号付き)	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	DIVS32R	使用機能	DIV0S 命令 DIV1 命令

プログラムリスト

```
 58 00001052 3204                58          DIV1    R0,R2           ;
 59                              59                                  ;
 60 00001054 4124                60          ROTCL   R1              ;
 61 00001056 3204                61          DIV1    R0,R2           ;
 62 00001058 4124                62          ROTCL   R1              ;
 63 0000105A 3204                63          DIV1    R0,R2           ;
 64 0000105C 4124                64          ROTCL   R1              ;
 65 0000105E 3204                65          DIV1    R0,R2           ;
 66 00001060 4124                66          ROTCL   R1              ;
 67 00001062 3204                67          DIV1    R0,R2           ;
 68 00001064 4124                68          ROTCL   R1              ;
 69 00001066 3204                69          DIV1    R0,R2           ;
 70 00001068 4124                70          ROTCL   R1              ;
 71 0000106A 3204                71          DIV1    R0,R2           ;
 72 0000106C 4124                72          ROTCL   R1              ;
 73 0000106E 3204                73          DIV1    R0,R2           ;
 74 00001070 4124                74          ROTCL   R1              ;
 75 00001072 3204                75          DIV1    R0,R2           ;
 76                              76                                  ;
 77 00001074 4124                77          ROTCL   R1              ;
 78 00001076 3204                78          DIV1    R0,R2           ;
 79 00001078 4124                79          ROTCL   R1              ;
 80 0000107A 3204                80          DIV1    R0,R2           ;
 81 0000107C 4124                81          ROTCL   R1              ;
 82 0000107E 3204                82          DIV1    R0,R2           ;
 83 00001080 4124                83          ROTCL   R1              ;
 84 00001082 3204                84          DIV1    R0,R2           ;
 85 00001084 4124                85          ROTCL   R1              ;
 86 00001086 3204                86          DIV1    R0,R2           ;
 87 00001088 4124                87          ROTCL   R1              ;
 88 0000108A 3204                88          DIV1    R0,R2           ;
 89 0000108C 4124                89          ROTCL   R1              ;
 90 0000108E 3204                90          DIV1    R0,R2           ;
 91 00001090 4124                91          ROTCL   R1              ;
 92 00001092 3204                92          DIV1    R0,R2           ;
 93                              93                                  ;
 94 00001094 2237                94          DIV0S   R3,R2           ; R2 : keep sign
 95 00001096 0329                95          MOVT    R3              ;
 96 00001098 234A                96          XOR     R4,R3           ; ( R4 xor R3 ) == 1 ? -> oversub or overadd
 97 0000109A 4325                97          ROTCR   R3              ;
 98 0000109C 8B02                98          BF      DIVS32R1        ; Tbit = 0 ?
 99 0000109E 2207                99          DIV0S   R0,R2           ; Clear oversub or overadd
100 000010A0 4221               100          SHAR    R2              ;
101 000010A2 3204               101          DIV1    R0,R2           ;
102 000010A4                    102  DIVS32R1                        ;
103 000010A4 324C               103          ADD     R4,R2           ;
104 000010A6 0008               104          CLRT                    ; T bit <- No error
105 000010A8 64F6               105          MOV.L   @R15+,R4        ; Return register
106 000010AA 000B               106          RTS                     ;
107 000010AC 63F6               107          MOV.L   @R15+,R3        ;
108 000010AE                    108  DIVS32R2                        ;
109 000010AE 0018               109          SETT                    ; T bit <- Error
110 000010B0 64F6               110          MOV.L   @R15+,R4        ; Return register
111 000010B2 000B               111          RTS                     ;
112 000010B4 63F6               112          MOV.L   @R15+,R3        ;
113                             113          .END
*****TOTAL ERRORS       0
```

3．2．15　アフィン変換

アフィン変換	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	AFIN	使用機能	MAC．W命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接

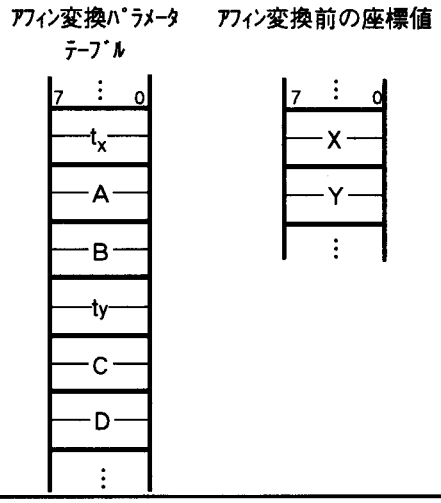
機　能

アフィン変換の行列演算を行います。但し、下図に示すデータテーブルが用意されているものとします。

アフィン変換行列

$$\begin{bmatrix} A & B & t_x \\ C & D & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix}$$

（X,Y）：アフィン変換前の座標値
（X',Y'）：アフィン変換後の座標値
A,B,C,D：アフィン変換のパラメータ
t_x, ty：アフィン変換時のX／Y座標シフト量



引　数

	内　容	格納場所	ﾃﾞｰﾀ長(ﾊﾞｲﾄ)
入力	アフィン変換パラメータテーブルの先頭アドレス	R0	4
	アフィン変換前の座標値の格納アドレス	R1	4
	アフィン変換後の座標値の格納アドレス	R2	4

内部レジスタ変化およびフラグ変化

実行前　→　実行後

R0	ｱﾌｨﾝ変換ﾊﾟﾗﾒｰﾀﾃｰﾌﾞﾙの先頭ｱﾄﾞﾚｽ　→　不定
R1	ｱﾌｨﾝ変換前の座標値の格納ｱﾄﾞﾚｽ　→　不定
R2	ｱﾌｨﾝ変換後の座標値の格納ｱﾄﾞﾚｽ　→　不定
R3	ワーク
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	(SP)

T [－]　－：不変
＊：変化
0：0固定
1：1固定

プログラミング仕様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ（ﾊﾞｲﾄ）
34
ﾃﾞｰﾀﾒﾓﾘ（ﾊﾞｲﾄ）
0
ｽﾀｯｸ（ﾊﾞｲﾄ）
4
ｽﾃｰﾄ数
22
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

アフィン変換	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	AFIN	使用機能	MAC．W命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接

説　明

（1） 機能説明

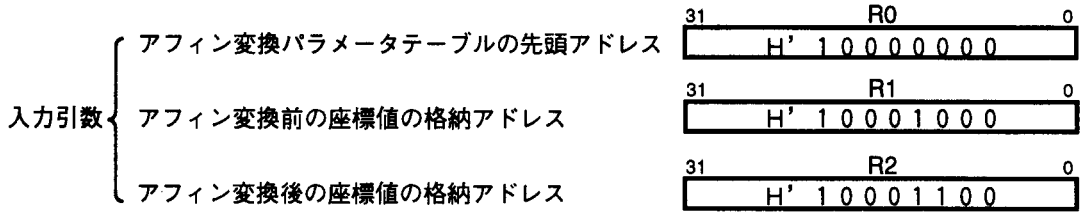
(a) 引数の詳細は以下の通りです。

R0：入力引数として、アフィン変換パラメータテーブルの先頭アドレスをセットします。

R1：入力引数として、アフィン変換前の座標値が格納されているアドレスをセットします。

R2：入力引数として、アフィン変換後の座標値を格納するアドレスをセットします。

(b) 図4.4.1にソフトウェアAFINの実行例を示します。メモリ上にはアフィン変換パラメータがH'1000 0000番地からt_x、A、B、ty、C、Dの順に、アフィン変換前の座標値がH'1000 1000番地からX、Yの順にあらかじめ配置されています。ソフトウェアAFINへは、入力引数として、アフィン変換パラメータテーブルの先頭アドレス、アフィン変換前の座標値の格納アドレスおよびアフィン変換後の座標値の格納アドレスを渡します。ソフトウェアAFINでは、アフィン変換行列の演算を行い、アフィン変換後の座標値を入力引数で指定されたH'1000 1100番地からX'、Y'の順に配置します。



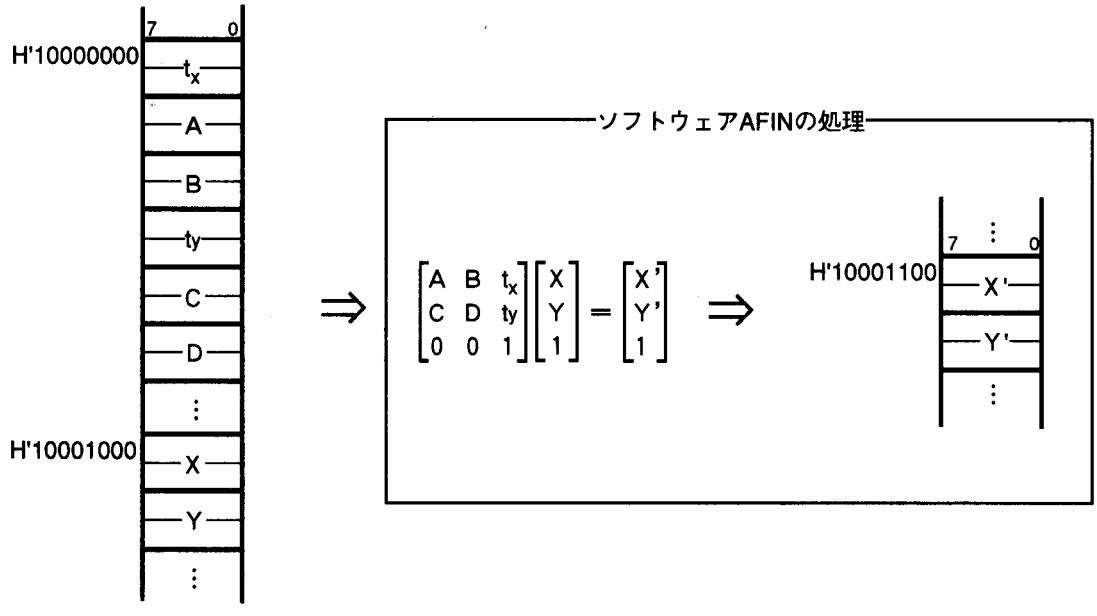


図4.4.1　ソフトウェアAFINの実行例

（2） 使用上の注意

アフィン変換パラメータおよびアフィン変換前の座標値は、図4.4.1に示すように必ず配置してください。

（3） 使用RAM説明

ソフトウェアAFINではRAMは使用していません。

アフィン変換	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	AFIN	使用機能	MAC．W命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接

説　明

（4） 使用例

アフィン変換パラメータテーブルの先頭アドレス、アフィン変換前の座標値の格納アドレスおよびアフィン変換後の座標値の格納アドレスを入力引数にセットしてからソフトウェアAFINをサブルーチンコールします。

```
                MOV. L    DATA1,R0      ……アフィン変換ﾊﾟﾗﾒｰﾀﾃｰﾌﾞﾙの先頭ｱﾄﾞﾚｽを入力引数にｾｯﾄ
                MOV. L    DATA2,R1      ……アフィン変換前の座標値が格納されているｱﾄﾞﾚｽを
                                              入力引数にｾｯﾄ
                BSR       AFIN          ……ソフトウェアAFINをサブルーチンコール
                MOV. L    DATA3,R2      ……アフィン変換後の座標値を格納するｱﾄﾞﾚｽを入力引数
                                              にｾｯﾄ
                          ⋮
                .align    4
DATA1           .data.l   H'10000000
DATA2           .data.l   H'10001000
DATA3           .data.l   H'10001100
```

（5） 動作原理

(a) アフィン変換行列を展開すると下式のようになります。

$$X' = AX + BY + t_x$$
$$Y' = CX + DY + t_y$$

(b) 図3.42に示すように、$AX+BY+t_x$および$CX+DY+t_y$を積和演算命令(MAC)を使用して求めます。

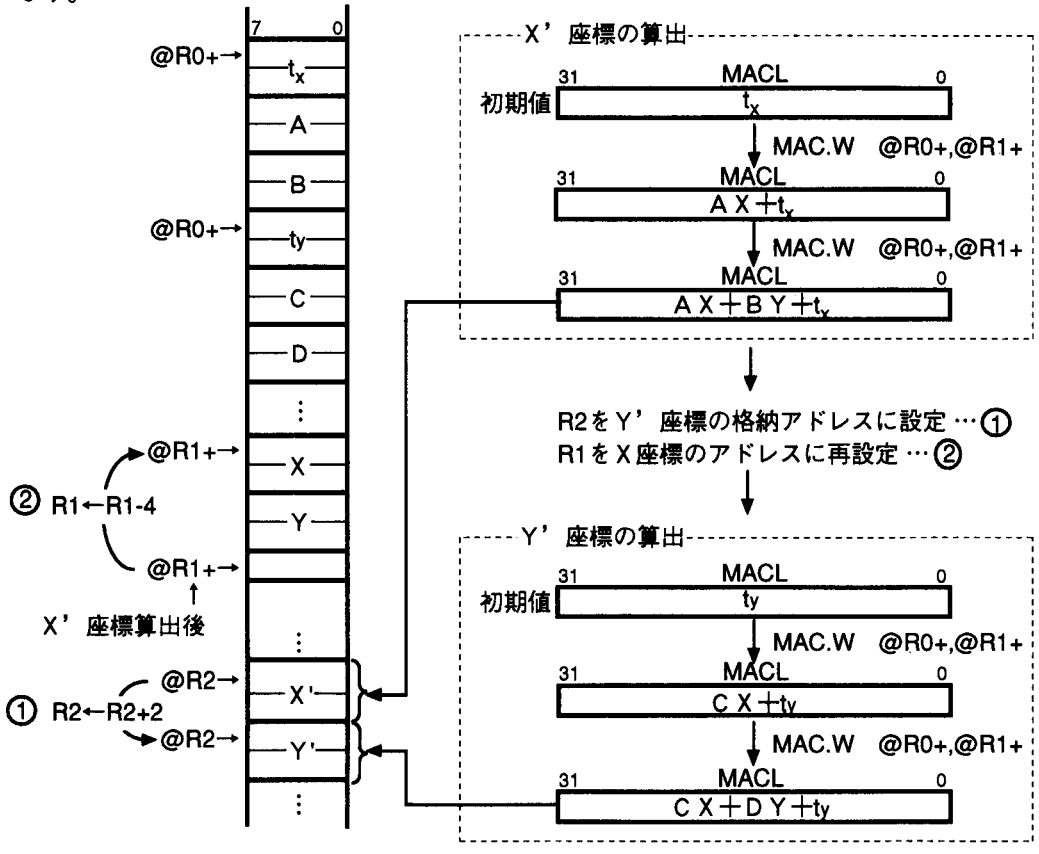
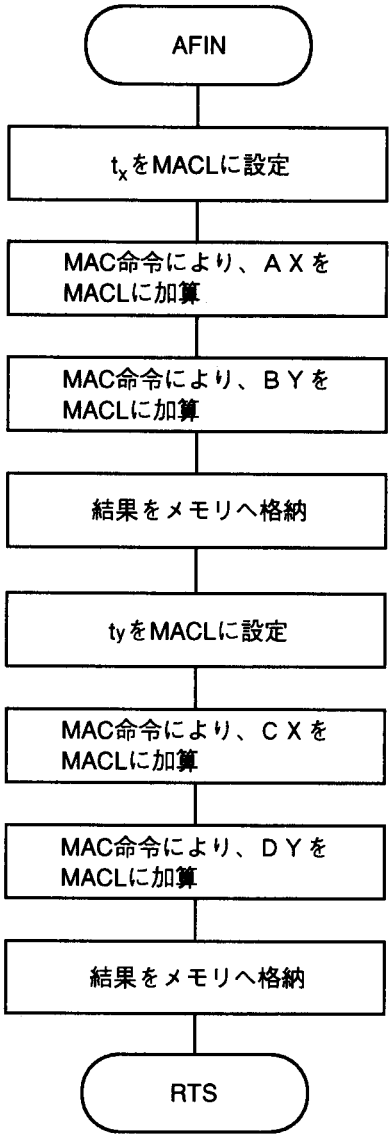


図4.42　X'およびY'座標の算出

アフィン変換	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	AFIN	使用機能	MAC．W命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接

フローチャート



アフィン変換	MCU	SH7000ｼﾘｰｽﾞ	ﾗﾍﾞﾙ名	AFIN	使用機能	MAC．W 命令 ﾎﾟｽﾄｲﾝｸﾘﾒﾝﾄﾚｼﾞｽﾀ間接

プログラムリスト

```
  1                                     1    ;*******************************************************************************
  2                                     2    ;*                                                                             *
  3                                     3    ;*          NAME  : AFIN CONVERSION (AFIN)                                     *
  4                                     4    ;*                                                                             *
  5                                     5    ;*******************************************************************************
  6                                     6    ;*                                                                             *
  7                                     7    ;*          ENTRY  :  R0   (TOP ADDRESS OF PARAMETER)                          *
  8                                     8    ;*                    R1   (STORED ADDRESS OF BEFORE AFIN CONVERSION)           *
  9                                     9    ;*                    R2   (STOR ADDRESS OF AFTER AFIN CONVERSION)              *
 10                                    10    ;*                                                                             *
 11                                    11    ;*******************************************************************************;
 12 00001000                           12            .SECTION A,CODE,LOCATE=H'1000
 13          00001000                  13    AFIN    .EQU    $               ; Entry point
 14 00001000 2F36                      14            MOV.L   R3,@-R15        ; Escape register
 15 00001002 6305                      15            MOV.W   @R0+,R3         ; tx -> MACL
 16 00001004 431A                      16            LDS     R3,MACL         ;
 17 00001006 410F                      17            MAC.W   @R0+,@R1+       ; A * X + MACL -> MACL ( = AX + Tx)
 18 00001008 410F                      18            MAC.W   @R0+,@R1+       ; B * Y + MACL -> MACL ( = AX + BY + Tx)
 19 0000100A 031A                      19            STS     MACL,R3         ; MACL -> X'
 20 0000100C 2231                      20            MOV.W   R3,@R2          ;
 21 0000100E 7202                      21            ADD     #2,R2           ;
 22 00001010 6305                      22            MOV.W   @R0+,R3         ; Ty -> MACL
 23 00001012 431A                      23            LDS     R3,MACL         ;
 24 00001014 71FC                      24            ADD     #-4,R1          ;
 25 00001016 410F                      25            MAC.W   @R0+,@R1+       ; C * X + MACL -> MACL ( = CX + Ty)
 26 00001018 410F                      26            MAC.W   @R0+,@R1+       ; D * Y + MACL -> MACL ( = CX + DY + Ty)
 27 0000101A 031A                      27            STS     MACL,R3         ; MACL -> Y'
 28 0000101C 2231                      28            MOV.W   R3,@R2          ;
 29 0000101E 000B                      29            RTS                     ;
 30 00001020 63F6                      30            MOV.L   @R15+,R3        ; Return register
 31                                    31            .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

付　録

付録　　目次

A．命令セット一覧

命令の命令コード、動作、実行ステートを、以下の形式で分類順に説明します。

命　令	命令コード	動作の概略	実行ステート	Tビット
ニーモニックで表示しています。 記号の説明 OP.Sz SRC, DEST OP:　オペコード Sz:　サイズ SRC:　ソース DEST:　デスティネーション Rm:　ソースレジスタ Rn:　デスティネーションレジスタ imm:　イミディエイトデータ disp:　ディスプレースメント	MSB ←→ LSB の順で表示しています。 記号の説明 mmmm:ソースレジスタ nnnn:　デスティネーションレジスタ 0000:　R0 0001:　R1 ..... 1111:　R15 iiii:　イミディエイトデータ dddd:　ディスプレースメント	動作の概略を表示しています。 記号の説明 →、←:　転送方向 (xx):　メモリオペランド M/Q/T:　SR 内のフラグビット &:　ビット毎の論理積 |:　ビット毎の論理和 ^:　ビット毎の排他的論理和 ~:　ビット毎の論理否定 <<n:　左 n ビットシフト >>n:　右 n ビットシフト	ノーウェイトのときの値です。*	命令実行後の、Tビットの値を表示しています。 記号の説明 —:　変化しない

【注】*　命令の実行ステートについて

表に示した実行ステートは最少値です。実際は、

(1)　命令フェッチとデータアクセスの競合が起こる場合

(2)　ロード命令（メモリ→レジスタ）のデスティネーションレジスタと、その直後の命令が使うレジスタが同一な場合

などの条件により、命令実行ステート数は増加します。

(1) データ転送命令

命　令	命令コード	動　作	実行ステート	Tビット
MOV #imm, Rn	1110nnnn iiiiiiii	#imm → 符号拡張 → Rn	1	—
MOV.W @(disp, PC), Rn	1001nnnndddddddd	(disp×2+PC) → 符号拡張 → Rn	1	—
MOV.L @(disp, PC), Rn	1101nnnndddddddd	(disp×4+PC) → Rn	1	—
MOV Rm, Rn	0110nnnnmmmm0011	Rm → Rn	1	—
MOV.B Rm, @Rn	0010nnnnmmmm0000	Rm → (Rn)	1	—
MOV.W Rm, @Rn	0010nnnnmmmm0001	Rm → (Rn)	1	—
MOV.L Rm, @Rn	0010nnnnmmmm0010	Rm → (Rn)	1	—
MOV.B @Rm, Rn	0110nnnnmmmm0000	(Rm) → 符号拡張 → Rn	1	—
MOV.W @Rm, Rn	0110nnnnmmmm0001	(Rm) → 符号拡張 → Rn	1	—
MOV.L @Rm, Rn	0110nnnnmmmm0010	(Rm) → Rn	1	—
MOV.B Rm, @-Rn	0010nnnnmmmm0100	Rn-1 → Rn, Rm → (Rn)	1	—
MOV.W Rm, @-Rn	0010nnnnmmmm0101	Rn-2 → Rn, Rm → (Rn)	1	—
MOV.L Rm, @-Rn	0010nnnnmmmm0110	Rn-4 → Rn, Rm → (Rn)	1	—
MOV.B @Rm+, Rn	0110nnnnmmmm0100	(Rm) → 符号拡張 → Rn, Rm+1 → Rm	1	—
MOV.W @Rm+,Rn	110nnnnmmmm0101	(Rm) → 符号拡張 → Rn, Rm+2 → Rm	1	—
MOV.L @Rm+, Rn	0110nnnnmmmm0110	(Rm) → Rn, Rm+4 → Rm	1	—
MOV.B R0, @(disp, Rn)	10000000nnnndddd	R0 → (disp+Rn)	1	—
MOV.W R0, @(disp, Rn)	10000001nnnndddd	R0 → (disp×2+Rn)	1	—
MOV.L Rm, @(disp, Rn)	0001nnnnmmmmdddd	Rm → (disp×4+Rn)	1	—
MOV.B @(disp, Rm), R0	10000100mmmmdddd	(disp+Rm) → 符号拡張 → R0	1	—
MOV.W @(disp, Rm), R0	10000101mmmmdddd	(disp×2+Rm) → 符号拡張 → R0	1	—
MOV.L @(disp, Rm), Rn	0101nnnnmmmmdddd	(disp×4+Rm) → Rn	1	—
MOV.B Rm, @(R0, Rn)	0000nnnnmmmm0100	Rm → (R0+Rn)	1	—
MOV.W Rm, @(R0, Rn)	0000nnnnmmmm0101	Rm → (R0+Rn)	1	—
MOV.L Rm, @(R0, Rn)	0000nnnnmmmm0110	Rm → (R0+Rn)	1	—
MOV.B @(R0, Rm), Rn	0000nnnnmmmm1100	(R0+Rm) → 符号拡張 → Rn	1	—
MOV.W @(R0, Rm), Rn	0000nnnnmmmm1101	(R0+Rm) → 符号拡張 → Rn	1	—
MOV.L @(R0, Rm), Rn	0000nnnnmmmm1110	(R0+Rm) → Rn	1	—
MOV.B R0, @(disp, GBR)	11000000dddddddd	R0 → (disp+GBR)	1	—
MOV.W R0, @(disp, GBR)	11000001dddddddd	R0 → (disp×2+GBR)	1	—
MOV.L R0, @(disp, GBR)	11000010dddddddd	R0 → (disp×4+GBR)	1	—
MOV.B @(disp, GBR), R0	11000100dddddddd	(disp+GBR) → 符号拡張 → R0	1	—
MOV.W @(disp, GBR), R0	11000101dddddddd	(disp×2+GBR) → 符号拡張 → R0	1	—
MOV.L @(disp, GBR), R0	11000110dddddddd	(disp×4+GBR) → R0	1	—
MOVA @(disp, PC), R0	11000111dddddddd	disp×4+PC → R0	1	—
MOVT Rn	0000nnnn00101001	T → Rn	1	—
SWAP.B Rm, Rn	0110nnnnmmmm1000	Rm → 下位2バイトの上下バイト交換 → Rn	1	—
SWAP.W Rm, Rn	0110nnnnmmmm1001	Rm → 上下ワード交換 → Rn	1	—
XTRCT Rm, Rn	0010nnnnmmmm1101	Rm: Rn の中央32ビット → Rn	1	—

(2) 算術演算命令

命令	命令コード	動作	実行ステート	Tビット
ADD Rm, Rn	0011nnnnmmmm1100	Rn+Rm→Rn	1	—
ADD #imm, Rn	0111nnnn iiiiiiii	Rn+imm→Rn	1	—
ADDC Rm, Rn	0011nnnnmmmm1110	Rn+Rm+T→Rn, キャリ→T	1	キャリ
ADDV Rm, Rn	0011nnnnmmmm1111	Rn+Rm→Rn, オーバフロー→T	1	オーバフロー
CMP/EQ #imm, R0	10001000 iiiiiiii	R0=imm のとき 1 →T	1	比較結果
CMP/EQ Rm, Rn	0011nnnnmmmm0000	Rn=Rm のとき 1 →T	1	比較結果
CMP/HS Rm, Rn	0011nnnnmmmm0010	無符号で Rn ≧ Rm のとき 1 →T	1	比較結果
CMP/GE Rm, Rn	0011nnnnmmmm0011	有符号で Rn ≧ Rm のとき 1 →T	1	比較結果
CMP/HI Rm, Rn	0011nnnnmmmm0110	無符号で Rn > Rm のとき 1 →T	1	比較結果
CMP/GT Rm, Rn	0011nnnnmmmm0111	有符号で Rn > Rm のとき 1 →T	1	比較結果
CMP/PZ Rn	0100nnnn00010001	Rn ≧ 0 のとき 1 →T	1	比較結果
CMP/PL Rn	0100nnnn00010101	Rn > 0 のとき 1 →T	1	比較結果
CMP/STR Rm, Rn	0010nnnnmmmm1100	いずれかのバイトが等しいとき 1 →T	1	比較結果
DIV1 Rm, Rn	0011nnnnmmmm0100	1 ステップ除算 (Rn ÷ Rm)	1	計算結果
DIV0S Rm, Rn	0010nnnnmmmm0111	Rn の MSB→Q, Rm の MSB→M, M^Q→T	1	計算結果
DIV0U	0000000000011001	0→M/Q/T	1	0
EXTS.B Rm, Rn	0110nnnnmmmm1110	Rm をバイトから符号拡張→Rn	1	—
EXTS.W Rm, Rn	0110nnnnmmmm1111	Rm をワードから符号拡張→Rn	1	—
EXTU.B Rm, Rn	0110nnnnmmmm1100	Rm をバイトからゼロ拡張→Rn	1	—
EXTU.W Rm, Rn	0110nnnnmmmm1101	Rm をワードからゼロ拡張→Rn	1	—
MAC.W @Rm+, @Rn+	0100nnnnmmmm1111	符号付きで (Rn) × (Rm) + MAC → MAC	3/(2)*1	—
MULS Rm, Rn	0010nnnnmmmm1111	符号付きで Rn × Rm → MAC	1 ～ 3 *1	—
MULU Rm, Rn	0010nnnnmmmm1110	符号なしで Rn × Rm → MAC	1 ～ 3 *1	—
NEG Rm, Rn	0110nnnnmmmm1011	0-Rm→Rn	1	—
NEGC Rm, Rn	0110nnnnmmmm1010	0-Rm-T→Rn, ボロー→T	1	ボロー
SUB Rm, Rn	0011nnnnmmmm1000	Rn-Rm→Rn	1	—
SUBC Rm, Rn	0011nnnnmmmm1010	Rn-Rm-T→Rn, ボロー→T	1	ボロー
SUBV Rm, Rn	0011nnnnmmmm1011	Rn-Rm→Rn, アンダフロー→T	1	アンダフロー

【注】 *1 通常実行ステートを示します。( ) 内の値は前後の命令との競合関係による実行ステートです。

(3) 論理演算命令

命　令	命令コード	動　作	実行ステート	Tビット
AND Rm, Rn	0010nnnnmmmm1001	Rn & Rm → Rn	1	—
AND #imm, R0	11001001 iiiiiiii	R0 & imm → R0	1	—
AND.B #imm, @(R0, GBR)	11001101 iiiiiiii	(R0+GBR) & imm → (R0+GBR)	3	—
NOT Rm, Rn	0110nnnnmmmm0111	~Rm → Rn	1	—
OR Rm, Rn	0010nnnnmmmm1011	Rn | Rm → Rn	1	—
OR #imm, R0	11001011 iiiiiiii	R0 | imm → R0	1	—
OR.B #imm, @(R0, GBR)	11001111 iiiiiiii	(R0+GBR) | imm → (R0+GBR)	3	—
TAS.B @Rn	0100nnnn00011011	(Rn) が 0 のとき 1→T, 1→MSB of (Rn)	4	テスト結果
TST Rm, Rn	0010nnnnmmmm1000	Rn & Rm, 結果が 0 のとき 1→T	1	テスト結果
TST #imm, R0	11001000 iiiiiiii	R0 & imm, 結果が 0 のとき 1→T	1	テスト結果
TST.B #imm, @(R0, GBR)	11001100 iiiiiiii	(R0+GBR) & imm, 結果が 0 のとき 1→T	3	テスト結果
XOR Rm, Rn	0010nnnnmmmm1010	Rn ^ Rm → Rn	1	—
XOR #imm, R0	11001010 iiiiiiii	R0 ^ imm → R0	1	—
XOR.B #imm, @(R0, GBR)	11001110 iiiiiiii	(R0+GBR) ^ imm → (R0+GBR)	3	—

(4) シフト命令

命　令	命令コード	動　作	実行ステート	Tビット
ROTL Rn	0100nnnn00000100	T ← Rn ← MSB	1	MSB
ROTR Rn	0100nnnn00000101	LSB → Rn → T	1	LSB
ROTCL Rn	0100nnnn00100100	T ← Rn ← T	1	MSB
ROTCR Rn	0100nnnn00100101	T → Rn → T	1	LSB
SHAL Rn	0100nnnn00100000	T ← Rn ← 0	1	MSB
SHAR Rn	0100nnnn00100001	MSB → Rn → T	1	LSB
SHLL Rn	0100nnnn00000000	T ← Rn ← 0	1	MSB
SHLR Rn	0100nnnn00000001	0 → Rn → T	1	LSB
SHLL2 Rn	0100nnnn00001000	Rn << 2 → Rn	1	—
SHLR2 Rn	0100nnnn00001001	Rn >> 2 → Rn	1	—
SHLL8 Rn	0100nnnn00011000	Rn << 8 → Rn	1	—
SHLR8 Rn	0100nnnn00011001	Rn >> 8 → Rn	1	—
SHLL16 Rn	0100nnnn00101000	Rn << 16 → Rn	1	—
SHLR16 Rn	0100nnnn00101001	Rn >> 16 → Rn	1	—

(5) 分岐命令

命　令	命令コード	動　作	実行ステート	Tビット
BF disp	10001011dddddddd	T=0 のとき disp×2+PC→PC, T=1 のとき nop	3/1*2	—
BT disp	10001001dddddddd	T=1 のとき disp×2+PC→PC, T=0 のとき nop	3/1*2	—
BRA disp	1010dddddddddddd	遅延分岐、disp×2+PC→PC	2	—
BSR disp	1011dddddddddddd	遅延分岐、PC→PR, disp×2+PC→PC	2	—
JMP @Rn	0100nnnn00101011	遅延分岐、Rn→PC	2	—
JSR @Rn	0100nnnn00001011	遅延分岐、PC→PR, Rn→PC	2	—
RTS	0000000000001011	遅延分岐、PR→PC	2	—

［注］*2 分岐するときは 3 ステート、分岐しないときは 1 ステートになります。

(6) システム制御命令

命　令	命令コード	動　作	実行ステート	Tビット
CLRT	0000000000001000	0 → T	1	0
CLRMAC	0000000000101000	0 → MACH, MACL	1	—
LDC Rm, SR	0100mmmm00001110	Rm → SR	1	LSB
LDC Rm, GBR	0100mmmm00011110	Rm → GBR	1	—
LDC Rm, VBR	0100mmmm00101110	Rm → VBR	1	—
LDC.L @Rm+, SR	0100mmmm00000111	(Rm) → SR, Rm+4 → Rm	3	LSB
LDC.L @Rm+, GBR	0100mmmm00010111	(Rm) → GBR, Rm+4 → Rm	3	—
LDC.L @Rm+, VBR	0100mmmm00100111	(Rm) → VBR, Rm+4 → Rm	3	—
LDS Rm, MACH	0100mmmm00001010	Rm → MACH	1	—
LDS Rm, MACL	0100mmmm00011010	Rm → MACL	1	—
LDS Rm, PR	0100mmmm00101010	Rm → PR	1	—
LDS.L @Rm+, MACH	0100mmmm00000110	(Rm) → MACH, Rm+4 → Rm	1	—
LDS.L @Rm+, MACL	0100mmmm00010110	(Rm) → MACL, Rm+4 → Rm	1	—
LDS.L @Rm+, PR	0100mmmm00100110	(Rm) → PR, Rm+4 → Rm	1	—
NOP	0000000000001001	無操作	1	—
RTE	0000000000101011	遅延分岐、スタック領域 → PC/SR	4	—
SETT	0000000000011000	1 → T	1	1
SLEEP	0000000000011011	スリープ	3*3	—
STC SR, Rn	0000nnnn00000010	SR → Rn	1	—
STC GBR, Rn	0000nnnn00010010	GBR → Rn	1	—
STC VBR, Rn	0000nnnn00100010	VBR → Rn	1	—
STC.L SR, @-Rn	0100nnnn00000011	Rn-4 → Rn, SR → (Rn)	2	—
STC.L GBR, @-Rn	0100nnnn00010011	Rn-4 → Rn, GBR → (Rn)	2	—
STC.L VBR, @-Rn	0100nnnn00100011	Rn-4 → Rn, VBR → (Rn)	2	—
STS MACH, Rn	0000nnnn00001010	MACH → Rn	1	—
STS MACL, Rn	0000nnnn00011010	MACL → Rn	1	—
STS PR, Rn	0000nnnn00101010	PR → Rn	1	—
STS.L MACH, @-Rn	0100nnnn00000010	Rn-4 → Rn, MACH → (Rn)	1	—
STS.L MACL, @-Rn	0100nnnn00010010	Rn-4 → Rn, MACL → (Rn)	1	—
STS.L PR, @-Rn	0100nnnn00100010	Rn-4 → Rn, PR → (Rn)	1	—
TRAPA #imm	11000011 iiiiiiii	PC/SR → スタック領域、(imm) → PC	8	—

【注】*3　スリープ状態に遷移するまでのステート数です。

- 命令の実行ステートについて

表に示した実行ステートは最少値です。実際は、

(1) 命令フェッチとデータアクセスの競合が起こる場合

(2) ロード命令（メモリ→レジスタ）のデスティネーションレジスタと、その直後の命令が使うレジスタが同一な場合

などの条件により、命令実行ステート数は増加します。

．SECTION

セクションを宣言する

【書　式】

．SECTION△　セクション名［．セクション属性［，{LOCATE＝先頭アドレス / ALIGN＝境界調整数}］］

【ステートメントの要素】

(1)　ラベル

記述できません。

(2)　オペレーション

ニーモニック.SECTIONを記述します。

(3)　オペランド

(a)　第１オペランド：セクション名

セクション名は、シンボルの一種です。

(b)　第２オペランド：セクション属性

指定内容	セクションの種類
CODE	コードセクション
DATA	データセクション
STACK	スタックセクション
COMMON	コモンセクション
DUMMY	ダミーセクション

　　は指定を省略したときの選択

指定内容によって、セクションの用途別の種類が決まります。

指定を省略すると、コードセクションになります。

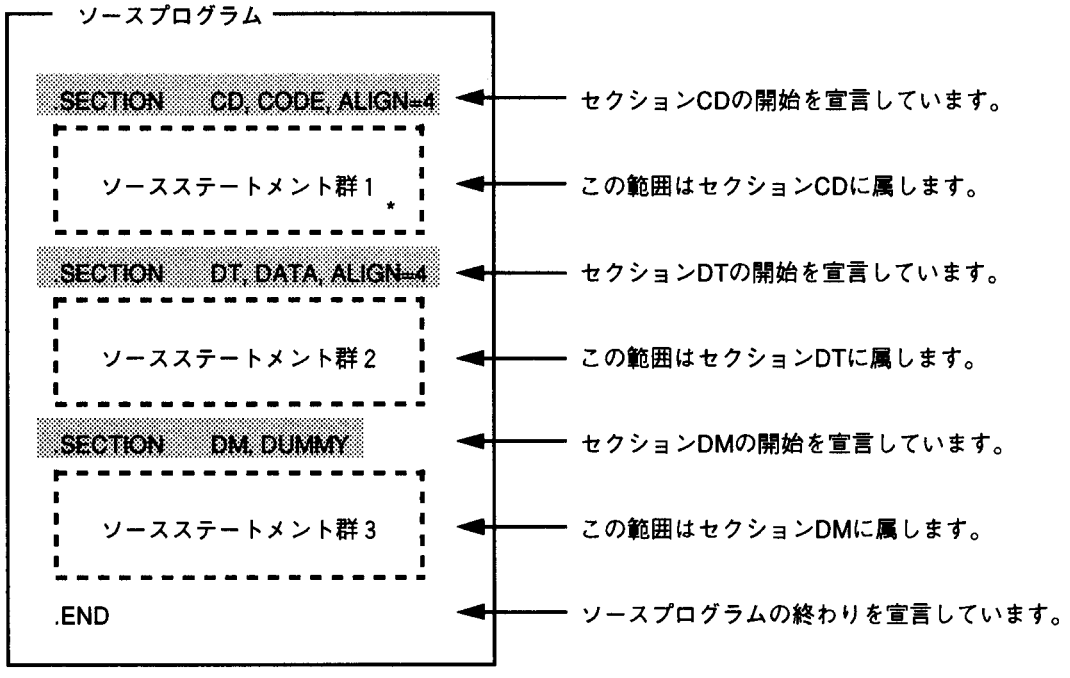
(c) 第3オペランド：先頭アドレスまたは境界調整数

指定内容	セクションの種類
LOCATE=先頭アドレス	絶対アドレスセクション
ALIGN=境界調整数	相対アドレスセクション

指定内容によって、絶対アドレスセクションになるか、相対アドレスセクションになるかが決まります。
指定を省略すると、境界調整数＝4の相対アドレスセクションになります。

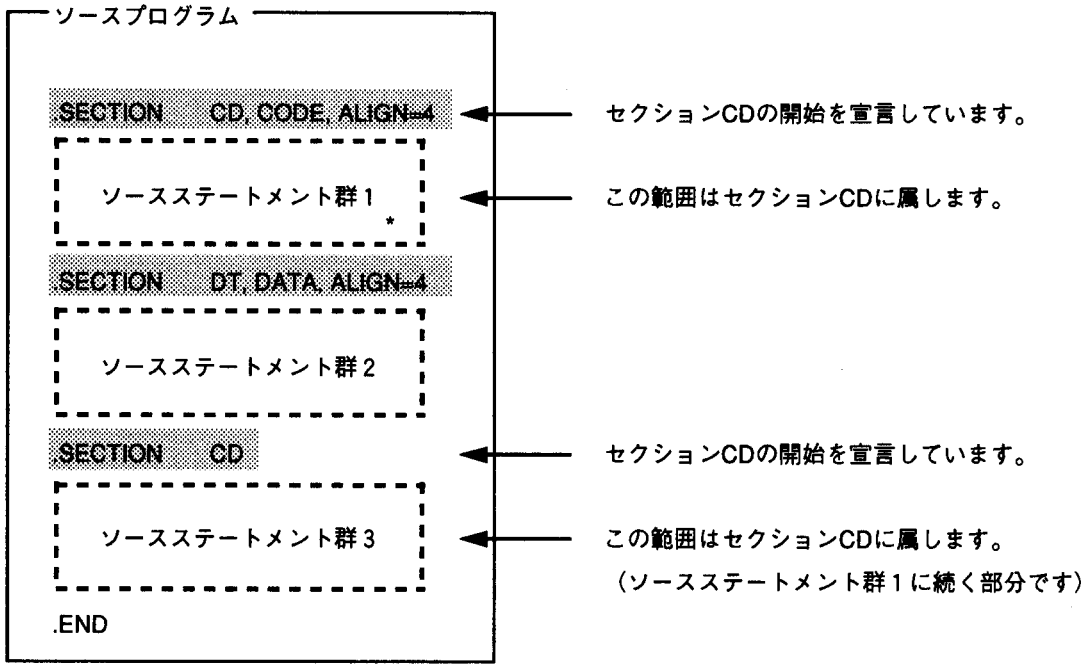
【解　説】

(1) .SECTIONは、セクションを宣言するアセンブラ制御命令です。
セクションとはプログラムの1区切りであり、リンケージの処理単位です。
セクションの宣言について、簡単な例をあげて説明します。



[注] * この例では、「ソースステートメント群1～3」に.SECTIONが現われないことを仮定しています。

(2) すでに宣言してあるセクションを同じファイルの中で再び宣言し、再開できます。
セクションの再開について、簡単な例をあげて説明します。



[注] *: この例では、「ソースステートメント群1～3」に
.SECTIONが現われないことを仮定しています。

セクションを再開する場合には、第2オペランドと第3オペランドを省略してください。
（そのセクションを開始したときの指定が、そのまま有効です）

(3) 絶対アドレスセクションを開始するときには、第3オペランドに「LOCATE=先頭アドレス」を指定してください。先頭アドレスは、そのセクションが始まる絶対アドレスです。
先頭アドレスは、次のように指定します。
— 絶対値を指定する。
かつ、
— 前方参照シンボルを使わずに指定する。
先頭アドレスとして許される値は、H'00000000～H'FFFFFFFFです。
（10進表現では-2,147,483,648～4,294,967,295）

(4) 相対アドレスセクションを開始するときには、第3オペランドに「ALIGN=境界調整数」を指定してください。
リンケージエディタは、そのセクションの先頭が境界調査数の倍数にあたる絶対アドレスにくるように調整します。

境界調整数は、次のように指定します。

ー 絶対値を指定する。

かつ、

ー 前方参照シンボルを使わずに指定する。

境界調整数として許されるの値は、2のべき乗（2^0, 2^1, 2^2, …, 2^{31}）です。

次のいずれかの場合に、アセンブラはデフォルトセクションを用意します。

ー セクションを宣言しないうちに、実行命令を記述している。

ー セクションを宣言しないうちに、データを確保するアセンブラ制御命令を記述している。

ー セクションを宣言しないうちに、.ALIGNアセンブラ制御命令を記述している。

ー セクションを宣言しないうちに、.ORGアセンブラ制御命令を記述している。

ー セクションを宣言しないうちに、ロケーションカウンタを参照している。

ー セクションを宣言しないうちに、ラベルだけの行を記述している。

デフォルトセクションとは、次のようなセクションです。

ー セクション名……P

ー セクションの種類……コードセクション

相対アドレスセクション（境界調整数＝4）

【コーディング例】

```
        .ALIGN      4               ;  この範囲は、デフォルトセクションPに属します。
        .DATA.L     H'11111111      ;  デフォルトセクションPはコードセクションであり、
         ~                          ;  境界調整数＝4の相対アドレスセクションです。
.SECTION CD, CODE, ALIGN=4
        MOV         .R0, R1         ;  この範囲は、セクションCDに属します。
        MOV         .R0, R2         ;  セクションCDはコードセクションであり、
         ~                          ;  境界調整数＝4の相対アドレスセクションです。
.SECTION DT, DATA, LOCATE=H'00001000
X1:     .DATA.L     H'22222222      ;  この範囲は、セクションDTに属します。
        .DATA.L     H'33333333      ;  セクションDTは、データセクションであり、
         ~                          ;  先頭アドレス=H'00001000の絶対アドレスセクション
        .END                        ;  です。
```

注　この例では、「 ~ 」の部分に.SECTIONが現われないことを仮定しています。

. ALIGN

ロケーションカウンタ値を補正する

【書　式】

```
. ALIGN△ 境界調整数
```

【ステートメントの要素】

(1) ラベル
　記述できません。

(2) オペレーション
　ニーモニック.ALIGNを記述します。

(3) オペランド
　境界調整数（ロケーションカウンタ値を調整する基準）として設定したい値を記述します。

【解　説】

(1) .ALIGNは、ロケーションカウンタ値を境界調整数の倍数に補正するアセンブラ制御命令です。
　.ALIGNによって、実行命令やデータを特定の境界（アドレスの区切り）に配置できます。

(2) ロケーションカウンタ値は、次のように指定します。
　　－ 絶対値を指定する。
　かつ、
　　－ 前方参照シンボルを使わずに指定する。
　境界調整数として許される値は、2のべき乗（2^0, 2^1, 2^2, …2^{31}）です。

(3) コードセクション、データセクション、ダミーセクションに.ALIGNを記述すると、アセンブラはNOP命令のオブジェクトコード*をメモリ上に埋めこみ、ロケーションカウンタ値を補正します。半端なバイトサイズの領域には、H'09を埋めこみます。

ダミーセクションまたはスタックセクションに.ALIGNを記述すると、アセンブラは単にロケーションカウンタ値を補正するだけであり、メモリ上にオブジェクトコードを埋めこみません。

[注] *: このようなオブジェクトコードは、アセンブルリスト上には表れません。

【コーディング例】

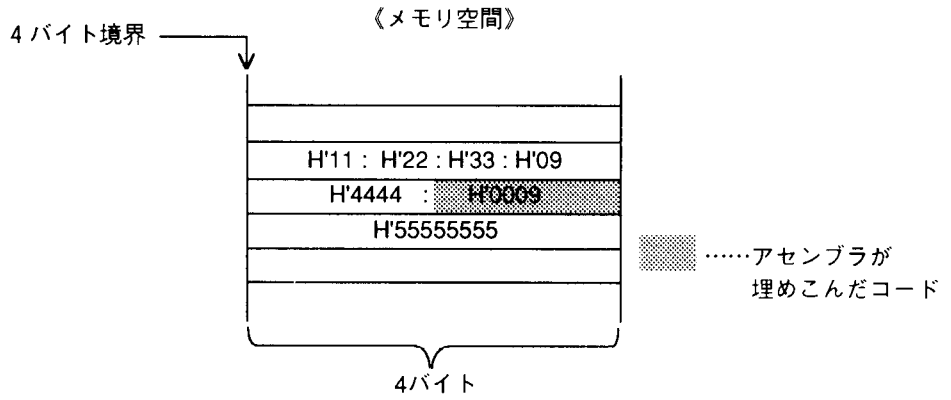
```
        ～
.DATA.B     H'11
.DATA.B     H'22
.DATA.B     H'33

.ALIGN      2           ;  ロケーションカウンタ値を2の倍数に補正しています。
.DATA.W     H'4444

.ALIGN      4           ;  ロケーションカウンタ値を4の倍数に補正しています。
.DATA.L     H'55555555
        ～
```

【コーディング例の図解】

バイトサイズの整数データH'11が、もともと4バイト境界に位置するものと仮定します。
アセンブラは、下図のようにオブジェクトコードを埋めこんで境界調整します。



. EQU

シンボルに値を設定する（再設定は不可能）

【書　式】

シンボル［:］　△　.EQU　△　シンボル値

【ステートメントの要素】

(1) ラベル

値を設定したいシンボルを記述します。

(2) オペレーション

ニーモニック.EQUを記述します。

(3) オペランド

シンボルに設定したい値を記述します。

【解　説】

(1) .EQUは、シンボル値を設定するアセンブラ制御命令です。
.EQUで定義したシンボルは、再定義できません。

(2) シンボル値は、次のように指定します。
ー 絶対値またはアドレス値を指定する。
かつ、
ー 前方参照シンボルを使わずに指定する。
シンボル値として許される値は、H'00000000～H'FFFFFFFFです。
（10進表現では-2,147,438,648～4,294,967,295）

【コーディング例】

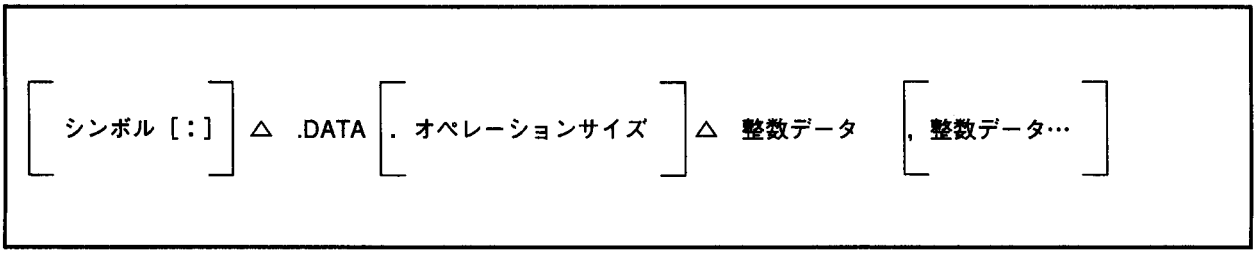
```
                ～
X1:     EQU     10          ;   X2の値は10になります。
X2:     EQU     20          ;   X2の値は20になります。

        CMP/EQ  #X1,R0      ;   CMP/EQ   #10,R0   と同じです。
        BT      LABEL1
        CMP/EQ  #X2,R0      ;   CMP/EQ   #20,R0   と同じです。
        BT      LABEL2
                ～
```

. DATA

整数データを確保する

【書　式】



【ステートメントの要素】

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

(a) ニーモニック

.DATAを記述します。

(b) オペレーションサイズ

指定内容	データのサイズ
B	バイト
W	ワード　（2バイト）
L	ロングワード（4バイト）

L は指定を省略したときの選択

指定内容によって、確保するデータのサイズが決まります。

指定を省略すると、ロングワードになります。

(3) オペランド

データとして確保したい値を記述します。

【解　説】

(1) .DATAは、整数データをメモリ上に確保するアセンブラ制御命令です。

(2) 整数データには、相対値や前方参照シンボルを含めて、任意の値を指定できます。

(3) オペレーションサイズによって、指定できる整数データの範囲が異なります。

オペレーションサイズ	整数データの範囲 *	
B	H'00000000～H'000000FF	
	H'FFFFFF80～H'FFFFFFFF	(-128～255)
W	H'00000000～H'0000FFFF	
	H'FFFF8000～H'FFFFFFFF	(-32,768～65,535)
L	H'00000000～H'FFFFFFFF	
		(-2,147,483,648～4,294,967,295)

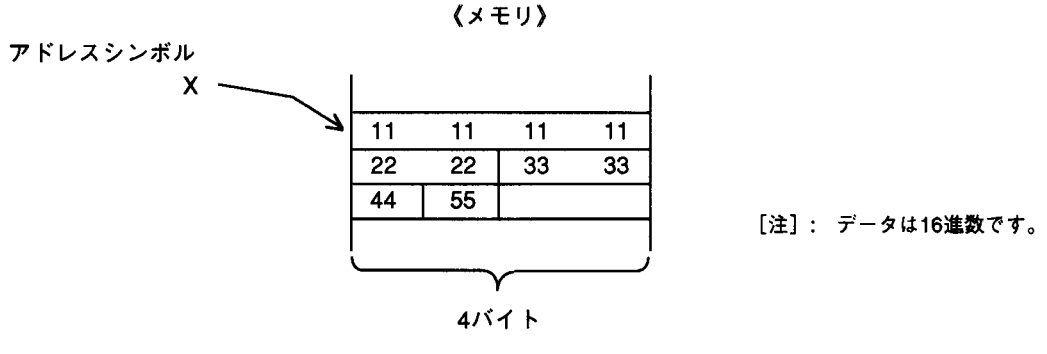
[注] *：( ) 内は10進表現

【コーディング例】

```
            ～
            .ALIGN      4           ;   (ロケーションカウンタ値を補正しています)

X:          DATA.L      H'11111111  ;
            DATA.W      H'2222,H'3333   ;   整数データを確保しています。
            DATA.B      H'44'H'55   ;
            ～
```

【コーディング例の図解】



[注]： データは16進数です。

.END

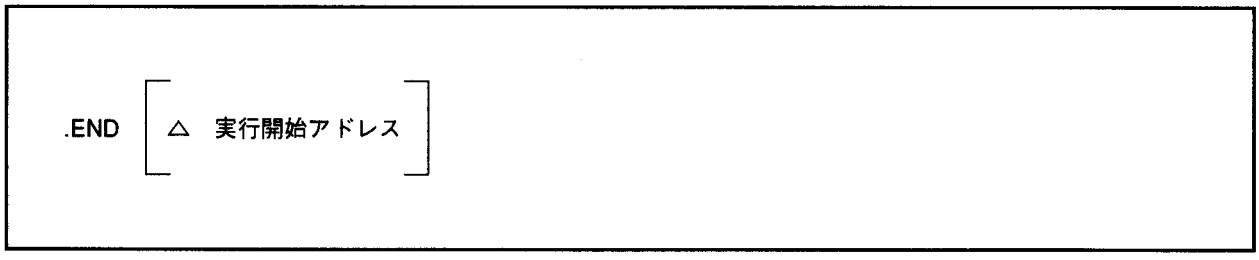
【コーディング例】

```
        .SECTION    CD,CODE,ALIGN=4
START:
        .END        START       ;  ソースプログラムの終了を宣言しています。

;  シミュレータ・デバッガは、シンボルSTARTが示すアドレスからシミュレーションを
;  開始します。
```

ソースプログラムの終わりと実行開始アドレスを宣言する

【書　式】



【ステートメントの要素】

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック.ENDを記述します。

(3) オペランド　：　実行開始アドレス

シミュレーションを開始したいアドレスを指定します。

【解　説】

(1) .ENDは、ソースプログラムの終わりを宣言するアセンブラ制御命令です。
.ENDが出現した時点で、アセンブラはアセンブル処理を終了します。

(2) .ENDで実行開始アドレスを指定しておくと、シミュレータ・デバッガは指定のアドレスからシミュレーションを開始します。

(3) 実行開始アドレスは、絶対値またはアドレス値で指定します。

(4) 実行開始アドレスには、コードセクションのアドレスを指定してください。

SH7000 シリーズ　CPU 編
アプリケーションマニュアル

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-502-044