お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

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

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

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

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

お客様各位

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

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

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

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

RENESAS

ご注意

安全設計に関するお願い

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

本資料ご利用に際しての留意事項

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

User's Manual

RENESAS

改訂一覧は表紙をクリックして直接ご覧になれます。

改訂一覧は改訂箇所をまとめたものであり、
詳細については必ず本文の内容をご確認ください。

H8/300 シリーズ

プログラミングマニュアル

ルネサスシングルチップマイクロコンピュータ

ルネサス エレクトロニクス
www.renesas.com

Rev.3.00　1993.03

ご注意

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

はじめに

Ｈ８／３００シリーズは、日立オリジナルアーキテクチャを採用したＨ８／３００ＣＰＵをコアとしています。Ｈ８／３００ＣＰＵは、８ビット×16本（または16ビット×８本）の汎用レジスタと高速動作を指向した簡潔で最適化された命令セットおよび、制御機器などの組込み用に最適なビット操作命令を備えています。このため、Ｈ８／３００シリーズの応用プログラムの作成は容易に行うことができます。また、Ｈ８／３００シリーズは、Ｈ８／３００Ｈシリーズのオブジェクトレベルで下位互換であり、容易にＨ８／３００Ｈシリーズへ移行できます。ニーモニックはＨシリーズ共通です。

本マニュアルは、Ｈ８／３００の命令の詳細について記載しており、Ｈ８／３００シリーズ共通に使用することができます。また、アセンブラについては、別にまとめた「**Ｈ８／３００シリーズクロスアセンブラユーザーズマニュアル**」が用意されていますので、あわせてご活用ください。

なお、ハードウェアの詳細については、各製品別のハードウェアマニュアルをご覧ください。

本版で改訂または追加された主な箇所

ページ	見　出　し	内　容
86	ＰＯＰ	オペランド形式の修正
87	ＰＵＳＨ	オペランド形式の修正
96	ＳＨＬＬ	コンディションコードの修正
101	ＳＵＢ（Ｗ）	コンディションコードの修正
107	命令セット一覧	MOV.W #xx:16,Rdのオペレーションの修正
116	命令実行ステート数	表の修正
119	命令の実行状態（サイクル数）	表の修正
120	命令の実行状態（サイクル数）	表の修正

目　次

第１章　ＣＰＵ

第２章　各命令の説明

図目次

〈第１章〉ＣＰＵ

〈第３章〉処理状態

〈第４章〉基本動作タイミング

表目次

〈第１章〉ＣＰＵ

〈第２章〉各命令の説明

〈第３章〉処理状態

1. ＣＰＵ

1.1 概要

Ｈ８／３００シリーズのＣＰＵは、日立オリジナルアーキテクチャを採用したＨ８／３００ＣＰＵです。Ｈ８／３００ＣＰＵは、８ビット×16本（または16ビット×８本）の汎用レジスタ、ならびに高速動作を指向した簡潔で最適化された命令セットを備えた高速ＣＰＵです。

1.1.1 特長

Ｈ８／３００ＣＰＵには、次の特長があります。

■汎用レジスタ方式
８ビット×16本（16ビット×８本としても使用可能）

■57種類の基本命令
- 乗除算命令
- 強力なビット操作命令

■８種類のアドレッシングモード
- レジスタ直接（Ｒｎ）
- レジスタ間接（@Ｒｎ）
- ディスプレースメント付レジスタ間接（@(d:16,Ｒｎ)）
- ポストインクリメント／プリデクリメントレジスタ間接（@Ｒｎ＋/@－Ｒｎ）
- 絶対アドレス（@aa:8/@aa:16）
- イミディエイト（#xx:8/#xx:16）
- プログラムカウンタ相対（@（d:8, PC））
- メモリ間接（@@aa:8）

■64ｋバイトのアドレス空間

■高速動作
- 頻出命令をすべて２～４ステートで実行
- 最高動作周波数：10MHz
  ８／16ビットレジスタ間加減算 0.2μs
  ８×８ビット乗算 1.4μs
  16÷８ビット除算 1.4μs

■低消費電力動作
- ＳＬＥＥＰ命令により低消費電力状態に遷移

1.1.2 データ構成

Ｈ８／３００ＣＰＵは、１ビット、４ビットＢＣＤ、８ビット（バイト）、および16ビット（ワード）のデータを扱うことができます。

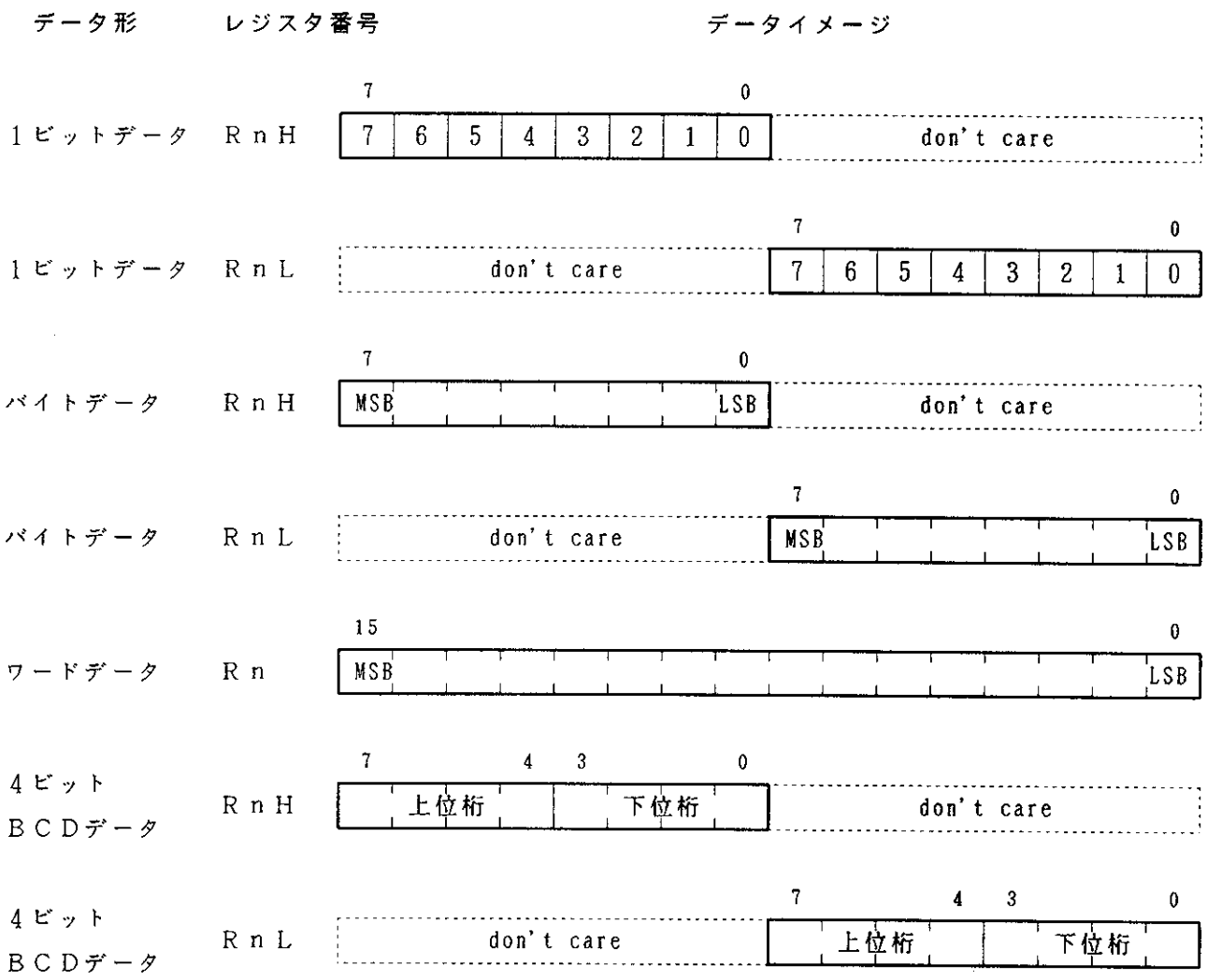
1ビットデータはビット操作命令で扱われ、オペランドデータ（バイト）の第nビット（n＝0、1、2、……7）という形式でアクセスされます。

バイトデータは、ＡＤＤＳ、ＳＵＢＳ以外の演算命令で扱われます。また、ワードデータは、ＭＯＶ.Ｗ、ＡＤＤ.Ｗ、ＳＵＢ.Ｗ、ＣＭＰ.Ｗ、ＡＤＤＳ、ＳＵＢＳ、ＭＵＬＸＵ（8×8ビット）、ＤＩＶＸＵ（16÷8ビット）命令で扱われます。

なお、ＤＡＡおよびＤＡＳの10進補正命令では、バイトデータは2桁の4ビットＢＣＤデータとなります。

(1) 汎用レジスタのデータ構成

汎用レジスタのデータ構成を図1.1に示します。



〈記号説明〉

ＲｎＨ：汎用レジスタ上位

ＲｎＬ：汎用レジスタ下位

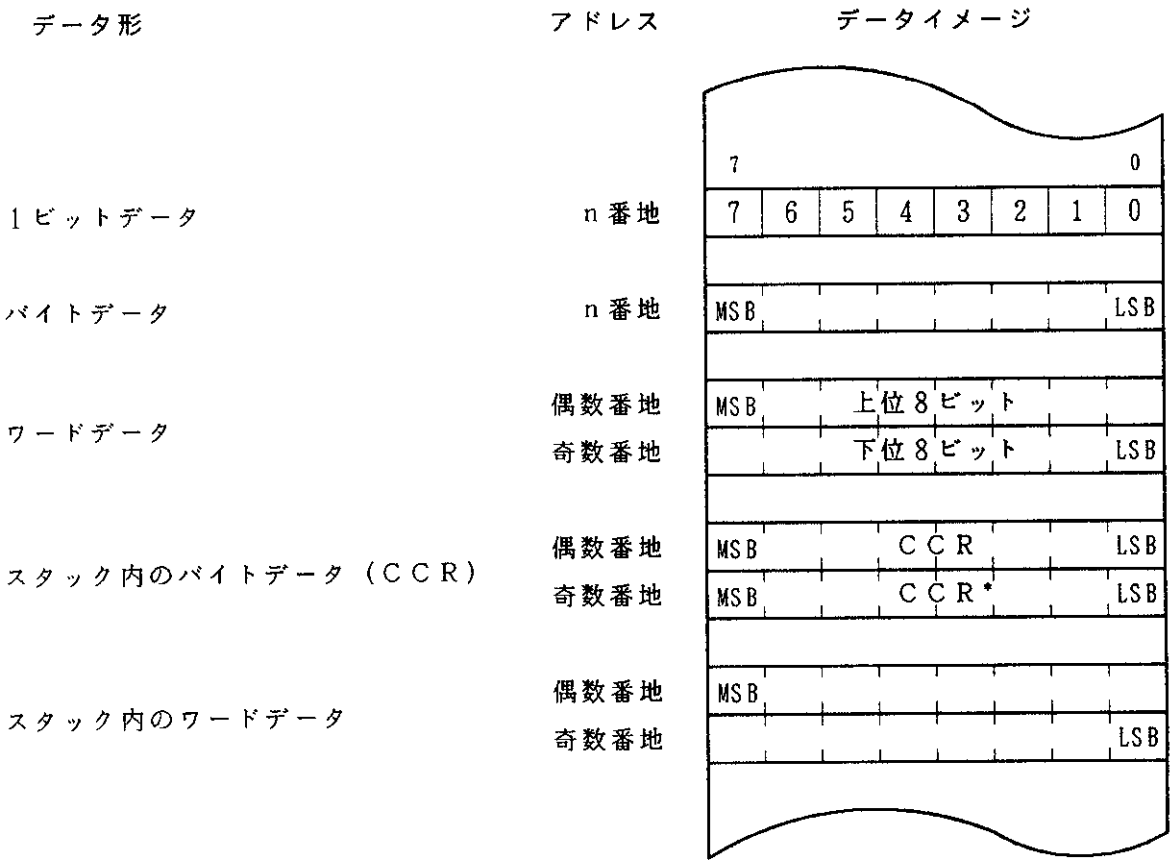
MSB　：最上位ビット

LSB　：最下位ビット

図1.1　汎用レジスタのデータ構成

(2) メモリ上でのデータ構成

メモリ上でのデータ構成を図1.2に示します。Ｈ８／３００ＣＰＵは、メモリ上のワードデータをアクセスすることができます（ＭＯＶ.Ｗ命令）が、偶数番地から始まるワードデータに限定されます。奇数番地から始まるワードデータをアクセスした場合、アドレスの最下位ビットは“０”とみなされ、１番地前から始まるワードデータをアクセスします。この場合、アドレスエラーは発生しません。命令コードについても同様です。



〈記号説明〉

ＣＣＲ：コンディションコードレジスタ

【注】 * リターン時には無視されます。

図1.2 メモリ上でのデータ構成

なお、Ｒ７をアドレスレジスタとしてスタックをアクセスするときは、必ずワードサイズでアクセスしてください。また、ＣＣＲは、ワードデータとして上位８ビット、下位８ビットに同じ値が格納され、リターン時には下位８ビットは無視されます。

1.1.3 アドレス空間

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

1.1.4　レジスタ構成

Ｈ８／３００ＣＰＵの内部レジスタ構成を図1.3に示します。これらのレジスタは、汎用レジスタとコントロールレジスタの２つに分類することができます。

汎用レジスタ（Ｒｎ）

7　　0	7　　0
Ｒ０Ｈ	Ｒ０Ｌ
Ｒ１Ｈ	Ｒ１Ｌ
Ｒ２Ｈ	Ｒ２Ｌ
Ｒ３Ｈ	Ｒ３Ｌ
Ｒ４Ｈ	Ｒ４Ｌ
Ｒ５Ｈ	Ｒ５Ｌ
Ｒ６Ｈ	Ｒ６Ｌ
Ｒ７Ｈ　　（ＳＰ）	Ｒ７Ｌ

コントロールレジスタ（ＣＲ）

15　　0
ＰＣ

	7	6	5	4	3	2	1	0
ＣＣＲ	Ｉ	Ｕ	Ｈ	Ｕ	Ｎ	Ｚ	Ｖ	Ｃ

〈記号説明〉

ＳＰ　：スタックポインタ
ＰＣ　：プログラムカウンタ
ＣＣＲ：コンディションコードレジスタ
Ｉ　　：割込みマスクビット
Ｕ　　：ユーザビット
Ｈ　　：ハーフキャリフラグ
Ｎ　　：ネガティブフラグ
Ｚ　　：ゼロフラグ
Ｖ　　：オーバフローフラグ
Ｃ　　：キャリフラグ

図1.3　ＣＰＵ内部レジスタ構成

1.2　各レジスタの説明

1.2.1　汎用レジスタ

汎用レジスタは、すべて同じ機能をもっており、データレジスタ、アドレスレジスタの区別なく使用できます。

データレジスタとして使用する場合は、8ビットレジスタとして上位（R0H～R7H）と下位（R0L～R7L）を別々に使用することも、また16ビットレジスタ（R0～R7）として使用することもできます。

アドレスレジスタとして使用する場合は、16ビットレジスタ（R0～R7）として使用します。

レジスタR7には、汎用レジスタとしての機能に加えて、スタックポインタ（SP）としての機能が割り当てられており、例外処理やサブルーチンコールなどで暗黙的に使用されます。このとき、SPは常にスタック領域の先頭を指しています。スタックの状態を図1.4に示します。

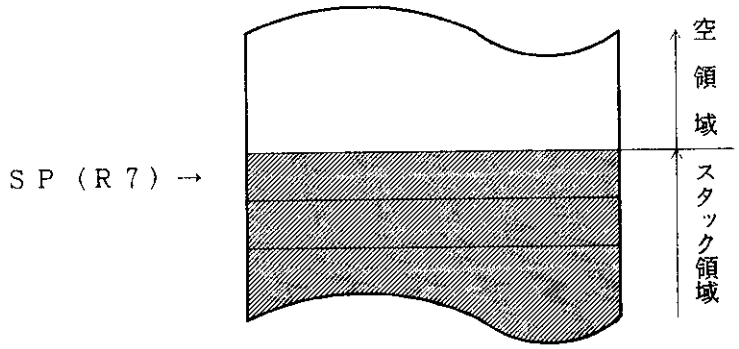


図1.4　スタックの状態

1.2.2　コントロールレジスタ

コントロールレジスタには、16ビットのプログラムカウンタ（PC）と8ビットのコンディションコードレジスタ（CCR）があります。

(1)　プログラムカウンタ（PC）

16ビットのカウンタで、CPUが次に実行する命令のアドレスを示しています。CPUの命令は、すべて16ビット（ワード）を単位としているため、最下位ビットは無効です（命令コードのリード時には最下位ビットは“0”とみなされます）。

(2)　コンディションコードレジスタ（CCR）

8ビットのレジスタで、CPUの内部状態を示しています。割込みマスクビット（I）とハーフキャリ（H）、ネガティブ（N）、ゼロ（Z）、オーバフロー（V）、キャリ（C）の各フラグを含む8ビットで構成されています。

ビット7：割込みマスクビット（I）

本ビットが“1”にセットされると、割込みがマスクされます。ただし、NMIはIビットに関係なく常に受付けられます。例外処理の実行が開始されたときに“1”にセットされます。

ビット6：ユーザビット（U）

ソフトウェア（ＬＤＣ、ＳＴＣ、ＡＮＤＣ、ＯＲＣ、ＸＯＲＣ命令）でリード／ライトできます。

ビット5：ハーフキャリフラグ（H）

ＡＤＤ.Ｂ、ＡＤＤＸ.Ｂ、ＳＵＢ.Ｂ、ＳＵＢＸ.Ｂ、ＣＭＰ.Ｂ、ＮＥＧ.Ｂ命令の実行により、ビット3にキャリまたはボローが生じたとき“1”にセットされ、生じなかったとき“0”にクリアされます。

ＤＡＡおよびＤＡＳ命令実行時に、暗黙的に使用されます。

ＡＤＤ.Ｗ、ＳＵＢ.Ｗ、ＣＭＰ.Ｗ命令ではビット11にキャリまたはボローが生じたとき“1”にセットされ、生じなかったとき“0”にクリアされます。

ビット4：ユーザビット（U）

ソフトウェア（ＬＤＣ、ＳＴＣ、ＡＮＤＣ、ＯＲＣ、ＸＯＲＣ命令）でリード／ライトできます。

ビット3：ネガティブフラグ（N）

データの最上位ビットを符号ビットとみなし、最上位ビットの値を格納します。

ビット2：ゼロフラグ（Z）

データがゼロのとき“1”にセットされ、ゼロ以外のとき“0”にクリアされます。

ビット1：オーバフローフラグ（V）

算術演算命令の実行により、オーバフローが生じたとき“1”にセットされます。それ以外のとき“0”にクリアされます。

ビット0：キャリフラグ（C）

演算の実行により、キャリが生じたとき“1”にセットされ、生じなかったとき“0”にクリアされます。キャリには次の種類があります。

(a) 加算結果のキャリ

(b) 減算結果のボロー

(c) シフト／ローテートのキャリ

また、キャリフラグには、ビットアキュムレータ機能があり、ビット操作命令で使用されます。

なお、命令によってはフラグが変化しない場合があります。

各命令ごとのフラグの変化については、2.2.1以降の各命令の説明を参照してください。

ＣＣＲは、ＬＤＣ、ＳＴＣ、ＡＮＤＣ、ＯＲＣ、ＸＯＲＣ命令で操作することができます。また、Ｎ、Ｚ、Ｖ、Ｃの各フラグは、条件分岐命令（Ｂｃｃ）で使用されます。

1.2.3 ＣＰＵ内部レジスタの初期値

リセット例外処理によって、ＣＰＵ内部レジスタのうち、ＰＣはベクタからロードすることにより初期化され、ＣＣＲのＩビットは“1”にセットされますが、汎用レジスタおよびＣＣＲの他のビットは初期化されません。レジスタＲ７（ＳＰ）の初期値も不定です。したがって、リセット直後に、Ｒ７の初期化を行ってください。

1.3　命令

Ｈ８／３００ＣＰＵの命令には、次の特長があります。

- 汎用レジスタアーキテクチャを採用
- 高速動作を指向した簡潔で最適化された57種類の基本命令
- 命令長は２バイトまたは４バイト
- 高速で実行可能な乗除算命令と強力なビット操作命令を用意
- ８種類のアドレッシングモード

1.3.1　命令の分類

Ｈ８／３００ＣＰＵの命令は合計57種類あり、各命令のもつ機能によって、表 1.1 に示すように分類されます。各命令についての詳細は 2.2.1 以降を参照してください。

表 1.1　命令の分類

機　　能	命　　令	種　類
データ転送命令	ＭＯＶ，ＭＯＶＦＰＥ，ＭＯＶＴＰＥ，ＰＯＰ*1，ＰＵＳＨ*1	3
算術演算命令	ＡＤＤ，ＳＵＢ，ＡＤＤＸ，ＳＵＢＸ，ＩＮＣ，ＤＥＣ，ＡＤＤＳ，ＳＵＢＳ，ＤＡＡ，ＤＡＳ，ＭＵＬＸＵ，ＤＩＶＸＵ，ＣＭＰ，ＮＥＧ	14
論理演算命令	ＡＮＤ，ＯＲ，ＸＯＲ，ＮＯＴ	4
シフト命令	ＳＨＡＬ，ＳＨＡＲ，ＳＨＬＬ，ＳＨＬＲ，ＲＯＴＬ，ＲＯＴＲ，ＲＯＴＸＬ，ＲＯＴＸＲ	8
ビット操作命令	ＢＳＥＴ，ＢＣＬＲ，ＢＮＯＴ，ＢＴＳＴ，ＢＡＮＤ，ＢＩＡＮＤ，ＢＯＲ，ＢＩＯＲ，ＢＸＯＲ，ＢＩＸＯＲ，ＢＬＤ，ＢＩＬＤ，ＢＳＴ，ＢＩＳＴ	14
分岐命令	Ｂｃｃ*2，ＪＭＰ，ＢＳＲ，ＪＳＲ，ＲＴＳ	5
システム制御命令	ＲＴＥ，ＳＬＥＥＰ，ＬＤＣ，ＳＴＣ，ＡＮＤＣ，ＯＲＣ，ＸＯＲＣ，ＮＯＰ	8
ブロック転送命令／EEPROM書込み専用命令	ＥＥＰＭＯＶ	1

合計57種類

【注】*1　POP Rn，PUSH Rnはそれぞれ MOV.W @SP+，Rn，MOV.W Rn，@-SPと同一です。

*2　Ｂｃｃは条件分岐命令の総称です。

1.3.2　命令の機能別一覧

表 1.2 に命令の機能別一覧を示します。また、下表に表 1.2 で使用される記号の意味を示します。

《オペレーションの記号》

Ｒｄ	汎用レジスタ（デスティネーション側）	ＳＰ	スタックポインタ
		#IMM	イミディエイトデータ
Ｒｓ	汎用レジスタ（ソース側）	disp	ディスプレースメント
Ｒｎ	汎用レジスタ	＋	加算
（ＥＡｄ）	デスティネーションオペランド	－	減算
（ＥＡｓ）	ソースオペランド	×	乗算
ＣＣＲ	コンディションコードレジスタ	÷	除算
Ｎ	ＣＣＲのＮ（ネガティブ）フラグ	∧	論理積
Ｚ	ＣＣＲのＺ（ゼロ）フラグ	∨	論理和
Ｖ	ＣＣＲのＶ（オーバフロー）フラグ	⊕	排他的論理和
		⟶	転送
Ｃ	ＣＣＲのＣ（キャリ）フラグ	～	反転論理（論理的補数）
ＰＣ	プログラムカウンタ	:3/:8/:16	3/8/16ビット長

表 1.2　命令の機能別一覧(1)

分類	命　　令	ｻｲｽﾞ*	機　　能
データ転送命令	ＭＯＶ	Ｂ／Ｗ	（ＥＡｓ）⟶Ｒｄ、Ｒｓ ⟶ （ＥＡｄ） 汎用レジスタと汎用レジスタまたは汎用レジスタとメモリ間でデータ転送します。また、イミディエイトデータを汎用レジスタに転送します。 ワードデータはＲｎ、@Ｒｎ、@（d:16，Ｒｎ）、@aa:16、#xx:16、@－Ｒｎ、@Ｒｎ＋の各アドレッシングモードで扱います。@aa:8はバイトデータのみです。 ただし、@－Ｒ７、@Ｒ７＋を使用する場合は必ずワードサイズを指定してください。

【注】 * サイズはオペランドサイズを示します。
Ｂ：バイト
Ｗ：ワード

表 1.2　命令の機能別一覧(2)

分類	命　令	ｻｲｽﾞ*	機　能
データ転送命令	MOVFPE	B	（EAs）⟶ Rd 外部メモリの内容（@aa:16で指定）をEクロックに同期したタイミングで汎用レジスタに転送します。
	MOVTPE	B	Rs ⟶ （EAd） 汎用レジスタの内容をEクロックに同期したタイミングで外部メモリ（@aa:16で指定）に転送します。
	POP	W	@SP+ ⟶ Rn スタックから汎用レジスタへデータを復帰します。 本命令はMOV.W @SP+, Rnと同一です。
	PUSH	W	Rn ⟶ @−SP 汎用レジスタの内容をスタックに退避します。 本命令はMOV.W Rn, @−SPと同一です。
算術演算命令	ADD SUB	B／W	Rd±Rs ⟶ Rd、Rd+#IMM ⟶ Rd 汎用レジスタ間の加減算、または汎用レジスタとイミディエイトデータの加算を行います。汎用レジスタとイミディエイトデータの減算はできません。（SUBX命令またはADD命令を使用してください。） ワードデータは、汎用レジスタ間の加減算のみで扱います。
	ADDX SUBX	B	Rd±Rs±C ⟶ Rd、Rd±#IMM±C ⟶ Rd 汎用レジスタ間のキャリ付の加減算、または汎用レジスタとイミディエイトデータのキャリ付の加減算を行います。
	INC DEC	B	Rd±1 ⟶ Rd 汎用レジスタに1を加減算します。
	ADDS SUBS	W	Rd±1 ⟶ Rd、Rd±2 ⟶ Rd 汎用レジスタに1または2を加減算します。
	DAA DAS	B	Rd（10進補正）⟶ Rd 汎用レジスタ上の加減算結果をCCRを参照して4ビットBCDデータに補正します。
	MULXU	B	Rd×Rs ⟶ Rd 汎用レジスタ間の符号なし乗算を行います。8ビット×8ビット→16ビットの演算が可能です。
	DIVXU	B	Rd÷Rs ⟶ Rd 汎用レジスタ間の符号なし除算を行います。16ビット÷8ビット→商8ビット 余り8ビットの演算が可能です。

【注】 * サイズはオペランドサイズを示します。
B：バイト
W：ワード

表 1.2　命令の機能別一覧(3)

分類	命　令	ｻｲｽﾞ*	機　能
算術演算命令	ＣＭＰ	Ｂ／Ｗ	Ｒｄ－Ｒｓ、Ｒｄ－#IMM 汎用レジスタ間の比較、または汎用レジスタとイミディエイトデータの比較を行い、その結果をＣＣＲに反映します。 ワードデータは、汎用レジスタ間の比較のみで扱います。
	ＮＥＧ	Ｂ	０－Ｒｄ ⟶ Ｒｄ 汎用レジスタの内容の２の補数（算術的補数）をとります。
論理演算命令	ＡＮＤ	Ｂ	Ｒｄ∧Ｒｓ ⟶ Ｒｄ、Ｒｄ∧#IMM ⟶ Ｒｄ 汎用レジスタ間の論理積、または汎用レジスタとイミディエイトデータの論理積をとります。
	ＯＲ	Ｂ	Ｒｄ∨Ｒｓ ⟶ Ｒｄ、Ｒｄ∨#IMM ⟶ Ｒｄ 汎用レジスタ間の論理和、または汎用レジスタとイミディエイトデータの論理和をとります。
	ＸＯＲ	Ｂ	Ｒｄ⊕Ｒｓ ⟶ Ｒｄ、Ｒｄ⊕#IMM ⟶ Ｒｄ 汎用レジスタ間の排他的論理和、または汎用レジスタとイミディエイトデータの排他的論理和をとります。
	ＮＯＴ	Ｂ	～Ｒｄ ⟶ Ｒｄ 汎用レジスタの内容の１の補数（論理的補数）をとります。
シフト命令	ＳＨＡＬ ＳＨＡＲ	Ｂ	Ｒｄ（シフト処理）⟶ Ｒｄ 汎用レジスタの内容を算術的にシフトします。
	ＳＨＬＬ ＳＨＬＲ	Ｂ	Ｒｄ（シフト処理）⟶ Ｒｄ 汎用レジスタの内容を論理的にシフトします。
	ＲＯＴＬ ＲＯＴＲ	Ｂ	Ｒｄ（ローテート処理）⟶ Ｒｄ 汎用レジスタの内容をローテートします。
	ＲＯＴＸＬ ＲＯＴＸＲ	Ｂ	Ｒｄ（ローテート処理）⟶ Ｒｄ 汎用レジスタの内容をキャリフラグを含めてローテートします。
ビット操作命令	ＢＳＥＴ	Ｂ	１ ⟶ （＜ビット番号＞ of ＜ＥＡｄ＞） 汎用レジスタまたはメモリのオペランドの指定された１ビットを“１”にセットします。ビット番号は、３ビットのイミディエイトデータまたは汎用レジスタの内容下位３ビットで指定されます。
	ＢＣＬＲ	Ｂ	０ ⟶ （＜ビット番号＞ of ＜ＥＡｄ＞） 汎用レジスタまたはメモリのオペランドの指定された１ビットを“０”にクリアします。ビット番号は、３ビットのイミディエイトデータまたは汎用レジスタの内容下位３ビットで指定されます。

【注】 * サイズはオペランドサイズを示します。
Ｂ：バイト
Ｗ：ワード

表 1.2　命令の機能別一覧(4)

分類	命　令	ｻｲｽﾞ*	機　能
ビット操作命令	BNOT	B	～（<ビット番号> of <EAd>） ⟶（<ビット番号> of <EAd>） 汎用レジスタまたはメモリのオペランドの指定された1ビットを反転します。ビット番号は、3ビットのイミディエイトデータまたは汎用レジスタの内容下位3ビットで指定されます。
	BTST	B	～（<ビット番号> of <EAd>）⟶ Z 汎用レジスタまたはメモリのオペランドの指定された1ビットをテストし、ゼロフラグに反映します。ビット番号は、3ビットのイミディエイトデータまたは汎用レジスタの内容下位3ビットで指定されます。
	BAND	B	C∧（<ビット番号> of <EAd>）⟶ C 汎用レジスタまたはメモリのオペランドの指定された1ビットとキャリフラグとの論理積をとり、キャリフラグに結果を格納します。
	BIAND	B	C∧〔～（<ビット番号> of <EAd>）〕⟶ C 汎用レジスタまたはメモリのオペランドの指定された1ビットを反転し、キャリフラグとの論理積をとり、キャリフラグに結果を格納します。 ビット番号は、3ビットのイミディエイトデータで指定されます。
	BOR	B	C∨（<ビット番号> of <EAd>）⟶ C 汎用レジスタまたはメモリのオペランドの指定された1ビットとキャリフラグとの論理和をとり、キャリフラグに結果を格納します。
	BIOR	B	C∨〔～（<ビット番号> of <EAd>）〕⟶ C 汎用レジスタまたはメモリのオペランドの指定された1ビットを反転し、キャリフラグとの論理和をとり、キャリフラグに結果を格納します。 ビット番号は、3ビットのイミディエイトデータで指定されます。

【注】 * サイズはオペランドサイズを示します。
B：バイト

表 1.2　命令の機能別一覧(5)

分類	命　令	ｻｲｽﾞ*	機　能
ビット操作命令	BXOR	B	C⊕（<ビット番号> of <EAd>）⟶ C 汎用レジスタまたはメモリのオペランドの指定された1ビットとキャリフラグとの排他的論理和をとり、キャリフラグに結果を格納します。
	BIXOR	B	C⊕〔～（<ビット番号> of <EAd>）〕⟶ C 汎用レジスタまたはメモリのオペランドの指定された1ビットを反転し、キャリフラグとの排他的論理和をとり、キャリフラグに結果を格納します。 ビット番号は、3ビットのイミディエイトデータで指定されます。
	BLD	B	（<ビット番号> of <EAd>）⟶ C 汎用レジスタまたはメモリのオペランドの指定された1ビットをキャリフラグに転送します。
	BILD	B	～（<ビット番号> of <EAd>）⟶ C 汎用レジスタまたはメモリのオペランドの指定された1ビットを反転し、キャリフラグに転送します。 ビット番号は、3ビットのイミディエイトデータで指定されます。
	BST	B	C ⟶ （<ビット番号> of <EAd>） 汎用レジスタまたはメモリのオペランドの指定された1ビットに、キャリフラグの内容を転送します。
	BIST	B	～C ⟶ （<ビット番号> of <EAd>） 汎用レジスタまたはメモリのオペランドの指定された1ビットに、反転されたキャリフラグの内容を転送します。 ビット番号は、3ビットのイミディエイトデータで指定されます。

【注】 * サイズはオペランドサイズを示します。
B：バイト

表 1.2　命令の機能別一覧(6)

<table>
<tr><th>分類</th><th>命　　令</th><th>ｻｲｽﾞ*</th><th>機　　能</th></tr>
<tr><td rowspan="13">分岐命令</td><td>Ｂｃｃ</td><td>－</td><td>指定した条件が成立しているとき、指定されたアドレスへ分岐します。分岐条件を下表に示します。
<table>
<tr><th>ﾆｰﾓﾆｯｸ</th><th>説　　明</th><th>分岐条件</th></tr>
<tr><td>BRA (BT)</td><td>Always (True)</td><td>Always</td></tr>
<tr><td>BRN (BF)</td><td>Never (False)</td><td>Never</td></tr>
<tr><td>BHI</td><td>HIgh</td><td>C∨Z＝0</td></tr>
<tr><td>BLS</td><td>Low or Same</td><td>C∨Z＝1</td></tr>
<tr><td>BCC (BHS)</td><td>Carry Clear (High or Same)</td><td>C＝0</td></tr>
<tr><td>BCS (BLO)</td><td>Carry Set (LOw)</td><td>C＝1</td></tr>
<tr><td>BNE</td><td>Not Equal</td><td>Z＝0</td></tr>
<tr><td>BEQ</td><td>EQual</td><td>Z＝1</td></tr>
<tr><td>BVC</td><td>oVerflow Clear</td><td>V＝0</td></tr>
<tr><td>BVS</td><td>oVerflow Set</td><td>V＝1</td></tr>
<tr><td>BPL</td><td>PLus</td><td>N＝0</td></tr>
<tr><td>BMI</td><td>MInus</td><td>N＝1</td></tr>
<tr><td>BGE</td><td>Greater or Equal</td><td>N⊕V＝0</td></tr>
<tr><td>BLT</td><td>Less Than</td><td>N⊕V＝1</td></tr>
<tr><td>BGT</td><td>Greater Than</td><td>Z∨(N⊕V)=0</td></tr>
<tr><td>BLE</td><td>Less or Equal</td><td>Z∨(N⊕V)=1</td></tr>
</table></td></tr>
<tr><td>ＪＭＰ</td><td>－</td><td>指定されたアドレスへ無条件に分岐します。</td></tr>
<tr><td>ＢＳＲ</td><td>－</td><td>指定されたアドレスへサブルーチン分岐します。</td></tr>
<tr><td>ＪＳＲ</td><td>－</td><td>指定されたアドレスへサブルーチン分岐します。</td></tr>
<tr><td>ＲＴＳ</td><td>－</td><td>サブルーチンから復帰します。</td></tr>
<tr><td rowspan="7">システム制御命令</td><td>ＲＴＥ</td><td>－</td><td>例外処理ルーチンから復帰します。</td></tr>
<tr><td>ＳＬＥＥＰ</td><td>－</td><td>低消費電力状態に遷移します。</td></tr>
<tr><td>ＬＤＣ</td><td>Ｂ</td><td>Ｒｓ ⟶ ＣＣＲ、#IMM ⟶ ＣＣＲ
汎用レジスタの内容、またはイミディエイトデータをＣＣＲに転送します。</td></tr>
<tr><td>ＳＴＣ</td><td>Ｂ</td><td>ＣＣＲ ⟶ Ｒｄ
ＣＣＲの内容を汎用レジスタに転送します。</td></tr>
<tr><td>ＡＮＤＣ</td><td>Ｂ</td><td>ＣＣＲ∧#IMM ⟶ ＣＣＲ
ＣＣＲとイミディエイトデータの論理積をとります。</td></tr>
<tr><td>ＯＲＣ</td><td>Ｂ</td><td>ＣＣＲ∨#IMM ⟶ ＣＣＲ
ＣＣＲとイミディエイトデータの論理和をとります。</td></tr>
<tr><td>ＸＯＲＣ</td><td>Ｂ</td><td>ＣＣＲ⊕#IMM ⟶ ＣＣＲ
ＣＣＲとイミディエイトデータの排他的論理和をとります。</td></tr>
</table>

【注】 * サイズはオペランドサイズを示します。
Ｂ：バイト

表 1.2　命令の機能別一覧(7)

分類	命　令	ｻｲｽﾞ	機　能
システム制御命令	ＮＯＰ	－	ＰＣ＋２ ⟶ ＰＣ ＰＣのインクリメントだけを行います。
ブロック転送命令／ＥＥＰＲＯＭ書込み専用命令	ＥＥＰＭＯＶ	－	if　Ｒ４Ｌ≠０　then Repeat　@Ｒ５＋→@Ｒ６＋，Ｒ４Ｌ－１→Ｒ４Ｌ Until　Ｒ４Ｌ＝０ else next ; ブロック転送命令です。Ｒ５で示されるアドレスから始まり、Ｒ４Ｌで指定されるバイト数のデータを、Ｒ６で示されるアドレスから始まるロケーションへ転送します。転送終了後、次の命令を実行します。 Ｈ８／３００シリーズの大容量ＥＥＰＲＯＭを内蔵したＬＳＩではＥＥＰＲＯＭ書込み専用命令として機能します。詳細は当該ＬＳＩのハードウェアマニュアルを参照してください。

【ビット操作命令使用上の注意】

ＢＳＥＴ、ＢＣＬＲ、ＢＮＯＴ、ＢＳＴ、ＢＩＳＴの各命令は、バイト単位でデータをリードし、ビット操作後に再びバイト単位でデータをライトします。

したがって、ライト専用ビットを含むレジスタ、またはポートに対してこれらの命令を使用する場合には注意が必要です。

動作順序		動　作　内　容
1	リード	指定したアドレスのデータ（バイト単位）をリードします。
2	ビット操作	リードしたデータの指定された１ビットを操作します。
3	ライト	指定したアドレスに、操作したデータ（バイト単位）をライトします。

（例１）ポート４のＤＤＲにＢＣＬＲ命令を実行した例を示します。ポート４にはプルアップＭＯＳが内蔵されているものとします。

Ｐ４$_7$、Ｐ４$_6$は入力端子に設定され、それぞれ“Lowレベル”、“Highレベル”が入力されているとし、Ｐ４$_5$～Ｐ４$_0$は出力端子に設定され、それぞれ“Lowレベル”出力状態とします。

ここで、ＢＣＬＲ命令で、Ｐ４$_0$を入力ポートにする例を示します。

【Ａ；ＢＣＬＲ命令を実行前】

	Ｐ４$_7$	Ｐ４$_6$	Ｐ４$_5$	Ｐ４$_4$	Ｐ４$_3$	Ｐ４$_2$	Ｐ４$_1$	Ｐ４$_0$
入出力	入力	入力	出力	出力	出力	出力	出力	出力
端子状態	Low レベル	High レベル	Low レベル	Low レベル	Low レベル	Low レベル	Low レベル	Low レベル
ＤＤＲ	0	0	1	1	1	1	1	1
ＤＲ	1	0	0	0	0	0	0	0
プルアップＭＯＳ	ＯＮ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ

【Ｂ；ＢＣＬＲ命令を実行】

```
ＢＣＬＲ　　＃０　，@Ｐ４ＤＤＲ
```

ＤＤＲに対してＢＣＬＲ命令を実行します。

【Ｃ；ＢＣＬＲ命令を実行後】

	Ｐ４$_7$	Ｐ４$_6$	Ｐ４$_5$	Ｐ４$_4$	Ｐ４$_3$	Ｐ４$_2$	Ｐ４$_1$	Ｐ４$_0$
入出力	出力	出力	出力	出力	出力	出力	出力	入力
端子状態	Low レベル	High レベル	Low レベル	Low レベル	Low レベル	Low レベル	Low レベル	High レベル
ＤＤＲ	1	1	1	1	1	1	1	0
ＤＲ	1	0	0	0	0	0	0	0
プルアップＭＯＳ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ

【Ｄ；ＢＣＬＲ命令の動作説明】

ＢＣＬＲ命令を実行するとＣＰＵは、最初にＰ４ＤＤＲをリードします。

Ｐ４ＤＤＲはライト専用レジスタですので、ＣＰＵはH'ＦＦをリードします。

したがってこの例では、ＤＤＲはH'３Ｆですが、ＣＰＵがリードしたデータはH'ＦＦとなります。

次に、ＣＰＵはリードしたデータのビット０を“０”にクリアして、データをH'ＦＥに変更します。

最後に、このデータ（H'ＦＥ）をＤＤＲに書き込んでＢＣＬＲ命令を終了します。

その結果、Ｐ４$_0$はＤＤＲが“０”になり、入力ポートになります。

しかし、入力ポートであったビット７、６のＤＤＲが１になって出力ポートに変化してしまいます。

（例２）ポート４にＢＳＥＴ命令を実行した例を示します。

Ｐ４$_7$、Ｐ４$_6$は入力端子に設定され、それぞれ"Lowレベル"、"Highレベル"が入力されているとし、Ｐ４$_5$～Ｐ４$_0$は出力端子に設定され、それぞれ"Lowレベル"出力状態とします。

ここで、ＢＳＥＴ命令で、Ｐ４$_0$に"Highレベル"出力を行う例を示します。

【Ａ；ＢＳＥＴ命令を実行前】

	Ｐ４$_7$	Ｐ４$_6$	Ｐ４$_5$	Ｐ４$_4$	Ｐ４$_3$	Ｐ４$_2$	Ｐ４$_1$	Ｐ４$_0$
入出力	入力	入力	出力	出力	出力	出力	出力	出力
端子状態	Low レベル	High レベル	Low レベル	Low レベル	Low レベル	Low レベル	Low レベル	Low レベル
ＤＤＲ	0	0	1	1	1	1	1	1
ＤＲ	1	0	0	0	0	0	0	0
プルアップＭＯＳ	ＯＮ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ

【Ｂ；ＢＳＥＴ命令を実行】

```
ＢＳＥＴ　　＃０　，　＠ＰＯＲＴ４
```

ポート４に対してＢＳＥＴ命令を実行します。

【Ｃ；ＢＳＥＴ命令を実行後】

	Ｐ４$_7$	Ｐ４$_6$	Ｐ４$_5$	Ｐ４$_4$	Ｐ４$_3$	Ｐ４$_2$	Ｐ４$_1$	Ｐ４$_0$
入出力	入力	入力	出力	出力	出力	出力	出力	出力
端子状態	Low レベル	High レベル	Low レベル	Low レベル	Low レベル	Low レベル	Low レベル	High レベル
ＤＤＲ	0	0	1	1	1	1	1	1
ＤＲ	0	1	0	0	0	0	0	1
プルアップＭＯＳ	ＯＦＦ	ＯＮ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ

【Ｄ；ＢＳＥＴ命令の動作説明】

ＢＳＥＴ命令を実行するとＣＰＵは、最初にポート４をリードします。

Ｐ４$_7$、Ｐ４$_6$は入力端子であるので、ＣＰＵは端子の状態（"Lowレベル"、"Highレベル"入力）をリードします。

Ｐ４$_5$～Ｐ４$_0$は出力端子であるので、ＣＰＵはＤＲの値をリードします。

したがってこの例では、ＤＲはＨ'８０ですが、ＣＰＵがリードしたデータはＨ'４０となります。

次に、ＣＰＵはリードしたデータのビット０を"１"にセットして、データをＨ'４１に変更します。

最後に、この値（Ｈ'４１）をＤＲに書き込んでＢＳＥＴ命令を終了します。

その結果、Ｐ４$_0$はＤＲが"１"になり、"Highレベル"出力になります。

しかし、ビット７、６のＤＲが変化し、内蔵プルアップＭＯＳのＯＮ／ＯＦＦ状態が変化してしまいます。

（例２）についての対策例を以下に示します。

ポート４のＤＲと同じデータをメモリ上のワークエリアに格納し、ワークエリア上のデータに対しビット操作を行った後、このデータをポート４のＤＲにライトしてください。

【Ａ；ＢＳＥＴ命令を実行前】

```
MOV.B   #80,   R0L
MOV.B   R0L,   @RAM0
MOV.B   R0L,   @PORT4
```

ＤＲに書き込む値（H'８０）をあらかじめメモリ上のワークエリア（ＲＡＭ０）とＤＲにライトします。

	$P4_7$	$P4_6$	$P4_5$	$P4_4$	$P4_3$	$P4_2$	$P4_1$	$P4_0$
入出力	入力	入力	出力	出力	出力	出力	出力	出力
端子状態	Low レベル	High レベル	Low レベル	Low レベル	Low レベル	Low レベル	Low レベル	Low レベル
ＤＤＲ	0	0	1	1	1	1	1	1
ＤＲ	1	0	0	0	0	0	0	0
プルアップＭＯＳ	ＯＮ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ
ＲＡＭ０	1	0	0	0	0	0	0	0

【Ｂ；ＢＳＥＴ命令を実行】

```
BSET    #0 ,@RAM0
```

ＤＲのワークエリア（ＲＡＭ０）に対してＢＳＥＴ命令を実行します。

【Ｃ；ＢＳＥＴ命令を実行後】

```
MOV.B   @RAM0, R0L
MOV.B   R0L,   @PORT4
```

ワークエリア（ＲＡＭ０）の値をＤＲにライトします。

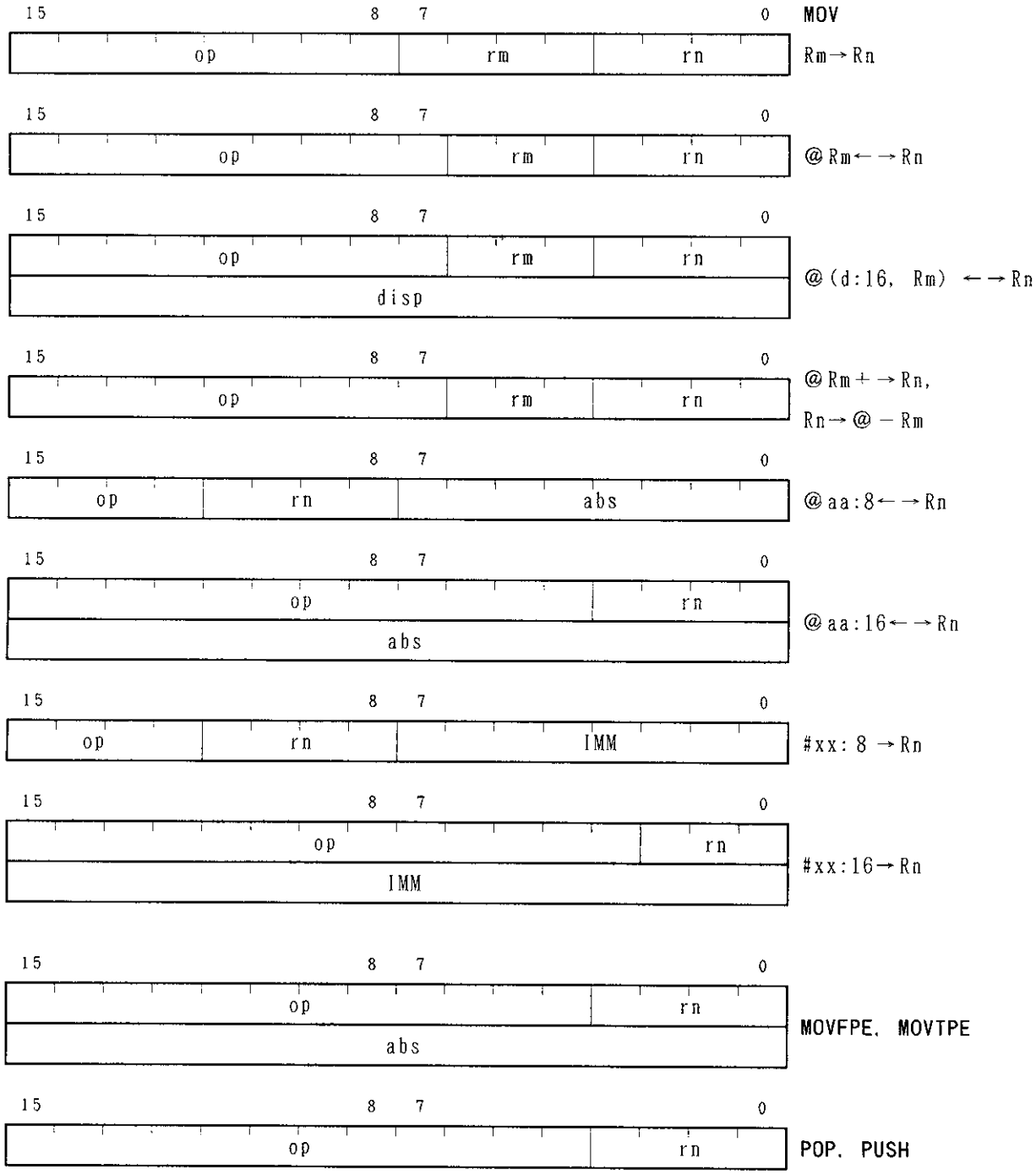
	$P4_7$	$P4_6$	$P4_5$	$P4_4$	$P4_3$	$P4_2$	$P4_1$	$P4_0$
入出力	入力	入力	出力	出力	出力	出力	出力	出力
端子状態	Low レベル	High レベル	Low レベル	Low レベル	Low レベル	Low レベル	Low レベル	High レベル
ＤＤＲ	0	0	1	1	1	1	1	1
ＤＲ	1	0	0	0	0	0	0	1
プルアップＭＯＳ	ＯＮ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ	ＯＦＦ
ＲＡＭ０	1	0	0	0	0	0	0	1

なお、ポートの仕様の詳細については各製品のハードウェアマニュアルを参照して下さい。

1.3.3　命令の基本フォーマット

(1)　データ転送命令の命令フォーマット

データ転送命令の命令フォーマットを図1.5に示します。



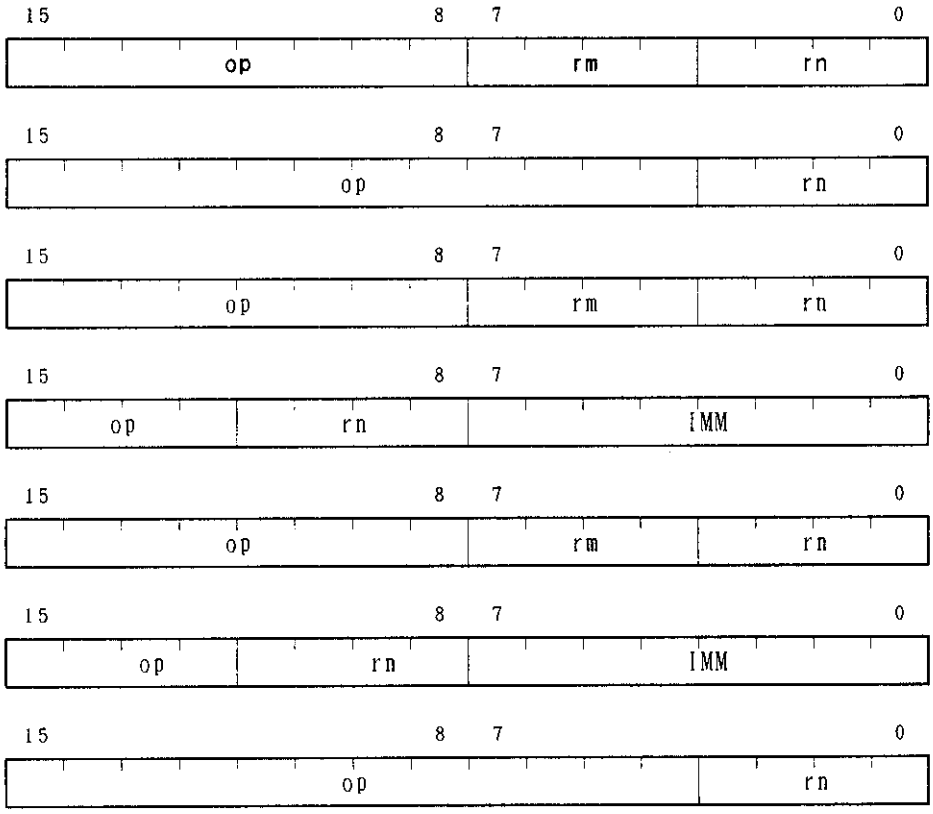
〈記号説明〉

op　　：オペレーションフィールド
rm、rn：レジスタフィールド
disp　：ディスプレースメント
abs　　：絶対アドレス
IMM　　：イミディエイトデータ

図1.5　データ転送命令の命令フォーマット

(2) 算術演算命令・論理演算命令・シフト命令の命令フォーマット

算術演算命令、論理演算命令、およびシフト命令の命令フォーマットを図1.6に示します。



〈記号説明〉

op　　：オペレーションフィールド

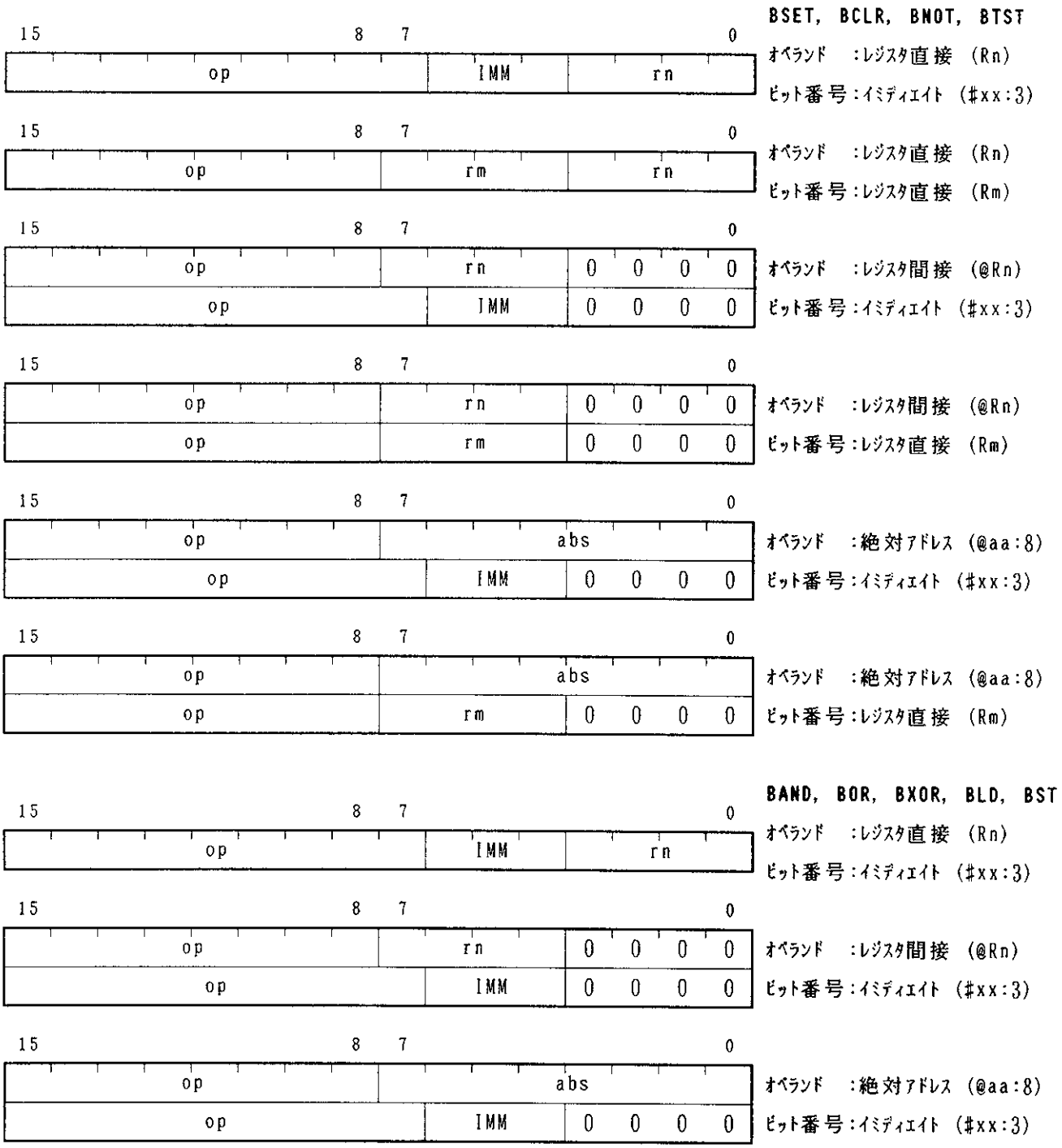
rm、rn：レジスタフィールド

IMM　　：イミディエイトデータ

図1.6　算術演算命令・論理演算命令・シフト命令の命令フォーマット

(3) ビット操作命令のフォーマット

ビット操作命令のフォーマットを図 1.7 に示します。



〈記号説明〉

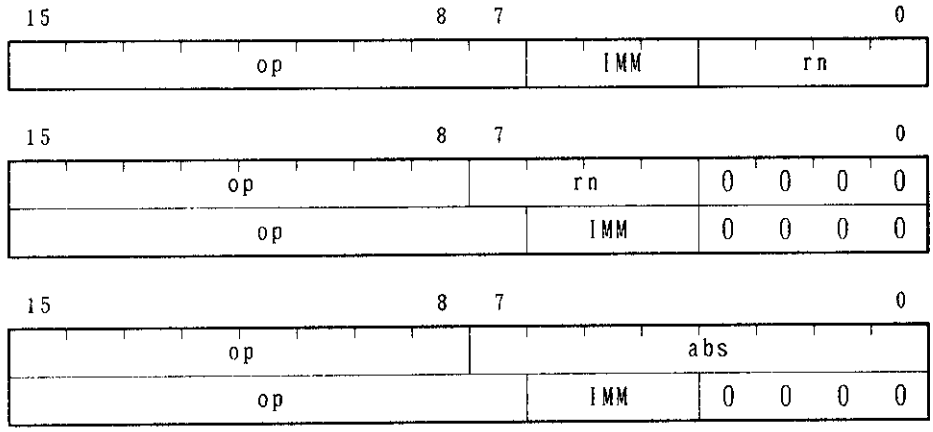
op : オペレーションフィールド

rm、rn : レジスタフィールド

abs : 絶対アドレス

IMM : イミディエイトデータ

図 1.7 ビット操作命令の命令フォーマット(1)



〈記号説明〉

op　　：オペレーションフィールド

rm、rn：レジスタフィールド

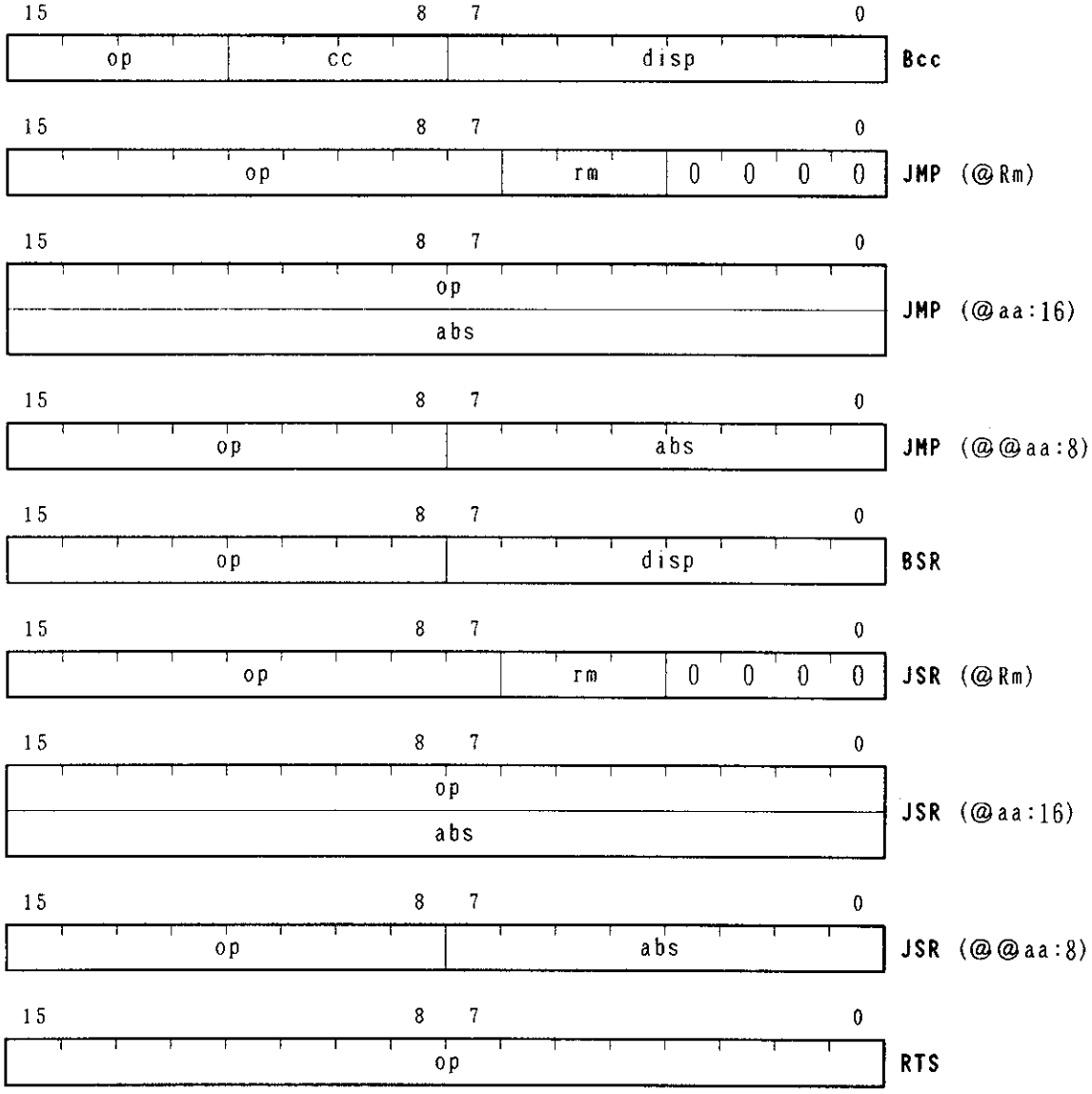
abs　　：絶対アドレス

IMM　　：イミディエイトデータ

図 1.7　ビット操作命令の命令フォーマット(2)

(4) 分岐命令の命令フォーマット

分岐命令の命令フォーマットを図 1.8 に示します。



〈記号説明〉

op　：オペレーションフィールド

cc　：コンディションフィールド

rm　：レジスタフィールド

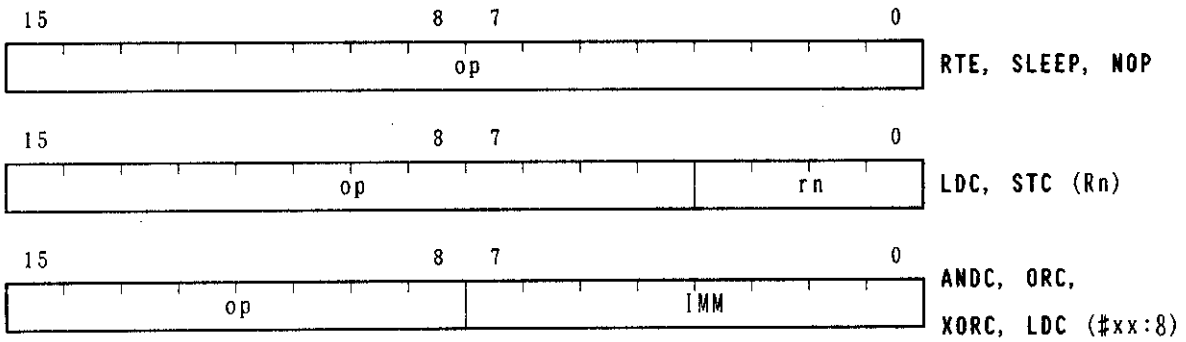
disp：ディスプレースメント

abs　：絶対アドレス

図 1.8　分岐命令の命令フォーマット

(5) システム制御命令の命令フォーマット

システム制御命令の命令フォーマットを図1.9に示します。



〈記号説明〉

op ：オペレーションフィールド

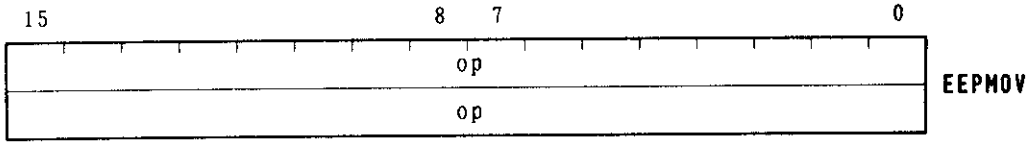
rn ：レジスタフィールド

IMM ：イミディエイトデータ

図1.9 システム制御命令の命令フォーマット

(6) ブロック転送命令／ＥＥＰＲＯＭ書込み専用命令の命令フォーマット

ブロック転送命令／ＥＥＰＲＯＭ書込み専用命令の命令フォーマットを図1.10に示します。



〈記号説明〉

op ：オペレーションフィールド

図1.10 ブロック転送命令／ＥＥＰＲＯＭ書込み専用命令の命令フォーマット

1.3.4　アドレッシングモードと実効アドレスの計算方法

(1)　アドレッシングモード

Ｈ８／３００ＣＰＵは、表1.3に示すように、８種類のアドレッシングモードをサポートしています。命令ごとに、使用できるアドレッシングモードは異なります。

演算命令では①レジスタ直接および⑥イミディエイト（ＡＤＤ.Ｂ、ＡＤＤＸ、ＳＵＢＸ、ＣＭＰ.Ｂ、ＡＮＤ、ＯＲ、ＸＯＲの各命令）が使用されます。

転送命令では、⑦プログラムカウンタ相対と⑧メモリ間接を除くすべてのアドレッシングモードが使用可能です。

また、ビット操作命令では、オペランドの指定に①レジスタ直接、②レジスタ間接および⑤絶対アドレス（８ビット）が使用可能です。さらに、オペランド中のビット番号を指定するために①レジスタ直接（ＢＳＥＴ、ＢＣＬＲ、ＢＮＯＴ、ＢＴＳＴの各命令）および⑥イミディエイト（３ビット）が独立して使用可能です。

表1.3　アドレッシングモード一覧表

No.	アドレッシングモード	記　　号
①	レジスタ直接	Ｒｎ
②	レジスタ間接	@Ｒｎ
③	ディスプレースメント付レジスタ間接	@(d:16, Rn)
④	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	@Ｒｎ＋ @－Ｒｎ
⑤	絶対アドレス	@aa:8／@aa:16
⑥	イミディエイト	#xx:8／#xx:16
⑦	プログラムカウンタ相対	@(d:8, PC)
⑧	メモリ間接	@@aa:8

①　レジスタ直接　Ｒｎ

命令コードのレジスタフィールドで指定されるレジスタ（８ビットまたは16ビット）がオペランドとなります。

16ビットレジスタを使用する命令は、ＭＯＶ.Ｗ、ＡＤＤ.Ｗ、ＳＵＢ.Ｗ、ＣＭＰ.Ｗ、ＡＤＤＳ、ＳＵＢＳ、ＭＵＬＸＵ（８ビット×８ビット）、ＤＩＶＸＵ（16ビット÷８ビット）の各命令です。

②　レジスタ間接　@Ｒｎ

命令コードのレジスタフィールドで指定されるレジスタ（16ビット）の内容をアドレスとしてメモリ上のオペランドを指定します。

③　ディスプレースメント付レジスタ間接　@(d:16, Rn)

命令コードのレジスタフィールドで指定されるレジスタ（16ビット）の内容に、命令コードの第２ワード（第３、第４バイト）の16ビットディスプレースメントを加算した内容をアドレスとして、メモリ上のオペランドを指定します。

本アドレッシングモードは、ＭＯＶ命令のみで使用されます。特に、ＭＯＶ.Ｗ命令では、加算結果が偶数となるようにしてください。

④　ポストインクリメントレジスタ間接　@Ｒｎ＋／プリデクリメントレジスタ間接　@－Ｒｎ

▪ポストインクリメントレジスタ間接　@Ｒｎ＋

ＭＯＶ（Load from)命令で使用されます。

命令コードのレジスタフィールドで指定されるレジスタ（16ビット）の内容をアドレスとして、メモリ上のオペランドを指定します。その後、レジスタの内容に１または２が加算され、加算結果がレジスタに格納されます。ＭＯＶ.Ｂ命令では１、ＭＯＶ.Ｗ命令では２がそれぞれ加算されます。ＭＯＶ.Ｗ命令では、レジスタの内容が偶数となるようにしてください。

▪プリデクリメントレジスタ間接　@－Ｒｎ

ＭＯＶ（Store to）命令で使用されます。

命令コードのレジスタフィールドで指定されるレジスタ（16ビット）の内容から１または２を減算した内容をアドレスとして、メモリ上のオペランドを指定します。その後、減算結果がレジスタに格納されます。ＭＯＶ.Ｂ命令では１、ＭＯＶ.Ｗ命令では２がそれぞれ減算されます。ＭＯＶ.Ｗ命令では、レジスタの内容が偶数となるようにしてください。

⑤　絶対アドレス　@aa:8／@aa:16

命令コード中に含まれる絶対アドレスで、メモリ上のオペランドを指定します。

このとき、絶対アドレスは８ビット（@aa:8）または16ビット（@aa:16)で、８ビット絶対アドレスはＭＯＶ.Ｂ、ビット操作命令で、16ビット絶対アドレスはＭＯＶ.Ｂ、ＭＯＶ.Ｗ、ＪＭＰ、ＪＳＲの各命令で使用されます。

８ビット絶対アドレスの場合、上位８ビットはすべて“１”（H'FF）となります。したがって、アクセス範囲は65280～65535（H'FF00～H'FFFF）番地です。

⑥　イミディエイト　#xx:８／#xx:16

命令コードの第２バイト（#xx:８）または第３、第４バイト（#xx:16）を直接オペランドとして使用します。#xx:16は、ＭＯＶ.Ｗ命令のみで使用されます。

なお、ＡＤＤＳおよびＳＵＢＳ命令では、イミディエイトデータ（１または２）が命令コード中に暗黙的に含まれます。ビット操作命令では、ビット番号を指定するための３ビットのイミディエイトデータが、命令コードの第２または第４バイトに含まれる場合があります。

⑦　プログラムカウンタ相対　@(d:8, PC)

Ｂｃｃ、ＢＳＲの各命令で使用されます。ＰＣの内容に、命令コードの第２バイトの８ビットディスプレースメントを加算して、分岐アドレスを生成します。加算に際して、ディスプレースメントは16ビットに符号拡張され、また加算されるＰＣの内容は次の命令の先頭アドレスとなっていますので、分岐可能範囲は分岐命令に対して－126～＋128バイト（－63～＋64ワード）です。このとき、加算結果が偶数となるようにしてください。

⑧　メモリ間接　@@aa:8

ＪＭＰおよびＪＳＲ命令で使用されます。

命令コードの第２バイトに含まれる８ビット絶対アドレスでメモリ上のオペランドを指定し、この内容を分岐アドレスとして分岐します。この場合、８ビット絶対アドレスの上位８ビットはすべて“０”（H'00）とされますので、分岐アドレスを格納できるのは０～255（H'0000～H'00FF）番地です。

ただし、この内の先頭領域は例外処理ベクタ領域と共通になっていますから注意してください。詳細は、当該ＬＳＩのハードウェアマニュアルを参照してください。

分岐アドレスまたはＭＯＶ.Ｗ命令のオペランドアドレスとして奇数アドレスを指定した場合、最下位ビットは０とみなされ、１番地前から始まるワードデータをアクセスします（「**1.1.2 (2) メモリ上でのデータ構成**」を参照してください）。

(2) 実効アドレスの計算方法

各アドレッシングモードにおける実効アドレス（ＥＡ：Effective Address)の計算法を**表 1.4** に示します。

表 1.4　実効アドレスの計算方法(1)

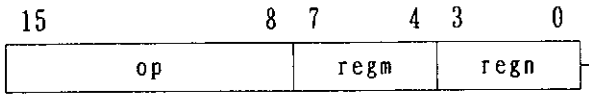
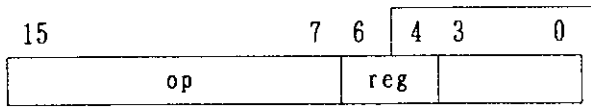
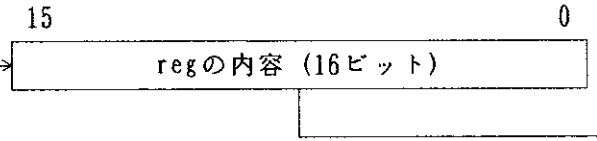
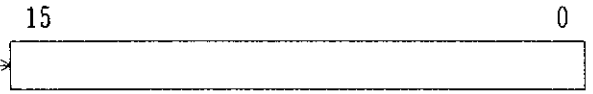
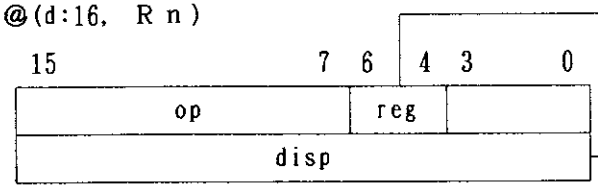
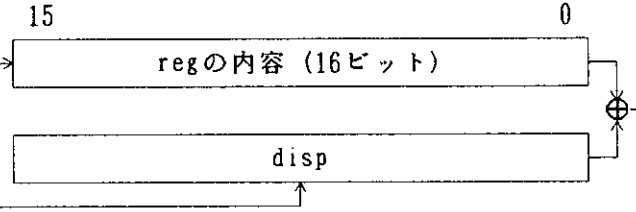
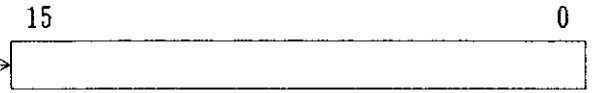
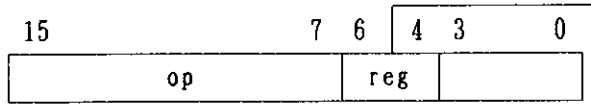
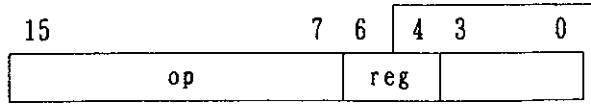
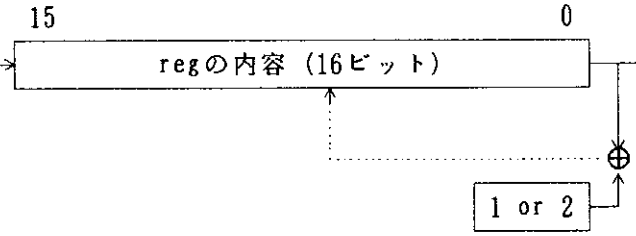
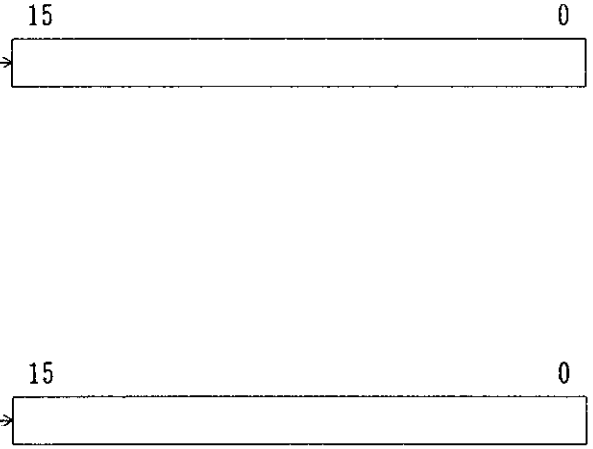
No.	アドレッシングモード・命令フォーマット	実効アドレス計算方法	実効アドレス（ＥＡ）
①	レジスタ直接　Ｒｎ 		 オペランドはregm／nの内容です。
②	レジスタ間接　@Ｒｎ 		
③	ディスプレースメント付レジスタ間接 @(d:16, Ｒｎ) 		
④	ポストインクリメントレジスタ間接／プリデクリメントレジスタ間接 ・ポストインクリメントレジスタ間接　@Ｒｎ＋  ・プリデクリメントレジスタ間接　@－Ｒｎ 	 オペランドサイズがバイトのとき１、 ワードのとき２が加減算されます。	

表 1.4　実効アドレスの計算方法(2)

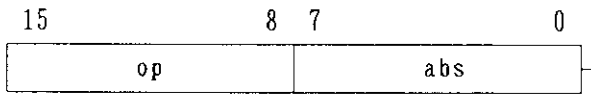
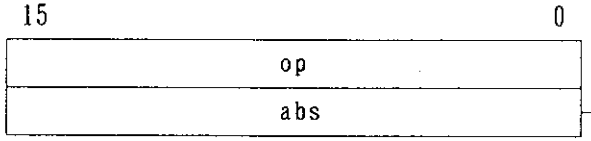
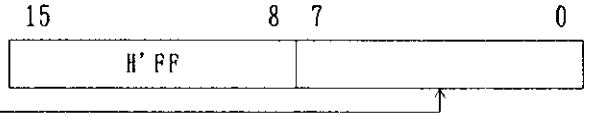
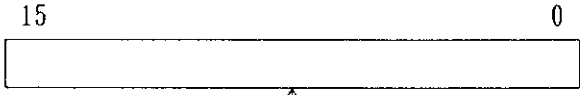
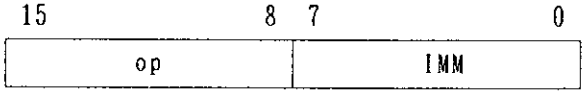
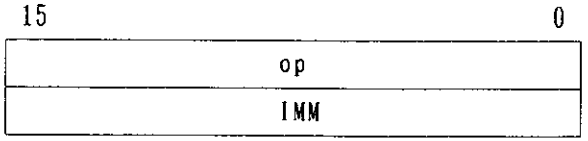
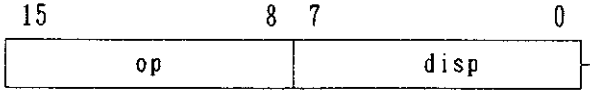
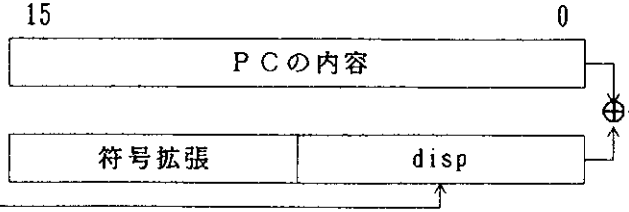
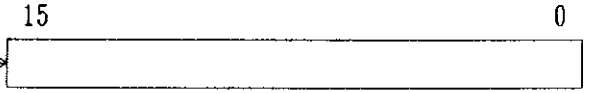
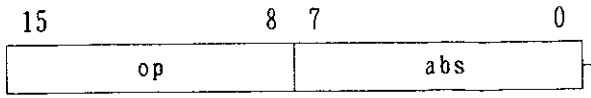
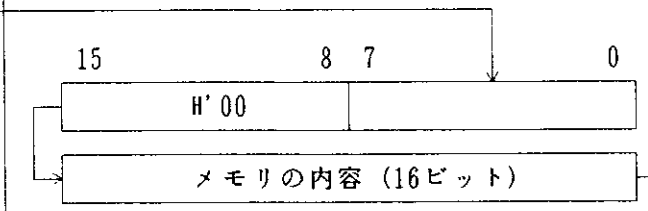
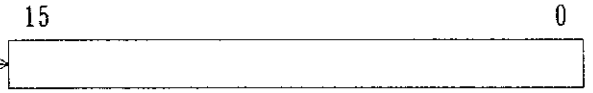
No.	アドレッシングモード・命令フォーマット	実効アドレス計算方法	実効アドレス（ＥＡ）
⑤	絶対アドレス @aa:8  @aa:16  		 
⑥	イミディエイト #xx:8  #xx:16  		オペランドはイミディエイトデータの１または２バイトデータです。
⑦	プログラムカウンタ相対　@(d:8, PC)  	 	

表 1.4　実効アドレスの計算方法(3)

No.	アドレッシングモード・命令フォーマット	実効アドレス計算方法	実効アドレス（ＥＡ）
⑧	メモリ間接@@aa:8 		

〈記号説明〉

reg、regm、regn：汎用レジスタ

op　：オペレーションフィールド

disp：ディスプレースメント

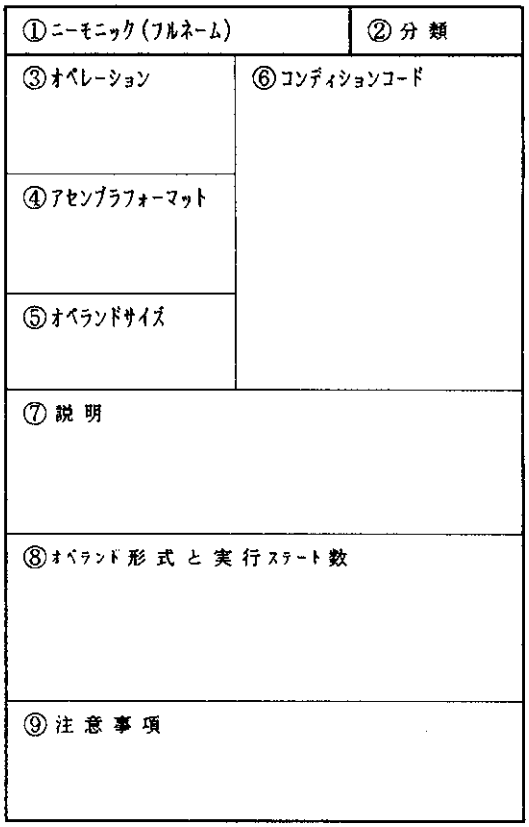
abs　：絶対アドレス

IMM　：イミディエイトデータ

2. 各命令の説明

2.1 表と記号の説明

本書の2.2.1以降の各命令を説明する表の見方について説明します。なお、同一の命令についての説明でも、複数ページにわたっているものがありますから注意してください。



①ニーモニック（フルネーム）

命令のニーモニックとフルネームを示します。

②分類

命令の機能を示します。

③オペレーション

命令の操作を簡潔に示します（2.1.2を参照）。

④アセンブラフォーマット

命令のアセンブラフォーマットを示します（2.1.1を参照）。

⑤オペランドサイズ

使用できるオペランドのサイズを示します。

⑥コンディションコード

命令実行後のコンディションコードレジスタ（ＣＣＲ）の各ビットの変化を示します（2.1.3を参照）。

⑦説明

命令の動作について詳細に説明します。

⑧オペランド形式と実行ステート数

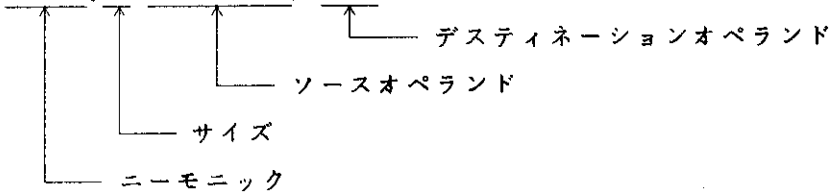
命令のアドレッシングモード、インストラクションフォーマット、ならびに実行ステート数を示します。

⑨注意事項

命令を実行するうえでの注意事項などを示します。

2.1.1　アセンブラフォーマット

例）　ＡＤＤ.Ｂ　＜ＥＡｓ＞，　Ｒｄ



オペランドサイズは、バイト（Ｂ）またはワード（Ｗ）があります。命令によって、使用できるオペランドサイズは制限されています。

＜ＥＡ＞は、複数のアドレッシングモードが使用できることを示します。Ｈ８／３００ＣＰＵがサポートするアドレッシングモードは、次の８種類です。実効アドレスの計算方法については「1.3.4　アドレッシングモードと実効アドレスの計算方法」を参照してください。

記　　号	アドレッシングモード
Ｒｎ	レジスタ直接
@Ｒｎ	レジスタ間接
@(d:16，Ｒｎ)	ディスプレースメント付レジスタ間接
@Ｒｎ＋、@－Ｒｎ	ポストインクリメント／プリデクリメントレジスタ間接
@aa:8/16	絶対アドレス（８／16ビット）
#xx:8/16	イミディエイト（８／16ビット）
@(d:8，ＰＣ)	プログラムカウンタ相対
@@aa:8	メモリ間接

2.1.2 オペレーション

オペレーションの欄で使用されている記号と動作記号を以下に示します。

記　号	内　　容
Ｒｄ	デスティネーション側の汎用レジスタ*
Ｒｓ	ソース側の汎用レジスタ*
Ｒｎ	汎用レジスタ*
（ＥＡｄ）	デスティネーションオペランド
（ＥＡｓ）	ソースオペランド
ＰＣ	プログラムカウンタ
ＳＰ	スタックポインタ
ＣＣＲ	コンディションコードレジスタ
Ｎ	ＣＣＲのＮ（ネガティブ）フラグ
Ｚ	ＣＣＲのＺ（ゼロ）フラグ
Ｖ	ＣＣＲのＶ（オーバフロー）フラグ
Ｃ	ＣＣＲのＣ（キャリ）フラグ
disp	ディスプレースメント
⟶	左辺のオペランドから右辺のオペランドへの転送 または左辺の状態から右辺の状態への遷移
＋	両辺のオペランドを加算
－	左辺のオペランドから右辺のオペランドを減算
×	両辺のオペランドを乗算
÷	左辺のオペランドを右辺のオペランドで除算
∧	両辺のオペランドの論理積
∨	両辺のオペランドの論理和
⊕	両辺のオペランドの排他的論理和
～	反転論理（論理的補数）
（　）<　>	オペランドの実効アドレスの内容

【注】 * 汎用レジスタは、８ビット（Ｒ０Ｈ／Ｒ０Ｌ～Ｒ７Ｈ／Ｒ７Ｌ）または16ビット（Ｒ０～Ｒ７）です。

2.1.3　コンディションコード

コンディションコードの欄で使用されている記号を以下に示します。

記　号	内　　容
↕	実行結果にしたがって変化することを表わします。
*	不確定であることを表わします（値を保証しません）。
0	常に“0”にクリアされることを表わします。
—	実行結果に影響を受けないことを表わします。

2.1.4　インストラクションフォーマット

インストラクションフォーマットの欄で使用されている記号を以下に示します。

記　号	内　　容
IMM	イミディエイトデータ（3、8または16ビット）
abs	絶対アドレス（8または16ビット）
disp	ディスプレースメント（8または16ビット）
rs、rd、rn	レジスタ番号（3または4ビット。rsはオペランドの形式のＲｓ、@Ｒｓなどに対応、rdはＲｄ、@Ｒｄなどに対応、rnはＲｎに対応）

2.1.5　レジスタの指定方法

(1)　アドレスレジスタの指定

rs、rdは、アドレスレジスタとして使用するとき（@Ｒｎ，@(d:16，Ｒｎ)，@Ｒｎ＋，@－Ｒｎ）は16ビットレジスタであり、３ビットで指定されます。

(2)　データレジスタの指定

rs、rd、rnは、データレジスタとして使用するとき（Ｒｎ）、サイズがワードのときは16ビットレジスタであり、３ビットで指定されます。

サイズがバイトのときは８ビットレジスタであり、４ビットで指定されます。このときrs、rd、rnの下位３ビットがレジスタ番号を示し、上位１ビットが“１”のとき下位（Ｌ）が指定され、“０”のとき上位（Ｈ）が指定されます。この対応を以下に示します。

16ビットレジスタ

r			Ｒｎ
0	0	0	Ｒ０
0	0	1	Ｒ１
⋮			⋮
1	1	1	Ｒ７

８ビットレジスタ

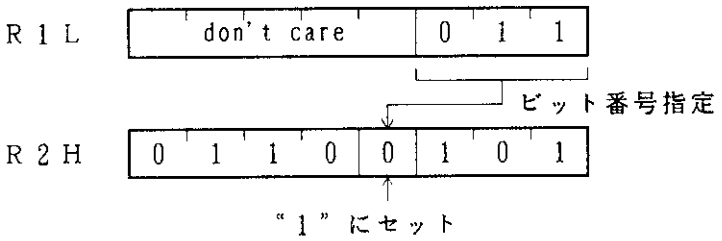
r				Ｒｎ
0	0	0	0	Ｒ０Ｈ
0	0	0	1	Ｒ１Ｈ
⋮				⋮
0	1	1	1	Ｒ７Ｈ
1	0	0	0	Ｒ０Ｌ
1	0	0	1	Ｒ１Ｌ
⋮				⋮
1	1	1	1	Ｒ７Ｌ

2.1.6　ビット操作命令におけるビットデータのアクセス方法

ビットデータは、レジスタまたはメモリ上のオペランドデータ（バイト）の第ｎビット（ｎ＝０、１、２、……、７）という形でアクセスされます。このとき、ビット番号は、３ビットのイミディエイトデータまたは汎用レジスタの内容（下位３ビットのみ有効）によって指定されます。

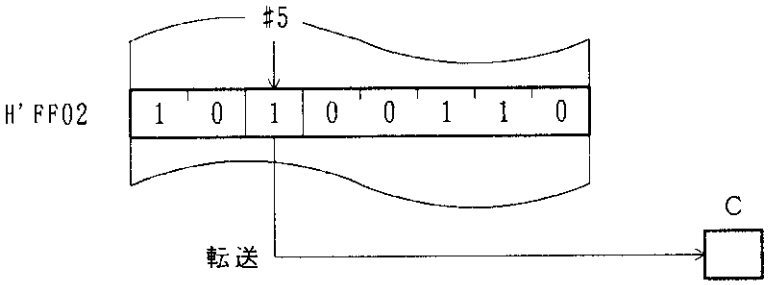
（例１）Ｒ２Ｈのビット３を１にセットする場合

ＢＳＥＴ　Ｒ１Ｌ，Ｒ２Ｈ



（例２）Ｈ'ＦＦ０２番地のビット５をビットアキュムレータに転送する場合

ＢＬＤ　＃５，@Ｈ'ＦＦ０２



なお、オペランドサイズおよびアドレス形式はレジスタまたはメモリ上のオペランドデータについて示しています。

2.1.7　実行ステート数とその補正

実行ステート数の欄に記載されている数値は、命令またはオペランドデータが内蔵メモリに存在する場合の値です。

命令またはオペランドデータが内蔵メモリ以外に存在する場合には、内蔵周辺モジュールや外部デバイスをアクセスするごとに、実行ステート数が増加します。詳細は「2.5　命令実行ステート数」を参照してください。

2.2　各命令の説明

2.2.1 以降に各命令について説明します。

2.2.1 (1)　ＡＤＤ（Ｂ）

ＡＤＤ（ADD binary）　　**２進加算**

《 オペレーション 》

Ｒｄ＋（ＥＡｓ）→Ｒｄ

《 アセンブラフォーマット 》

ＡＤＤ.Ｂ　＜ＥＡｓ＞，Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
–	–	↕	–	↕	↕	↕	↕

Ｉ：実行前の値が保持されます。

Ｈ：ビット３にキャリが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：ビット７にキャリが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）とソースオペランドを加算し、結果を汎用レジスタＲｄに格納します。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト			
イミディエイト	ADD.B	#xx:8，Rd	8	rd	IMM						2
レジスタ直接	ADD.B	Rs，Rd	0	8	rs	rd					2

《 注意事項 》

2.2.1(2)　ＡＤＤ（Ｗ）

ＡＤＤ（ADD binary）　　**２進加算**

《　オペレーション　》

Ｒｄ＋Ｒｓ→Ｒｄ

《　アセンブラフォーマット　》

ＡＤＤ.Ｗ　Ｒｓ，Ｒｄ

《　オペランドサイズ　》

ワード

《　コンディションコード　》

I	H	N	Z	V	C
–	–	↕	–	↕	↕	↕	↕

Ｉ：実行前の値が保持されます。

Ｈ：ビット11にキャリが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：ビット15にキャリが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

《　説　明　》

汎用レジスタＲｄの内容（デスティネーションオペランド）と汎用レジスタＲｓの内容（ソースオペランド）を加算し、結果を汎用レジスタＲｄに格納します。

《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	ADD.W	Rs，Rd	0　9	0　rs　0　rd			2

《　注意事項　》

2.2.2 ＡＤＤＳ

ＡＤＤＳ　(ADD with Sign extention)　｜　**アドレスデータ２進加算**

《 オペレーション 》

Ｒｄ＋１→Ｒｄ
Ｒｄ＋２→Ｒｄ

《 アセンブラフォーマット 》

ＡＤＤＳ　＃１，Ｒｄ
ＡＤＤＳ　＃２，Ｒｄ

《 オペランドサイズ 》

ワード

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説　明 》

汎用レジスタＲｄ（デスティネーションオペランド）にイミディエイトデータの１または２を加算します。
ＡＤＤ命令と異なり、コンディションコードは実行前の値が保持されます。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	ADDS	#1, Rd	0 B	0 0 rd			2
レジスタ直接	ADDS	#2, Rd	0 B	8 0 rd			2

《 注意事項 》

本命令では、バイトサイズのデータは扱えません。

2.2.3　ＡＤＤＸ

ＡＤＤＸ　(ADD with eXtend carry)　　**キャリ付加算**

《　オペレーション　》

Ｒｄ＋（ＥＡｓ）＋Ｃ→Ｒｄ

《　アセンブラフォーマット　》

ＡＤＤＸ　＜ＥＡｓ＞，Ｒｄ

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
－	－	↕	－	↕	↕	↕	↕

Ｉ：実行前の値が保持されます。

Ｈ：ビット３にキャリが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき実行前の値が保持され、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：ビット７にキャリが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

《　説　明　》

汎用レジスタＲｄの内容（デスティネーションオペランド）とソースオペランドとキャリフラグの値を加算し、結果を汎用レジスタＲｄに格納します。

《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト		第４バイト		
イミディエイト	ADDX	#xx:8, Rd	9	rd	IMM						2
レジスタ直接	ADDX	Rs, Rd	0	E	rs	rd					2

《　注意事項　》

2.2.4 ＡＮＤ

ＡＮＤ　(AND logical)　　**論理積**

《 オペレーション 》

Ｒｄ∧（ＥＡｓ）→Ｒｄ

《 アセンブラフォーマット 》

ＡＮＤ　＜ＥＡｓ＞，Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）とソースオペランドの論理積をとり、結果を汎用レジスタＲｄに格納します。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
イミディエイト	AND	#xx:8，Rd	E　rd	IMM			2
レジスタ直接	AND	Rs，Rd	1　6	rs　rd			2

《 注意事項 》

2.2.5 ＡＮＤＣ

ＡＮＤＣ (AND Control register) | **ＣＣＲとの論理積**

《 オペレーション 》

ＣＣＲ∧＃ＩＭＭ→ＣＣＲ

《 コンディションコード 》

I	H	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

Ｉ：実行結果の対応するビットの値が格納されます。

Ｈ：実行結果の対応するビットの値が格納されます。

Ｎ：実行結果の対応するビットの値が格納されます。

Ｚ：実行結果の対応するビットの値が格納されます。

Ｖ：実行結果の対応するビットの値が格納されます。

Ｃ：実行結果の対応するビットの値が格納されます。

《 アセンブラフォーマット 》

ＡＮＤＣ　＃ｘｘ：８，ＣＣＲ

《 オペランドサイズ 》

バイト

《 説　明 》

ＣＣＲの内容とイミディエイトデータの論理積をとり、結果をＣＣＲに格納します。ビット６およびビット４に対しても、他のビットと同様に操作することができます。
なお、本命令の実行終了時点では、ＮＭＩを含めてすべての割込みは受け付けられません。

《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
イミディエイト	ANDC	#xx:8, CCR	0 ⋮ 6	IMM			2

《 注意事項 》

2.2.6 B A N D

B A N D (Bit AND) | **ビット論理積**

《 オペレーション 》

C∧（<ビット番号> of <EAd>）→C

《 アセンブラフォーマット 》

BAND #xx:3, <EAd>

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
−	−	−	−	−	−	−	↕

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：実行前の値が保持されます。

Z：実行前の値が保持されます。

V：実行前の値が保持されます。

C：実行結果が格納されます。

《 説 明 》

デスティネーションオペランドの指定された1ビットとキャリフラグとの論理積をとり、結果をキャリフラグに格納します。
ビット番号は、3ビットのイミディエイトデータで指定されます。デスティネーションの内容は変化しません。

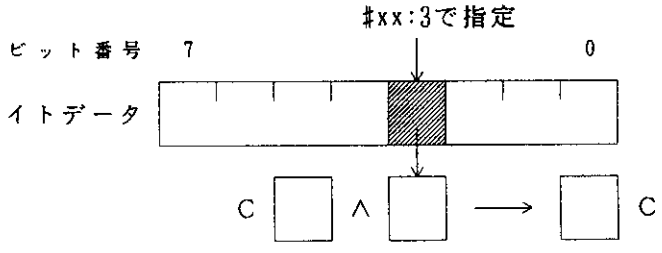
《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット									実行ｽﾃｰﾄ数
			第1バイト		第2バイト			第3バイト		第4バイト		
レジスタ直接	BAND	#xx:3, Rd	7	6	0	IMM	rd					2
レジスタ間接	BAND	#xx:3, @Rd	7	C	0	rd	0	7	6	0	IMM	0	6
絶対アドレス	BAND	#xx:3,@aa:8	7	E	abs			7	6	0	IMM	0	6

【注】 * 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指定<EAd>です。

《 注意事項 》

<EAd> ⟶ レジスタまたはメモリ上のバイトデータ



2.2.7 Ｂｃｃ

Ｂｃｃ (Branch conditionary) | **条件付分岐**

《 オペレーション 》

```
If condition is true, then
   PC + d:8  → PC
else next;
```

《 アセンブラフォーマット 》

Ｂｃｃ　d:8

└→ コンディションフィールド
《 ニーモニックとコンディションフィールド 》参照

《 オペランドサイズ 》

―

《 コンディションコード 》

I	H	N	Z	V	C
–	–	–	–	–	–	–	–

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：実行前の値が保持されます。

Z：実行前の値が保持されます。

V：実行前の値が保持されます。

C：実行前の値が保持されます。

《 説　明 》

ｃｃ（コンディションフィールド）で指定された条件が成立していると、ＰＣにディスプレースメントを加えたアドレスに分岐し、条件が不成立の場合は、次の命令を実行します。

ディスプレースメントは、符号付８ビットデータで、アドレス計算に用いられるＰＣの値は、本命令の直後の命令の先頭アドレスです。したがって、分岐できる範囲は、本命令に対して－126～＋128バイトです。

使用できる条件を以下に示します。

《 ニーモニック と コンディションフィールド 》

ニーモニック	説　明	ｃｃ	条　件	符号条件コードの対応
BRA (BT)	Always (True)	0 0 0 0	Always	
BRN (BF)	Never (False)	0 0 0 1	Never	
BHI	High	0 0 1 0	C ∨ Z = 0	X > Y　符号なし
BLS	Low or Same	0 0 1 1	C ∨ Z = 1	X ≦ Y　符号なし
BCC(BHS)	Carry Clear (High or Same)	0 1 0 0	C = 0	X ≧ Y　符号なし
BCS(BLO)	Carry Set (LOw)	0 1 0 1	C = 1	X < Y　符号なし
BNE	Not Equal	0 1 1 0	Z = 0	X ≠ Y　符号なし・あり
BEQ	EQual	0 1 1 1	Z = 1	X = Y　符号なし・あり
BVC	oVerflow Clear	1 0 0 0	V = 0	
BVS	oVerflow Set	1 0 0 1	V = 1	
BPL	PLus	1 0 1 0	N = 0	
BMI	MInus	1 0 1 1	N = 1	
BGE	Gteater or Equal	1 1 0 0	N ⊕ V = 0	X ≧ Y　符号あり
BLT	Less Than	1 1 0 1	N ⊕ V = 1	X < Y　符号あり
BGT	Greater Than	1 1 1 0	Z ∨ (N ⊕ V) = 0	X > Y　符号あり
BLE	Less or Equal	1 1 1 1	Z ∨ (N ⊕ V) = 1	X ≦ Y　符号あり

Ｂｃｃ（Branch conditionary）　条件付分岐

《　オペランド形式　と　実行ステート数　》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット				実行ステート数
			第１バイト	第２バイト	第３バイト	第４バイト	
プログラムカウンタ相対	BRA（BT）	d:8	4	0	disp		4
プログラムカウンタ相対	BRN（BF）	d:8	4	1	disp		4
プログラムカウンタ相対	BHI	d:8	4	2	disp		4
プログラムカウンタ相対	BLS	d:8	4	3	disp		4
プログラムカウンタ相対	BCC(BHS)	d:8	4	4	disp		4
プログラムカウンタ相対	BCS(BLO)	d:8	4	5	disp		4
プログラムカウンタ相対	BNE	d:8	4	6	disp		4
プログラムカウンタ相対	BEQ	d:8	4	7	disp		4
プログラムカウンタ相対	BVC	d:8	4	8	disp		4
プログラムカウンタ相対	BVS	d:8	4	9	disp		4
プログラムカウンタ相対	BPL	d:8	4	A	disp		4
プログラムカウンタ相対	BMI	d:8	4	B	disp		4
プログラムカウンタ相対	BGE	d:8	4	C	disp		4
プログラムカウンタ相対	BLT	d:8	4	D	disp		4
プログラムカウンタ相対	BGT	d:8	4	E	disp		4
プログラムカウンタ相対	BLE	d:8	4	F	disp		4

《　注意事項　》

1) 分岐先アドレスは、必ず偶数になるようにしてください。
2) ＢＲＡ、ＢＲＮ、ＢＣＣ、ＢＣＳの機械語は、それぞれＢＴ、ＢＦ、ＢＨＳ、ＢＬＯと同一です。
　また、ＢＲＮ（ＢＦ）の実行ステート数はＮＯＰ命令２個と同等です。

2.2.8 ＢＣＬＲ

ＢＣＬＲ (Bit CLeaR) — ビットクリア

《 オペレーション 》

0→（＜ビット番号＞ of ＜ＥＡｄ＞）

《 アセンブラフォーマット 》

ＢＣＬＲ #ｘｘ：３，＜ＥＡｄ＞
ＢＣＬＲ Ｒｎ，＜ＥＡｄ＞

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：実行前の値が保持されます。

Z：実行前の値が保持されます。

V：実行前の値が保持されます。

C：実行前の値が保持されます。

《 説 明 》

デスティネーションオペランドの指定された１ビットを“０”にクリアします。ビット番号は、３ビットのイミディエイトデータまたは汎用レジスタの内容の下位３ビットで指定されます。
指定された１ビットのテストは行いません（コンディションコードは変化しません）。

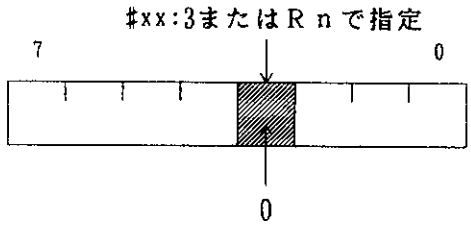
《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	BCLR	#xx:3, Rd	7 2	0 IMM rd			2
レジスタ間接	BCLR	#xx:3, @Rd	7 D	0 rd 0	7 2	0 IMM 0	8
絶対アドレス	BCLR	#xx:3,@aa:8	7 F	abs	7 2	0 IMM 0	8
レジスタ直接	BCLR	Rn, Rd	6 2	rn rd			2
レジスタ間接	BCLR	Rn, @Rd	7 D	0 rd 0	6 2	rn 0	8
絶対アドレス	BCLR	Rn, @aa:8	7 F	abs	6 2	rn 0	8

【注】 * 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指定＜ＥＡｄ＞です。

《 注意事項 》

＜ＥＡｄ＞ ⟶ レジスタまたはメモリ上のバイトデータ



2.2.9 B I A N D

B I A N D (Bit Invert AND) | **ビット論理積**

《 オペレーション 》

C∧〔～(<ビット番号>of<EAd>)〕→C

《 アセンブラフォーマット 》

BIAND #xx:3, <EAd>

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	↕

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：実行前の値が保持されます。

Z：実行前の値が保持されます。

V：実行前の値が保持されます。

C：実行結果が格納されます。

《 説 明 》

デスティネーションオペランドの指定された1ビットを反転し、これとキャリフラグとの論理積をとり、結果をキャリフラグに格納します。ビット番号は、3ビットのイミディエイトデータで指定されます。デスティネーションの内容は変化しません。

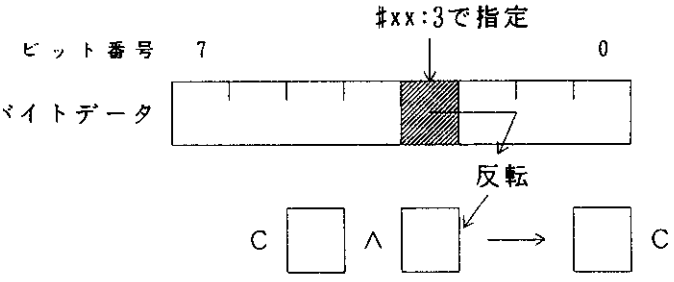
《 オペランド形式 と 実行ステート数 》

アドレッシングモード*	ニーモニック	オペランド形式	インストラクションフォーマット				実行ステート数
			第1バイト	第2バイト	第3バイト	第4バイト	
レジスタ直接	BIAND	#xx:3, Rd	7 6	1 IMM rd			2
レジスタ間接	BIAND	#xx:3, @Rd	7 C	0 rd 0	7 6	1 IMM 0	6
絶対アドレス	BIAND	#xx:3,@aa:8	7 E	abs	7 6	1 IMM 0	6

【注】 * 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指定<EAd>です。

《 注意事項 》

<EAd> ⟶ レジスタまたはメモリ上のバイトデータ



2.2.10 B I L D

B I L D (Bit Invert LoaD) | **ビット転送**

《 オペレーション 》

～（＜ビット番号＞ of ＜ＥＡｄ＞）→Ｃ

《 アセンブラフォーマット 》

ＢＩＬＤ #ｘｘ：３，＜ＥＡｄ＞

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
—	—	—	—	—	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：指定ビットの内容が反転されて格納されます。

《 説 明 》

デスティネーションオペランドの指定された１ビットを反転し、これをキャリフラグに転送します。ビット番号は、３ビットのイミディエイトデータで指定されます。デスティネーションの内容は変化しません。

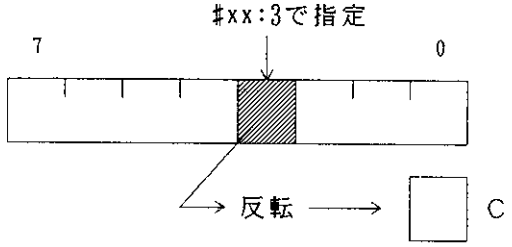
《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	BILD	#xx:3, Rd	7 7	1 IMM rd			2
レジスタ間接	BILD	#xx:3, @Rd	7 C	0 rd 0	7 7	1 IMM 0	6
絶対アドレス	BILD	#xx:3,@aa:8	7 E	abs	7 7	1 IMM 0	6

【注】 * 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指定＜ＥＡｄ＞です。

《 注意事項 》

＜ＥＡｄ＞ ⟶ レジスタまたはメモリ上のバイトデータ



2.2.11 ＢＩＯＲ

ＢＩＯＲ　(Bit Invert inclusive OR)　　**ビット論理和**

《　オペレーション　》

ＣＶ〔～(<ﾋﾞｯﾄ番号>of<ＥＡｄ>)〕→Ｃ

《　アセンブラフォーマット　》

ＢＩＯＲ　#ｘｘ：３，<ＥＡｄ>

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
－	－	－	－	－	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行結果が格納されます。

《　説　明　》

デスティネーションオペランドの指定された１ビットを反転し、これとキャリフラグとの論理和をとり、結果をキャリフラグに格納します。ビット番号は、３ビットのイミディエイトデータで指定されます。デスティネーションの内容は変化しません。

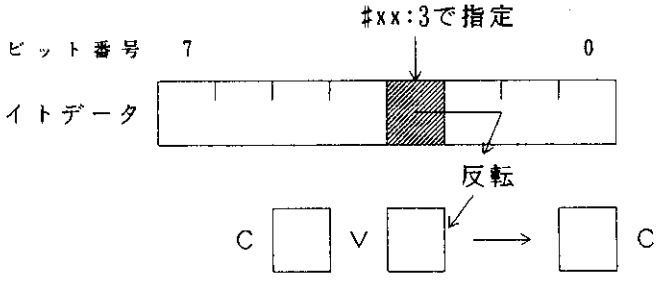
《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	BIOR	#xx:3, Rd	7 4	1 IMM rd			2
レジスタ間接	BIOR	#xx:3, @Rd	7 C	0 rd 0	7 4	1 IMM 0	6
絶対アドレス	BIOR	#xx:3,@aa:8	7 E	abs	7 4	1 IMM 0	6

【注】 * 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指定＜ＥＡｄ＞です。

《　注意事項　》

＜ＥＡｄ＞ ⟶ レジスタまたはメモリ上のバイトデータ



2.2.12 ＢＩＳＴ

ＢＩＳＴ (Bit Invert STore) | **ビット転送**

《 オペレーション 》

～Ｃ→（＜ビット番号＞ of ＜ＥＡｄ＞）

《 アセンブラフォーマット 》

ＢＩＳＴ #ｘｘ：３，＜ＥＡｄ＞

《 オペランドサイズ 》

バイト

《 コンディションコード 》

Ｉ	Ｈ	Ｎ	Ｚ	Ｖ	Ｃ
－	－	－	－	－	－	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説 明 》

デスティネーションオペランドの指定された１ビットのロケーションに、キャリフラグの内容を反転して転送します。ビット番号は、３ビットのイミディエイトデータで指定されます。なお、デスティネーションオペランドの指定されない他のビットの内容は変化しません。

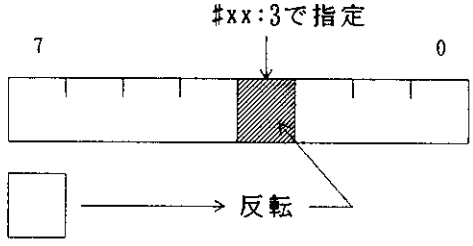
《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット									実行ｽﾃｰﾄ数
			第１バイト		第２バイト			第３バイト		第４バイト		
レジスタ直接	BIST	#xx:3, Rd	6	7	1	IMM	rd					2
レジスタ間接	BIST	#xx:3, @Rd	7	D	0	rd	0	6	7	1	IMM	0	8
絶対アドレス	BIST	#xx:3,@aa:8	7	F	abs			6	7	1	IMM	0	8

【注】 * 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指定＜ＥＡｄ＞です。

《 注意事項 》

＜ＥＡｄ＞ ⟶ レジスタまたはメモリ上のバイトデータ



2.2.13 B I X O R

B I X O R (Bit Invert eXclusive OR)	ビット排他的論理和

《 オペレーション 》

C⊕〔～(<ビット番号>of<EAd>)〕→C

《 アセンブラフォーマット 》

BIXOR #xx:3, <EAd>

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
–	–	–	–	–	–	–	↕

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：実行前の値が保持されます。

Z：実行前の値が保持されます。

V：実行前の値が保持されます。

C：実行結果が格納されます。

《 説 明 》

デスティネーションオペランドの指定された1ビットを反転し、これとキャリフラグとの排他的論理和をとり、結果をキャリフラグに格納します。ビット番号は、3ビットのイミディエイトデータで指定されます。デスティネーションの内容は変化しません。

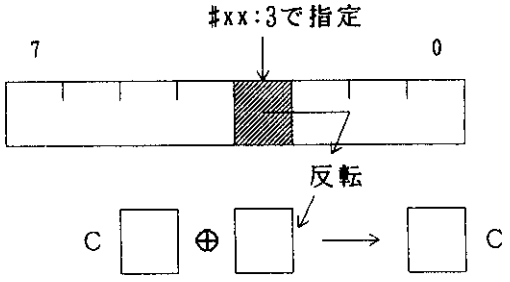
《 オペランド形式 と 実行ステート数 》

アドレッシングモード*	ニーモニック	オペランド形式	インストラクションフォーマット				実行ステート数
			第1バイト	第2バイト	第3バイト	第4バイト	
レジスタ直接	BIXOR	#xx:3, Rd	7 5	1 IMM rd			2
レジスタ間接	BIXOR	#xx:3, @Rd	7 C	0 rd 0	7 5	1 IMM 0	6
絶対アドレス	BIXOR	#xx:3,@aa:8	7 E	abs	7 5	1 IMM 0	6

【注】 * 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指定<EAd>です。

《 注意事項 》

<EAd> ⟶ レジスタまたはメモリ上のバイトデータ



2.2.14 ＢＬＤ

ＢＬＤ　(Bit LoaD)　　**ビット転送**

《　オペレーション　》

（＜ビット番号＞ of ＜ＥＡｄ＞）→Ｃ

《　アセンブラフォーマット　》

ＢＬＤ　＃ｘｘ：３，＜ＥＡｄ＞

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
－	－	－	－	－	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：指定ビットの内容が格納されます。

《　説　明　》

デスティネーションオペランドの指定された１ビットをキャリフラグに転送します。ビット番号は、３ビットのイミディエイトデータで指定されます。デスティネーションの内容は変化しません。

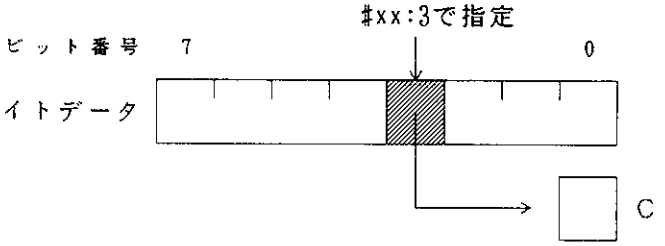
《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ｽﾃｰﾄ数
			第１バイト		第２バイト			第３バイト		第４バイト	
レジスタ直接	BLD	#xx:3, Rd	7	7	0	IMM	rd				2
レジスタ間接	BLD	#xx:3, @Rd	7	C	0	rd	0	7	7	0 IMM 0	6
絶対アドレス	BLD	#xx:3,@aa:8	7	E	abs			7	7	0 IMM 0	6

【注】＊　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指定＜ＥＡｄ＞です。

《　注意事項　》

＜ＥＡｄ＞ ⟶ レジスタまたはメモリ上のバイトデータ



2.2.15 ＢＮＯＴ

ＢＮＯＴ　(Bit NOT)　　**ビット反転**

《 オペレーション 》

～（＜ビット番号＞ of ＜ＥＡｄ＞）
　→（＜ビット番号＞ of ＜ＥＡｄ＞）

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

《 アセンブラフォーマット 》

```
ＢＮＯＴ　＃ｘｘ：３，＜ＥＡｄ＞
ＢＮＯＴ　Ｒｎ，＜ＥＡｄ＞
```

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 オペランドサイズ 》

バイト

《 説　明 》

デスティネーションオペランドの指定された１ビットを反転します。ビット番号は、３ビットのイミディエイトデータまたは汎用レジスタの内容の下位３ビットで指定されます。
指定された１ビットのテストは行いません（コンディションコードは変化しません）。

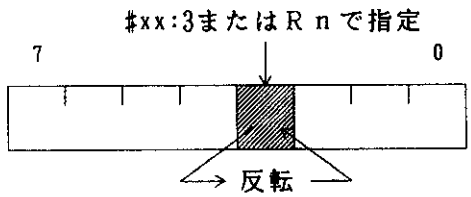
《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	BNOT	#xx:3, Rd	7 1	0 IMM rd			2
レジスタ間接	BNOT	#xx:3, @Rd	7 D	0 rd 0	7 1	0 IMM 0	8
絶対アドレス	BNOT	#xx:3,@aa:8	7 F	abs	7 1	0 IMM 0	8
レジスタ直接	BNOT	Rn, Rd	6 1	rn rd			2
レジスタ間接	BNOT	Rn, @Rd	7 D	0 rd 0	6 1	rn 0	8
絶対アドレス	BNOT	Rn, @aa:8	7 F	abs	6 1	rn 0	8

【注】 * 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指定＜ＥＡｄ＞です。

《 注意事項 》

＜ＥＡｄ＞ ⟶ レジスタまたはメモリ上のバイトデータ



2.2.16 ＢＯＲ

ＢＯＲ　(Bit inclusive OR)　　**ビット論理和**

《 オペレーション 》

ＣＶ（＜ビット番号＞ of ＜ＥＡｄ＞）→Ｃ

《 アセンブラフォーマット 》

ＢＯＲ　＃ｘｘ：３，＜ＥＡｄ＞

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行結果が格納されます。

《 説　明 》

デスティネーションオペランドの指定された１ビットとキャリフラグとの論理和をとり、結果をキャリフラグに格納します。ビット番号は、３ビットのイミディエイトデータで指定されます。デスティネーションの内容は変化しません。

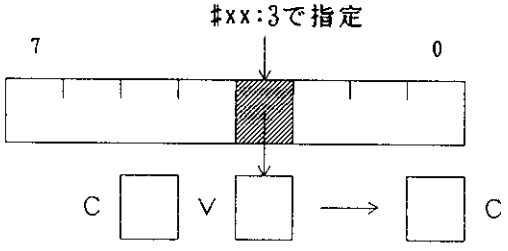
《 オペランド形式　と　実行ステート 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット									実行ｽﾃｰﾄ数
			第１バイト		第２バイト			第３バイト		第４バイト		
レジスタ直接	BOR	#xx:3, Rd	7	4	0	IMM	rd					2
レジスタ間接	BOR	#xx:3, @Rd	7	C	0	rd	0	7	4	0	IMM	0	6
絶対アドレス	BOR	#xx:3,@aa:8	7	E	abs			7	4	0	IMM	0	6

【注】 * 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指定＜ＥＡｄ＞です。

《 注意事項 》

＜ＥＡｄ＞⟶レジスタまたはメモリ上のバイトデータ



2.2.17 ＢＳＥＴ

ＢＳＥＴ (Bit SET) | **ビットセット**

《 オペレーション 》

1→（＜ビット番号＞ of ＜ＥＡｄ＞）

《 アセンブラフォーマット 》

ＢＳＥＴ #ｘｘ：３，＜ＥＡｄ＞
ＢＳＥＴ Ｒｎ，＜ＥＡｄ＞

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説 明 》

デスティネーションオペランドの指定された１ビットを“１”にセットします。ビット番号は、３ビットのイミディエイトデータまたは汎用レジスタの内容の下位３ビットで指定されます。
指定された１ビットのテストは行いません（コンディションコードは変化しません）。

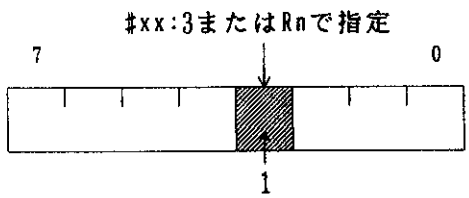
《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット										実行ｽﾃｰﾄ数
			第１バイト		第２バイト			第３バイト		第４バイト			
レジスタ直接	BSET	#xx:3, Rd	7	0	0	IMM	rd						2
レジスタ間接	BSET	#xx:3, @Rd	7	D	0	rd	0	7	0	0	IMM	0	8
絶対アドレス	BSET	#xx:3,@aa:8	7	F	abs			7	0	0	IMM	0	8
レジスタ直接	BSET	Rn, Rd	6	0	rn		rd						2
レジスタ間接	BSET	Rn, @Rd	7	D	0	rd	0	6	0	rn		0	8
絶対アドレス	BSET	Rn, @aa:8	7	F	abs			6	0	rn		0	8

【注】 * 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指定＜ＥＡｄ＞です。

《 注意事項 》

＜ＥＡｄ＞⟶レジスタまたはメモリ上のバイトデータ



2.2.18 ＢＳＲ

ＢＳＲ　(Branch to SubRoutine)　　**サブルーチン分岐**

《 オペレーション 》

ＰＣ→@－ＳＰ

ＰＣ＋ d:8→ＰＣ

《 アセンブラフォーマット 》

ＢＳＲ　d:8

《 オペランドサイズ 》

――

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説　明 》

指定されたアドレスにサブルーチン分岐します。
ＰＣの内容をリスタートアドレスとしてスタックに退避し、ＰＣにディスプレースメントを加えたアドレスに分岐します。スタックに退避されるＰＣの内容は、本命令の直後の命令の先頭アドレスとなっています。また、ディスプレースメントは符号付８ビットデータで、分岐できる範囲は本命令に対して－126～＋128バイトです。

《 オペランド形式　と　実行ステート数 》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット				実行ステート数
			第１バイト	第２バイト	第３バイト	第４バイト	
プログラムカウンタ相対	BSR	d:8	5	5	disp			6

《 注意事項 》

分岐先アドレスは、必ず偶数になるようにしてください。

2.2.19 ＢＳＴ

ＢＳＴ　(Bit STore)　　**ビット転送**

《　オペレーション　》

Ｃ→（＜ビット番号＞ of ＜ＥＡｄ＞）

《　アセンブラフォーマット　》

ＢＳＴ　＃ｘｘ：３，＜ＥＡｄ＞

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《　説　明　》

デスティネーションオペランドの指定された１ビットのロケーションに、キャリフラグの内容を転送します。ビット番号は、３ビットのイミディエイトデータで指定されます。

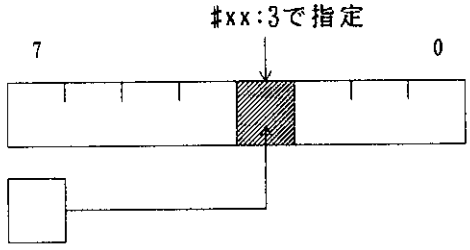
《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ＊	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	BST	#xx:3, Rd	6 7	0 IMM rd			2
レジスタ間接	BST	#xx:3, @Rd	7 D	0 rd 0	6 7	0 IMM 0	8
絶対アドレス	BST	#xx:3,@aa:8	7 F	abs	6 7	0 IMM 0	8

【注】＊　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指定＜ＥＡｄ＞です。

《　注意事項　》

＜ＥＡｄ＞⟶レジスタまたはメモリ上のバイトデータ



2.2.20 ＢＴＳＴ

ＢＴＳＴ (Bit TeST) | **ビットテスト**

《 オペレーション 》

～（＜ビット番号＞ of ＜ＥＡｄ＞）→Ｚ

《 アセンブラフォーマット 》

```
ＢＴＳＴ  #ｘｘ：３，＜ＥＡｄ＞
ＢＴＳＴ  Ｒｎ，＜ＥＡｄ＞
```

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
—	—	—	—	—	↕	—	—

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：指定したビットが０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説 明 》

デスティネーションオペランドの指定された１ビットの状態を調べて、その結果をゼロフラグに反映します。ビット番号は、３ビットのイミディエイトデータまたは汎用レジスタの内容の下位３ビットで指定されます。デスティネーションの内容は変化しません。

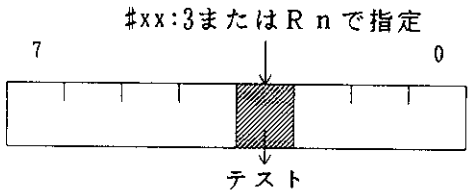
《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	BTST	#xx:3, Rd	7 3	0 IMM rd			2
レジスタ間接	BTST	#xx:3, @Rd	7 C	0 rd 0	7 3	0 IMM 0	6
絶対アドレス	BTST	#xx:3,@aa:8	7 E	abs	7 3	0 IMM 0	6
レジスタ直接	BTST	Rn, Rd	6 3	rn rd			2
レジスタ間接	BTST	Rn, @Rd	7 C	0 rd 0	6 3	rn 0	6
絶対アドレス	BTST	Rn, @aa:8	7 E	abs	6 3	rn 0	6

【注】 * 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指定＜ＥＡｄ＞です。

《 注意事項 》

＜ＥＡｄ＞ ⟶ レジスタまたはメモリ上のバイトデータ



2.2.21 B X O R

B X O R (Bit eXclusive OR) | **ビット排他的論理和**

《 オペレーション 》

C⊕（＜ビット番号＞ of ＜ＥＡｄ＞）→C

《 アセンブラフォーマット 》

ＢＸＯＲ #ｘｘ：３，＜ＥＡｄ＞

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行結果が格納されます。

《 説 明 》

デスティネーションオペランドの指定された１ビットとキャリフラグとの排他的論理和をとり、結果をキャリフラグに格納します。ビット番号は、３ビットのイミディエイトデータで指定されます。デスティネーションの内容は変化しません。

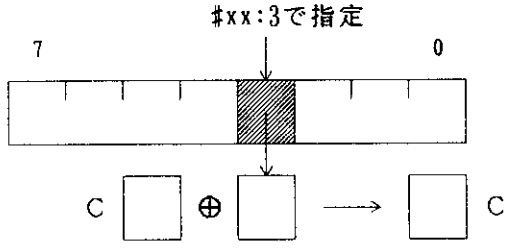
《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ*	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	BXOR	#xx:3, Rd	7 5	0 IMM rd			2
レジスタ間接	BXOR	#xx:3, @Rd	7 C	0 rd 0	7 5	0 IMM 0	6
絶対アドレス	BXOR	#xx:3,@aa:8	7 E	abs	7 5	0 IMM 0	6

【注】 * 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指定＜ＥＡｄ＞です。

《 注意事項 》

＜ＥＡｄ＞ ⟶ レジスタまたはメモリ上のバイトデータ



2.2.22(1)　ＣＭＰ（Ｂ）

ＣＭＰ　(CoMPare)　　**比較**

《 オペレーション 》

Ｒｄ－（ＥＡｓ），ＣＣＲセット

《 アセンブラフォーマット 》

ＣＭＰ.Ｂ　＜ＥＡｓ＞，Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	↕	－	↕	↕	↕	↕

Ｉ：実行前の値が保持されます。

Ｈ：ビット３にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：ビット７にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）からソースオペランドを減算し、その結果にしたがってＣＣＲの各ビットをセットまたはクリアします。デスティネーションの内容は変化しません。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
イミディエイト	CMP.B	#xx:8, Rd	A　rd	IMM			2
レジスタ直接	CMP.B	Rs, Rd	1　C	rs　rd			2

《 注意事項 》

2.2.22(2)　ＣＭＰ（Ｗ）

ＣＭＰ　(CoMPare)　　**比較**

《 オペレーション 》

Ｒｄ－Ｒｓ，ＣＣＲセット

《 アセンブラフォーマット 》

ＣＭＰ.Ｗ　Ｒｓ，Ｒｄ

《 オペランドサイズ 》

ワード

《 コンディションコード 》

I	H		N	Z	V	C
–	–	↕	–	↕	↕	↕	↕

Ｉ：実行前の値が保持されます。

Ｈ：ビット11にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：ビット15にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）から汎用レジスタＲｓの内容（ソースオペランド）を減算し、その結果にしたがってＣＣＲの各ビットをセットまたはクリアします。デスティネーションの内容は変化しません。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	CMP.W	Rs, Rd	1 D	0 rs 0 rd			2

《 注意事項 》

2.2.23 ＤＡＡ

ＤＡＡ　(Decimal Adjust Add)　　**10進補正**

《 オペレーション 》

Ｒｄ（10進補正）→Ｒｄ

《 アセンブラフォーマット 》

ＤＡＡ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
–	*	↕	↕	*	↕

Ｉ：実行前の値が保持されます。

Ｈ：値を保証しません。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：値を保証しません。

Ｃ：ビット７にキャリが発生したとき“１”にセットされ、それ以外のときは実行前の値が保証されます。

《 説　明 》

ＡＤＤ.Ｂ、ＡＤＤＸ命令で、４ビットＢＣＤデータを加算した結果が汎用レジスタの内容（デスティネーションオペランド）およびキャリフラグおよびハーフキャリフラグにあるとき、下表にしたがって汎用レジスタの内容を補正（00、06、60、66を加算）します。

補正前の Ｃフラグ	補正前の 上位４ビット	補正前の Ｈフラグ	補正前の 下位４ビット	加算される数 (16進数)	補正後の Ｃフラグ
0	0～9	0	0～9	00	0
0	0～8	0	A～F	06	0
0	0～9	1	0～3	06	0
0	A～F	0	0～9	60	1
0	9～F	0	A～F	66	1
0	A～F	1	0～3	66	1
1	0～2	0	0～9	60	1
1	0～2	0	A～F	66	1
1	0～3	1	0～3	66	1

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	DAA	Rd	0	F	0	rd			2

《 注意事項 》

上記以外の場合について本命令を実行したときの結果（汎用レジスタの内容、およびＣ，Ｖ，Ｚ，Ｎ，Ｈの各フラグ）は保証しません。

2.2.24 ＤＡＳ

ＤＡＳ　(Decimal Adjust Subtract)　　**10進補正**

《 オペレーション 》

Ｒｄ（10進補正）→Ｒｄ

《 アセンブラフォーマット 》

ＤＡＳ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H		N	Z	V	C
－	－	*	－	↕	↕	*	－

Ｉ：実行前の値が保持されます。

Ｈ：値を保証しません。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：値を保証しません。

Ｃ：実行前の値が保持されます。

《 説　明 》

ＳＵＢ.Ｂ、ＳＵＢＸおよびＮＥＧ命令で、４ビットＢＣＤデータを減算した結果が汎用レジスタ（デスティネーションオペランド）およびキャリフラグおよびハーフキャリフラグにあるとき、下表にしたがって汎用レジスタの内容を補正（00、FA、A0、9Aを加算）します。

補正前の Ｃフラグ	補正前の 上位４ビット	補正前の Ｈフラグ	補正前の 下位４ビット	加算される数 (16進数)	補正後の Ｃフラグ
0	0 ～ 9	0	0 ～ 9	00	0
0	0 ～ 8	1	6 ～ F	FA	0
1	7 ～ F	0	0 ～ 9	A0	1
1	6 ～ F	1	6 ～ F	9A	1

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	DAS	Rd	1	F	0	rd			2

2.2.25 ＤＥＣ

ＤＥＣ　(DECrement)　　**デクリメント**

《 オペレーション 》

Ｒｄ－１→Ｒｄ

《 アセンブラフォーマット 》

ＤＥＣ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	↕	↕	↕	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生したとき（実行前のＲｄの内容がH'80のとき）“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）から“１”を減算し、結果を汎用レジスタＲｄに格納します。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	DEC	Rd	1	A	0	rd			2

《 注意事項 》

2.2.26 ＤＩＶＸＵ

ＤＩＶＸＵ　(DIVide eXtend as Unsigned)　　**除算**

《 オペレーション 》

Ｒｄ÷Ｒｓ→Ｒｄ

《 アセンブラフォーマット 》

ＤＩＶＸＵ　Ｒｓ，Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H			N	Z	V	C
－	－	－	－	↕	↕	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：除数が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：除数が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

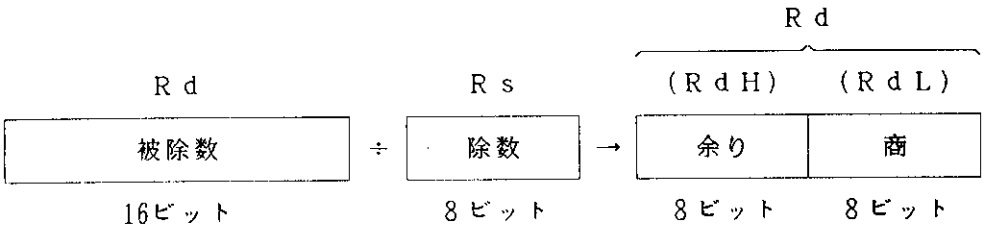
Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）を汎用レジスタＲｓの内容（ソースオペランド）で除算し、結果を汎用レジスタＲｄに格納します。Ｒｄは16ビットレジスタとして、Ｒｓは８ビットレジスタとして指定してください。

演算は、「16ビット÷８ビット→商８ビット 余り８ビット」として行われます。商はＲｄの下位レジスタ（ＲｄＬ）に、余りは上位レジスタ（ＲｄＨ）に格納されます。



なお、ゼロ除算またはオーバフローが発生した場合の結果は保証しません。また、オーバフローについては、次ページの《ＤＩＶＸＵ命令とオーバフロー》を参照してください。

《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ｽﾃｰﾄ数
			第１バイト		第２バイト			第３バイト	第４バイト		
レジスタ直接	DIVXU	Rs, Rd	5	1	rs	0	rd				14

ＤＩＶＸＵ　(DIVide eXtend as Unsigned)

《　ＤＩＶＸＵ命令　と　オーバフロー　》

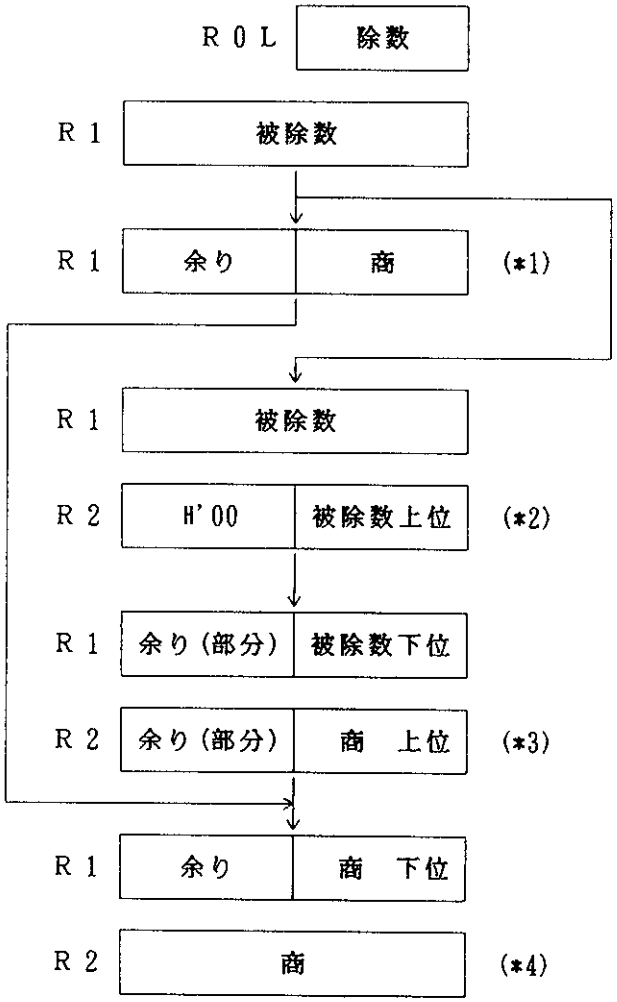
ＤＩＶＸＵ命令は、「16ビット÷８ビット→商８ビット 余り８ビット」という機能仕様になっているため、オーバフローを生じる場合があります。
たとえば、「H'FFFF÷H'01→商H'FFFF 余りH'00」となり、商８ビットを超えてしまいます。
このような場合は、以下のプログラムによって、オーバフローの発生を防ぐことができます。

```
ＤＩＶＸＵ　Ｒ０Ｌ，Ｒ１を行う場合：
        ＭＯＶ.Ｂ   #Ｈ'００，Ｒ２Ｈ
        ＣＭＰ.Ｂ   Ｒ０Ｌ，Ｒ１Ｈ
        ＢＣＣ      Ｌ１
        ＤＩＶＸＵ  Ｒ０Ｌ，Ｒ１    (*1)
        ＭＯＶ.Ｂ   Ｒ１Ｌ，Ｒ２Ｌ
        ＢＲＡ      Ｌ２

Ｌ１     ＭＯＶ.Ｂ   Ｒ１Ｈ，Ｒ２Ｌ  (*2)
        ＤＩＶＸＵ  Ｒ０Ｌ，Ｒ２
        ＭＯＶ.Ｂ   Ｒ２Ｈ，Ｒ１Ｈ  (*3)
        ＤＩＶＸＵ  Ｒ０Ｌ，Ｒ１
        ＭＯＶ.Ｂ   Ｒ２Ｌ，Ｒ２Ｈ
        ＭＯＶ.Ｂ   Ｒ１Ｌ，Ｒ２Ｌ

Ｌ２     ＲＴＳ                    (*4)
```

この結果、商（16ビット）はＲ２に、余り（８ビット）はＲ１Ｈに格納されています。



《　注意事項　》

2.2.27 E E P M O V

E E P M O V (MOVe data to EEPROM) | **ブロック転送**

《 オペレーション 》

```
if  R4L≠0  then
    Repeat  @R5+→@R6+
      R4L−1→R4L
    Until  R4L=0
else next;
```

《 アセンブラフォーマット 》

```
EEPMOV
```

《 オペランドサイズ 》

——

《 コンディションコード 》

I	H			N	Z	V	C
−	−	−	−	−	−	−	−

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：実行前の値が保持されます。

Z：実行前の値が保持されます。

V：実行前の値が保持されます。

C：実行前の値が保持されます。

《 説 明 》

ブロック転送命令です。R5で示されるメモリ上のデータをR6で示されるメモリへ転送し、R5，R6の値をインクリメント、R4Lの値をデクリメントします。R4Lの内容が0となるまで上記動作を繰り返します。その後、次の命令を実行します。データ転送中は割込みの検出を行いません。

本命令の実行終了時には、R4Lは0を、またR5、R6はそれぞれ（最終アドレス＋1）の内容を保持しています。

H8／300シリーズの大容量EEPROMを内蔵したLSIではEEPROM書込み専用命令として機能します。詳細は当該LSIのハードウェアマニュアルを参照してください。

《 オペランド形式 と 実行ステート数 》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット								実行ステート数*
			第1バイト		第2バイト		第3バイト		第4バイト		
————	EEPMOV		7	B	5	C	5	9	8	F	9+4n

＊：R4Lの初期設定値がnの場合です。このとき転送データはnバイトですが、データアクセスは2（n＋1）回行われ、このデータアクセスに必要なステート数は4（n＋1）です。

（n＝0，1，2…255）

《 注意事項 》

本命令ではまず、R5、R6で示されるメモリのリードを行い、その後、データのブロック転送を行います。

2.2.28 ＩＮＣ

ＩＮＣ (INCrement) — インクリメント

《 オペレーション 》

Ｒｄ＋１→Ｒｄ

《 アセンブラフォーマット 》

ＩＮＣ Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	↕	↕	↕	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生（実行前のＲｄの内容がH'7F）したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《 説 明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）に“１”を加算し、結果を汎用レジスタＲｄに格納します。

《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	INC	Rd	0	A	0	rd			2

《 注意事項 》

2.2.29 ＪＭＰ

ＪＭＰ　(JuMP)　　**無条件ジャンプ**

《 オペレーション 》

実効アドレス→ＰＣ

《 アセンブラフォーマット 》

ＪＭＰ　＜ＥＡ＞

《 オペランドサイズ 》

——

《 コンディションコード 》

I	H	N	Z	V	C
–	–	–	–	–	–	–	–

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説　明 》

指定された実効アドレスに無条件分岐します。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ間接	JMP	@Rn	5 9	0 rn 0			4
絶対アドレス	JMP	@aa:16	5 A	0 0	abs		6
メモリ間接	JMP	@@aa:8	5 B	abs			8

《 注意事項 》

分岐先アドレスは必ず偶数になるようにしてください。

2.2.30 ＪＳＲ

ＪＳＲ　(Jump to SubRoutine)　　**サブルーチンジャンプ**

《 オペレーション 》

ＰＣ→@－ＳＰ

実効アドレス→ＰＣ

《 アセンブラフォーマット 》

ＪＳＲ　＜ＥＡ＞

《 オペランドサイズ 》

－

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説　明 》

ＰＣの内容をリスタートアドレスとしてスタックに退避し、指定された実効アドレスに分岐します。退避されるＰＣの値は、本命令の直後の命令の先頭アドレスになります。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ステート数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ間接	JSR	@Rn	5 D	0 rn 0			6
絶対アドレス	JSR	@aa:16	5 E	0 0	abs		8
メモリ間接	JSR	@@aa:8	5 F	abs			8

《 注意事項 》

分岐先アドレスは必ず偶数になるようにしてください。

2.2.31 ＬＤＣ

ＬＤＣ　(LoaD to Control register)　　**ＣＣＲ転送**

《　オペレーション　》

（ＥＡｓ）→ＣＣＲ

《　アセンブラフォーマット　》

ＬＤＣ　＜ＥＡｓ＞，ＣＣＲ

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

Ｉ：ソースオペランドの対応するビットの値が格納されます。

Ｈ：ソースオペランドの対応するビットの値が格納されます。

Ｎ：ソースオペランドの対応するビットの値が格納されます。

Ｚ：ソースオペランドの対応するビットの値が格納されます。

Ｖ：ソースオペランドの対応するビットの値が格納されます。

Ｃ：ソースオペランドの対応するビットの値が格納されます。

《　説　明　》

ソースオペランドをＣＣＲに転送します。ビット６およびビット４に対しても、他のビットと同様に操作することができます。

なお、本命令の実行終了時点では、ＮＭＩを含めてすべての割込みは受け付けられません。

《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
イミディエイト	LDC	#xx:8, CCR	0	7	IMM				2
レジスタ直接	LDC	Rs, CCR	0	3	0	rs			2

《　注意事項　》

2.2.32(1)　ＭＯＶ（Ｂ）

ＭＯＶ　(MOVe data)　　**転送**

《　オペレーション　》

Ｒｓ→Ｒｄ

《　アセンブラフォーマット　》

ＭＯＶ.Ｂ　Ｒｓ，Ｒｄ

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：転送データが負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：転送データが０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《　説　明　》

汎用レジスタＲｓの内容を汎用レジスタＲｄへ転送します。このとき転送するデータを検査し、その結果をＣＣＲに反映します。

《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	MOV.B	Rs，Rd	0	C	rs	rd			2

《　注意事項　》

2.2.32(2)　ＭＯＶ（Ｗ）

ＭＯＶ　(MOVe data)　　**転送**

《　オペレーション　》

Ｒｓ→Ｒｄ

《　アセンブラフォーマット　》

ＭＯＶ.Ｗ　Ｒｓ，Ｒｄ

《　オペランドサイズ　》

ワード

《　コンディションコード　》

I	H			N	Z	V	C
−	−	−	−	↕	↕	0	−

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：転送データが負のとき“１”にセットされ、それ以外にときは“０”にクリアされます。

Ｚ：転送データが０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《　説　明　》

汎用レジスタＲｓの内容を汎用レジスタＲｄへ転送します。このとき転送するデータを検査し、その結果をＣＣＲに反映します。

《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	MOV.W	Rs, Rd	0 D	0 rs 0 rd			2

《　注意事項　》

2.2.32(3)　ＭＯＶ（Ｂ）

ＭＯＶ　(MOVe data)　　**転送**

《　オペレーション　》

（ＥＡｓ）→Ｒｄ

《　アセンブラフォーマット　》

ＭＯＶ.Ｂ　＜ＥＡｓ＞，Ｒｄ

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：転送データが負のとき“１”にセットされ、それ以外にときは“０”にクリアされます。

Ｚ：転送データが０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《　説　明　》

ソースオペランドの内容を汎用レジスタＲｄへ転送します。このとき転送するデータを検査し、その結果をＣＣＲに反映します。

《　オペランド形式　と　実行ステート数　》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット				実行ステート数
			第１バイト	第２バイト	第３バイト	第４バイト	
イミディエイト	MOV.B	#xx:8, Rd	F rd	IMM			2
レジスタ間接	MOV.B	@Rs, Rd	6 8	0 rs rd			4
ディスプレースメント付レジスタ間接	MOV.B	@(d:16,Rs), Rd	6 E	0 rs rd	disp		6
ポストインクリメントレジスタ間接	MOV.B	@Rs+, Rd	6 C	0 rs rd			6
絶対アドレス	MOV.B	@aa:8, Rd	2 rd	abs			4
絶対アドレス	MOV.B	@aa:16, Rd	6 A	0 rd	abs		6

《　注意事項　》

「ＭＯＶ.Ｂ　@Ｒ７＋，Ｒｄ」は、Ｒ７の内容が奇数値となるので使用しないでください。詳細は、「3.2.3　例外処理の動作」またはハードウェアマニュアルを参照してください。

2.2.32(4) ＭＯＶ（Ｗ）

ＭＯＶ　(MOVe data)　　**転送**

《 オペレーション 》

（ＥＡｓ）→Ｒｄ

《 アセンブラフォーマット 》

ＭＯＶ.Ｗ　＜ＥＡｓ＞，Ｒｄ

《 オペランドサイズ 》

ワード

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：転送データが負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：転送データが０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《 説　明 》

ソースオペランドの内容を汎用レジスタＲｄへ転送します。このとき転送するデータを検査し、その結果をＣＣＲに反映します。

《 オペランド形式　と　実行ステート数 》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット				実行ステート数
			第１バイト	第２バイト	第３バイト	第４バイト	
イミディエイト	MOV.W	#xx:16, Rd	7 9	0 0 rd	IMM		4
レジスタ間接	MOV.W	@Rs, Rd	6 9	0 rs 0 rd			4
ディスプレースメント付レジスタ間接	MOV.W	@(d:16,Rs), Rd	6 F	0 rs 0 rd	disp		6
ポストインクリメントレジスタ間接	MOV.W	@Rs+, Rd	6 D	0 rs 0 rd			6
絶対アドレス	MOV.W	@aa:16, Rd	6 B	0 0 rd	abs		6

《 注意事項 》

1) アドレス＜ＥＡｓ＞は必ず偶数になるようにしてください。
2) 「ＭＯＶ.Ｗ　@Ｒ７＋，Ｒｄ」の機械語はＰＯＰ.Ｗ　Ｒｄと同一です。

2.2.32(5)　ＭＯＶ（Ｂ）

ＭＯＶ　(MOVe data)　　**転送**

《　オペレーション　》

Ｒｓ→（ＥＡｄ）

《　アセンブラフォーマット　》

ＭＯＶ.Ｂ　Ｒｓ，＜ＥＡｄ＞

《　オペランドサイズ　》

バイト

《　コンディションコード　》

Ｉ	Ｈ			Ｎ	Ｚ	Ｖ	Ｃ
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：転送データが負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：転送データが０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《　説　明　》

汎用レジスタＲｓの内容（ソースオペランド）をデスティネーションのロケーションへ転送します。このとき転送するデータを検査し、その結果をＣＣＲに反映します。

《　オペランド形式　と　実行ステート数　》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット				実行ステート数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ間接	MOV.B	Rs, @Rd	6 8	1 rd rs			4
ディスプレースメント付レジスタ間接	MOV.B	Rs,@(d:16,Rd)	6 E	1 rd rs	disp		6
プリデクリメントレジスタ間接	MOV.B	Rs, @-Rd	6 C	1 rd rs			6
絶対アドレス	MOV.B	Rs, @aa:8	3 rs	abs			4
絶対アドレス	MOV.B	Rs, @aa:16	6 A	8 rs	abs		6

《　注意事項　》

1)　「ＭＯＶ.Ｂ　Ｒｓ，@－Ｒ７」は、Ｒ７の内容が奇数値となるため使用しないでください。
詳細は、**「3.2.3　例外処理の動作」**またはハードウェアマニュアルを参照してください。

2)　ＭＯＶ.Ｂ　ＲｎＬ，@－ＲｎまたはＭＯＶ.Ｂ　ＲｎＨ，@－Ｒｎを実行すると（実行前のＲｎの内容－１）の下位ＲｎＬまたは上位ＲｎＨが転送されます。

2.2.32(6)　ＭＯＶ（Ｗ）

ＭＯＶ　(MOVe data)　　**転送**

《　オペレーション　》

Ｒｓ→（ＥＡｄ）

《　アセンブラフォーマット　》

ＭＯＶ.Ｗ　Ｒｓ，＜ＥＡｄ＞

《　オペランドサイズ　》

ワード

《　コンディションコード　》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：転送データが負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：転送データが０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《　説　明　》

汎用レジスタＲｓの内容（ソースオペランド）をデスティネーションのロケーションへ転送します。このとき転送するデータを検査し、その結果をＣＣＲに反映します。

《　オペランド形式　と　実行ステート数　》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット				実行ステート数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ間接	MOV.W	Rs, @Rd	6 9	1 rd 0 rs			4
ディスプレースメント付レジスタ間接	MOV.W	Rs,@(d:16,Rd)	6 F	1 rd 0 rs	disp		6
プリデクリメントレジスタ間接	MOV.W	Rs, @-Rd	6 D	1 rd 0 rs			6
絶対アドレス	MOV.W	Rs, @aa:16	6 B	8 0 rs	abs		6

《　注意事項　》

1)　アドレス＜ＥＡｄ＞は必ず偶数になるようにしてください。
2)　「ＭＯＶ.Ｗ　Ｒｓ，@－Ｒ７」の機械語はＰＵＳＨ.Ｗ　Ｒｓと同一です。
3)　ＭＯＶ.Ｗ　Ｒｎ，@－Ｒｎを実行すると（実行前のＲｎの内容－２）が転送されます。

2.2.33 MOVFPE

MOVFPE (MOVe From Peripheral with E clock)　　**E同期データ転送**

《 オペレーション 》

(EAs)` ⟶ Rd
E同期

《 アセンブラフォーマット 》

MOVFPE　@aa:16, Rd

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
−	−	−	−	↕	↕	0	−

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：転送データが負のとき“1”にセットされ、それ以外のときは“0”にクリアされます。

Z：転送データが0（ゼロ）のとき“1”にセットされ、それ以外のときは“0”にクリアされます。

V：常に“0”にクリアされます。

C：実行前の値が保持されます。

《 説 明 》

16ビット絶対アドレスで指定されるメモリの内容を、Eクロックに同期したタイミングで汎用レジスタRdに転送します。このとき転送するデータを検査し、結果をCCRに反映します。

【注】 Eクロック出力端子を備えていない製品およびシングルチップモードでは、本命令を使用しないでください。

《 オペランド形式 と 実行ステート数 》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット								実行ステート数
			第1バイト		第2バイト		第3バイト		第4バイト		
絶対アドレス	MOVFPE	@aa:16, Rd	6	A	4	rd	abs				*

*：本命令の実行ステート数は、最小で13ステートです。

《 注意事項 》

1) 本命令では、上記以外のアドレッシングモードおよびワードサイズのデータは扱えません。
2) 本命令のデータ転送には、9～16ステートを必要とします。ただし、一定ではありません。詳細は、「第4章 基本動作タイミング」を参照してください。

2.2.34 MOVTPE

MOVTPE
(MOVe To Peripheral with E clock)

E同期データ転送

《 オペレーション 》

Rs ⟶ (EAd)
E同期

《 アセンブラフォーマット 》

MOVTPE Rs, @aa:16

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
−	−	−	−	↕	↕	0	−

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：転送データが負のとき“1”にセットされ、それ以外のときは“0”にクリアされます。

Z：転送データが0（ゼロ）のとき“1”にセットされ、それ以外のときは“0”にクリアされます。

V：常に“0”にクリアされます。

C：実行前の値が保持されます。

《 説 明 》

汎用レジスタRsの内容（ソースオペランド）を、Eクロックに同期したタイミングで、16ビット絶対アドレスで指定されるデスティネーションのロケーションに転送します。このとき転送するデータを検査し、結果をCCRに反映します。

【注】 Eクロック出力端子を備えていない製品およびシングルチップモードでは、本命令を使用しないでください。

《 オペランド形式 と 実行ステート数 》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット								実行ステート数
			第1バイト		第2バイト		第3バイト		第4バイト		
絶対アドレス	MOVTPE	Rs, @aa:16	6	A	C	rs	abs				*

＊：本命令の実行ステート数は、最小で13ステートです。

《 注意事項 》

1) 本命令では、上記以外のアドレッシングモードおよびワードサイズのデータは扱えません。
2) 本命令のデータ転送には、9～16ステートを必要とします。ただし、一定ではありません。詳細は、「第4章 基本動作タイミング」を参照してください。

2.2.35 ＭＵＬＸＵ

ＭＵＬＸＵ　(MULtiply eXtend as Unsigned)　　**乗算**

《 オペレーション 》

Ｒｄ×Ｒｓ→Ｒｄ

《 アセンブラフォーマット 》

ＭＵＬＸＵ　Ｒｓ，Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）と汎用レジスタＲｓの内容（ソースオペランド）を乗算し、結果を汎用レジスタＲｄに格納します。Ｒｄは16ビットレジスタ（上位８ビットは無視されます）として、Ｒｓは８ビットレジスタとして指定してください。このとき、ＲｓはＲｄの上位または下位レジスタを指定することも可能です。

演算は、８ビット×８ビット→16ビットで行われます。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	MULXU	Rs，Rd	5　0	rs　0　rd			14

《 注意事項 》

2.2.36 ＮＥＧ

ＮＥＧ　(NEGate)　　**２進符号反転**

《 オペレーション 》

０－Ｒｄ→Ｒｄ

《 アセンブラフォーマット 》

ＮＥＧ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I		H		N	Z	V	C
－	－	↕	－	↕	↕	↕	↕

Ｉ：実行前の値が保持されます。

Ｈ：ビット３にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生した（実行前のＲｄの内容がH'80）とき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：ビット７にボローが発生した（実行前のＲｄの内容がH'00以外）とき“１”にセットされ、それ以外のときは“０”にクリアされます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）の２の補数をとり（H'00から減算し）、結果を汎用レジスタＲｄに格納します。ただし、実行前のＲｄの内容がH'80の場合の結果はH'80となります。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	NEG	Rd	1	7	8	rd			2

《 注意事項 》

2.2.37 ＮＯＰ

ＮＯＰ　(No OPeration)　　**無操作**

《 オペレーション 》

ＰＣ＋２→ＰＣ

《 アセンブラフォーマット 》

ＮＯＰ

《 オペランドサイズ 》

――

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説　明 》

ＰＣのインクリメントのみを行い、次の命令に実行が移ります。ＣＰＵの内部状態には影響を与えません。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト			
――――――	NOP		0	0	0	0					2

《 注意事項 》

2.2.38 ＮＯＴ

ＮＯＴ　(NOT= logical complement)　　**論理反転**

《 オペレーション 》

～Ｒｄ→Ｒｄ

《 アセンブラフォーマット 》

ＮＯＴ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H			N	Z	V	C
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき（実行前のＲｄの内容がH'FFのとき）“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）の１の補数をとり、結果を汎用レジスタＲｄに格納します。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト		第４バイト		
レジスタ直接	NOT	Rd	1	7	0	rd					2

《 注意事項 》

2.2.39 ＯＲ

ＯＲ　(inclusive OR logical)　　**論理和**

《 オペレーション 》

ＲｄＶ（ＥＡｓ）→Ｒｄ

《 アセンブラフォーマット 》

ＯＲ　＜ＥＡｓ＞，Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）と、ソースオペランドの論理和をとり、結果を汎用レジスタＲｄに格納します。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
イミディエイト	OR	#xx:8, Rd	C　rd	IMM			2
レジスタ直接	OR	Rs, Rd	1　4	rs　rd			2

《 注意事項 》

2.2.40 ＯＲＣ

ＯＲＣ　(inclusive OR Control register)	ＣＣＲとの論理和

《 オペレーション 》

ＣＣＲ∨＃ＩＭＭ→ＣＣＲ

《 アセンブラフォーマット 》

ＯＲＣ　＃ｘｘ：８　ＣＣＲ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

Ｉ：実行結果の対応するビットの値が格納されます。

Ｈ：実行結果の対応するビットの値が格納されます。

Ｎ：実行結果の対応するビットの値が格納されます。

Ｚ：実行結果の対応するビットの値が格納されます。

Ｖ：実行結果の対応するビットの値が格納されます。

Ｃ：実行結果の対応するビットの値が格納されます。

《 説　明 》

ＣＣＲの内容とイミディエイトデータの論理和をとり、結果をＣＣＲに格納します。ビット６およびビット４に対しても、他のビットと同様に操作することができます。本命令の実行終了時点では、ＮＭＩを含めてすべての割込みは受け付けられません。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
イミディエイト	ORC	#xx:8, CCR	0　4	IMM			2

《 注意事項 》

2.2.41　ＰＯＰ

ＰＯＰ　(POP data)　　**スタックよりデータ復帰**

《　オペレーション　》

@ＳＰ＋→Ｒｎ

《　アセンブラフォーマット　》

ＰＯＰ　Ｒｎ

《　オペランドサイズ　》

ワード

《　コンディションコード　》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：転送データが負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：転送データが０のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《　説　明　》

スタックから汎用レジスタＲｎへデータを復帰します。このとき復帰するデータを検査し、その結果をＣＣＲに反映します。

《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ｽﾃｰﾄ数
			第１バイト		第２バイト			第３バイト	第４バイト		
――	POP	Rn	6	D	7	0	rn				6

《　注意事項　》

本命令は、ＭＯＶ.Ｗ　@ＳＰ＋，Ｒｎと同一です。

2.2.42 ＰＵＳＨ

ＰＵＳＨ　(PUSH data)　　**スタックへデータ退避**

《 オペレーション 》

Ｒｎ→@－ＳＰ

《 アセンブラフォーマット 》

ＰＵＳＨ　Ｒｎ

《 オペランドサイズ 》

ワード

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：転送データが負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：転送データが０のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《 説　明 》

汎用レジスタＲｎの内容をスタックに退避します。このとき退避するデータを検査し、その結果をＣＣＲに反映します。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
――	PUSH	Rn	6 D	F 0 rn			6

《 注意事項 》

本命令は、ＭＯＶ.Ｗ　Ｒｎ，@－ＳＰと同一です。

2.2.43 ＲＯＴＬ

ＲＯＴＬ　(ROTate Left)　　**ローテート**

《　オペレーション　》

Ｒｄ（左ローテート）→Ｒｄ

《　アセンブラフォーマット　》

ＲＯＴＬ　Ｒｄ

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
−	−	−	−	↕	↕	0	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

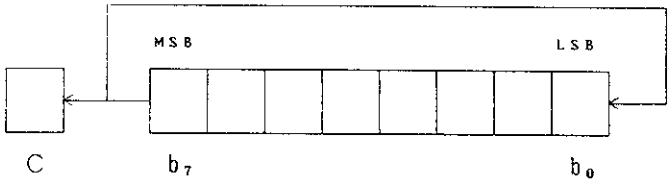
Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前のビット７の値が格納されます。

《　説　明　》

汎用レジスタＲｄの内容（デスティネーションオペランド）のビット群を、左方向に１ビットローテート（回転）します。ローテートしてシフトアウトしたビットは、ビット０に戻り、かつキャリフラグに反映されます。



《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ステート数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	ROTL	Rd	1	2	8	rd			2

《　注意事項　》

2.2.44 ＲＯＴＲ

ＲＯＴＲ　(ROTate Right)	ローテート

《 オペレーション 》

Ｒｄ（右ローテート）→Ｒｄ

《 アセンブラフォーマット 》

ＲＯＴＲ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
–	–	–	–	↕	↕	0	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

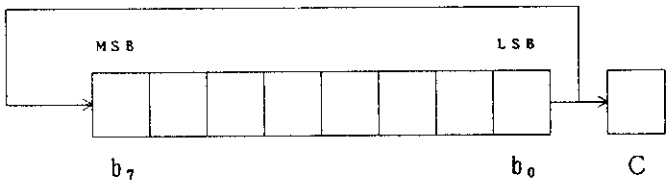
Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前のビット０の値が格納されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）のビット群を、右方向に１ビットローテート（回転）します。ローテートしてシフトアウトしたビットは、ビット７に戻り、かつキャリフラグに反映されます。



《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	ROTR	Rd	1	3	8	rd			2

《 注意事項 》

2.2.45 ＲＯＴＸＬ

ＲＯＴＸＬ　(ROTate with eXtend carry Left)　　**キャリ付ローテート**

《　オペレーション　》

Ｒｄ（キャリ付左ローテート）→Ｒｄ

《　アセンブラフォーマット　》

ＲＯＴＸＬ　Ｒｄ

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

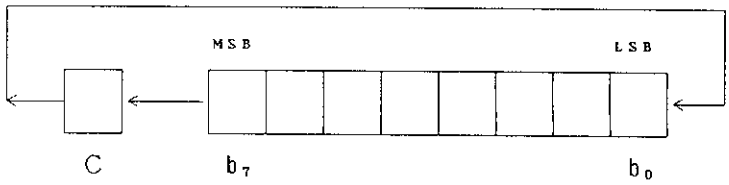
Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前のビット７の値が格納されます。

《　説　明　》

汎用レジスタＲｄの内容（デスティネーションオペランド）のビット群を、キャリフラグを含めて左方向に１ビットローテート（回転）します。ビット０にはキャリフラグの値が入り、ローテートしてシフトアウトしたビットはキャリフラグに格納されます。



《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ステート数
			第１バイト		第２バイト		第３バイト	第４バイト			
レジスタ直接	ROTXL	Rd	1	2	0	rd					2

《　注意事項　》

2.2.46 ＲＯＴＸＲ

ＲＯＴＸＲ　(ROTate with eXtend carry Right)	キャリ付ローテート

《 オペレーション 》

Ｒｄ（キャリ付右ローテート）→Ｒｄ

《 アセンブラフォーマット 》

ＲＯＴＸＲ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

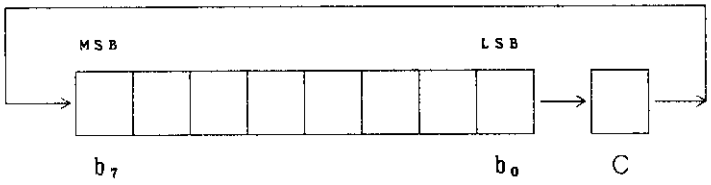
Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前のビット０の値が格納されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）のビット群を、キャリフラグを含めて右方向に１ビットローテート（回転）します。ビット７にはキャリフラグの値が入り、ローテートしてシフトアウトしたビットはキャリフラグに格納されます。



《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	ROTXR	Rd	1	3	0	rd			2

《 注意事項 》

2.2.47 ＲＴＥ

ＲＴＥ　(ReTurn from Exception)　　**例外処理からのリターン**

《 オペレーション 》

@ＳＰ＋→ＣＣＲ
@ＳＰ＋→ＰＣ

《 コンディションコード 》

I	H	N	Z	V	C
↕	↕	↕	↕	↕	↕

I：スタックの内容の対応するビットの値が格納されます。

H：スタックの内容の対応するビットの値が格納されます。

N：スタックの内容の対応するビットの値が格納されます。

Z：スタックの内容の対応するビットの値が格納されます。

V：スタックの内容の対応するビットの値が格納されます。

C：スタックの内容の対応するビットの値が格納されます。

《 アセンブラフォーマット 》

ＲＴＥ

《 オペランドサイズ 》

——

《 説　明 》

例外処理から復帰します。スタックからＣＣＲとＰＣを復帰し、復帰したＰＣが示すアドレスから処理を行います。本命令を実行する直前のＣＣＲおよびＰＣの内容は失われます。

なお、ＣＣＲはバイトサイズですが、スタックからの復帰はワードサイズ（下位８ビットは無視）で行われます。したがって、本命令によってＳＰの内容は＋４されます。

《 オペランド形式　と　実行ステート数 》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット						実行ステート数
			第１バイト		第２バイト		第３バイト	第４バイト	
————	RTE		5	6	7	0			10

《 注意事項 》

2.2.48 ＲＴＳ

ＲＴＳ　(ReTurn from Subroutine)　　**サブルーチンリターン**

《 オペレーション 》

@ＳＰ＋→ＰＣ

《 アセンブラフォーマット 》

ＲＴＳ

《 オペランドサイズ 》

——

《 コンディションコード 》

I	H	N	Z	V	C
–	–	–	–	–	–	–	–

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：実行前の値が保持されます。

Z：実行前の値が保持されます。

V：実行前の値が保持されます。

C：実行前の値が保持されます。

《 説　明 》

サブルーチンから復帰します。スタックからＰＣを復帰し、復帰したＰＣが示すアドレスから処理を行います。本命令を実行する直前のＰＣの内容は失われます。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
————	RTS		5	4	7	0			8

《 注意事項 》

2.2.49 ＳＨＡＬ

ＳＨＡＬ　(SHift Arithmetic Left)　　**算術シフト**

《 オペレーション 》

Ｒｄ（左算術シフト）→Ｒｄ

《 アセンブラフォーマット 》

ＳＨＡＬ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	↕	↕	↕	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

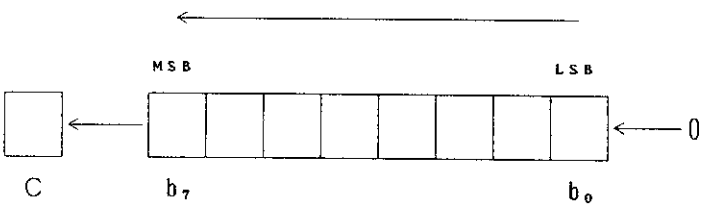
Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：実行前のビット７の値が格納されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）のビット群を、左方向へ算術的に１ビットシフトします。シフトアウトしたビットはキャリフラグに格納され、ビット０には０（ゼロ）が格納されます。



《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	SHAL	Rd	1	0	8	rd			2

《 注意事項 》

本命令とＳＨＬＬ命令とでは、オーバフローフラグの動作が異なります。

2.2.50 ＳＨＡＲ

ＳＨＡＲ　(SHift Arithmetic Right)　　**算術シフト**

《 オペレーション 》

Ｒｄ（右算術シフト）→Ｒｄ

《 アセンブラフォーマット 》

ＳＨＡＲ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	↕

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

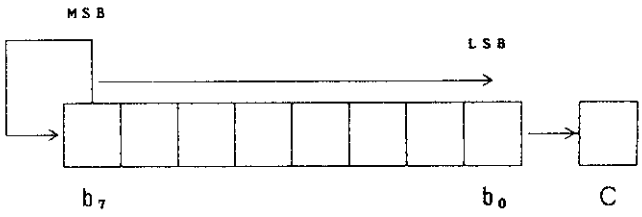
Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前のビット０の値が格納されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）のビット群を、右方向へ算術的に１ビットシフトします。シフトアウトしたビットはＣフラグに格納され、ビット７にはシフト処理前のビット７がセットされます。ビット７は変化しないので、符号変化は起りません。



《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	SHAR	Rd	1 1	8 rd			2

《 注意事項 》

2.2.51 ＳＨＬＬ

ＳＨＬＬ　(SHift Logical Left)　　**論理シフト**

《 オペレーション 》

Ｒｄ（左論理シフト）→Ｒｄ

《 アセンブラフォーマット 》

ＳＨＬＬ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
–	–	–	–	↕	↕	0	↕

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

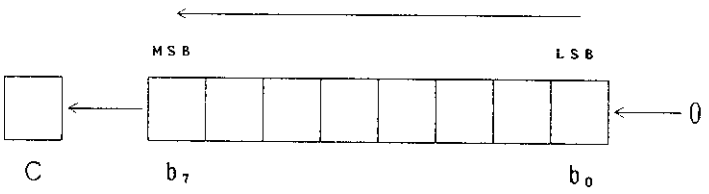
Z：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

V：常に“０”にクリアされます。

C：実行前のビット７の値が格納されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）のビット群を、左方向へ１ビットシフトします。シフトアウトしたビットはキャリフラグに格納され、ビット０には０（ゼロ）が格納されます。



《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ステート数
			第１バイト		第２バイト		第３バイト	第４バイト			
レジスタ直接	SHLL	Rd	1	0	0	rd					2

《 注意事項 》

本命令とＳＨＡＬ命令とでは、オーバフローフラグの動作が異なります。

2.2.52 ＳＨＬＲ

ＳＨＬＲ　(SHift Logical Right)　　**論理シフト**

《 オペレーション 》

Ｒｄ（右論理シフト）→Ｒｄ

《 アセンブラフォーマット 》

ＳＨＬＲ　Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
–	–	–	–	0	↕	0	↕

I：実行前の値が保持されます。

H：実行前の値が保持されます。

N：常に“０”にクリアされます。

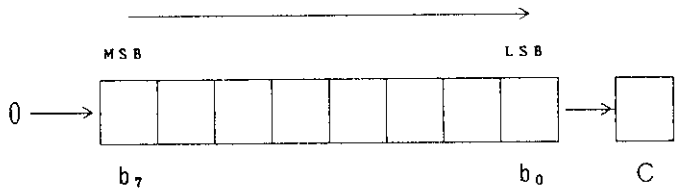
Z：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

V：常に“０”にクリアされます。

C：実行前のビット０の値が格納されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）のビット群を、右方向へ１ビットシフトします。シフトアウトしたビットはキャリフラグに格納され、ビット７には０（ゼロ）が格納されます。



《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	SHLR	Rd	1	1	0	rd			2

《 注意事項 》

2.2.53 ＳＬＥＥＰ

ＳＬＥＥＰ　(SLEEP)	低消費電力状態命令

《 オペレーション 》

プログラム実行状態→低消費電力状態

《 アセンブラフォーマット 》

ＳＬＥＥＰ

《 オペランドサイズ 》

——

《 コンディションコード 》

I	H	N	Z	V	C
–	–	–	–	–	–	–	–

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説　明 》

ＳＬＥＥＰ命令を実行すると、ＣＰＵは低消費電力状態に入ります。低消費電力状態では、ＣＰＵの内部状態は保持され、命令の実行を停止し、例外処理要求の発生を待ち続けます。例外処理要求が発生すると、低消費電力状態は解除され、ＣＰＵは例外処理を開始します。このときＮＭＩ以外の割込み要求では、ＣＰＵ側で割込みがマスクされている（Ｉ＝１）場合、低消費電力状態は解除されません。

《 オペランド形式　と　実行ステート数 》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット						実行ステート数
			第１バイト		第２バイト		第３バイト	第４バイト	
————	SLEEP		0	1	8	0			2

《 注意事項 》

低消費電力状態については、当該ＬＳＩのハードウェアマニュアルを参照してください。

2.2.54 ＳＴＣ

ＳＴＣ (STore from Control register) | **ＣＣＲ転送**

《 オペレーション 》

ＣＣＲ→Ｒｄ

《 アセンブラフォーマット 》

ＳＴＣ ＣＣＲ，Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説 明 》

ＣＣＲの内容を汎用レジスタＲｄに転送します。ビット６およびビット４についても他のビットと同様に扱うことができます。

《 オペランド形式 と 実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
レジスタ直接	STC	CCR，Rd	0	2	0	rd			2

《 注意事項 》

2.2.55(1)　ＳＵＢ（Ｂ）

ＳＵＢ　(SUBtract binary)　　**２進減算**

《　オペレーション　》

Ｒｄ－Ｒｓ→Ｒｄ

《　アセンブラフォーマット　》

ＳＵＢ.Ｂ　Ｒｓ，Ｒｄ

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
–	–	↕	–	↕	↕	↕	↕

Ｉ：実行前の値が保持されます。

Ｈ：ビット３にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：ビット７にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

《　説　明　》

汎用レジスタＲｄの内容（デスティネーションオペランド）から汎用レジスタＲｓの内容（ソースオペランド）を減算し、結果をＲｄに格納します。

《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	SUB.B	Rs，Rd	1　8	rs　rd			2

《　注意事項　》

本命令は汎用レジスタ間の減算のみ可能ですが、汎用レジスタの内容とイミディエイトデータの減算はＳＵＢＸ.Ｂ命令を使用することにより実現できます。この場合、「ＳＵＢＸ.Ｂ　＃ｘｘ：８，Ｒｄ」を実行する前に、Ｚフラグを“１”にセットし、Ｃフラグを“０”にクリアしてください。また、イミディエイト値＃ＩＭＭ≠０の場合、次のプログラム例も使用できます。

(1)　ＯＲＣ　＃Ｈ'０５，ＣＣＲ
　　ＳＵＢＸ　＃（ＩＭＭ－１），Ｒｄ

(2)　ＡＤＤ　＃（０－ＩＭＭ），Ｒｄ
　　ＸＯＲＣ　＃Ｈ'０１，ＣＣＲ

2.2.55(2)　ＳＵＢ（Ｗ）

ＳＵＢ　(SUBtract binary)　　**２進減算**

《　オペレーション　》

Ｒｄ－Ｒｓ→Ｒｄ

《　アセンブラフォーマット　》

ＳＵＢ.Ｗ　Ｒｓ，Ｒｄ

《　オペランドサイズ　》

ワード

《　コンディションコード　》

I		H		N	Z	V	C
–	–	↕	–	↕	↕	↕	↕

Ｉ：実行前の値が保持されます。

Ｈ：ビット11にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：ビット15にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

《　説　明　》

汎用レジスタＲｄの内容（デスティネーションオペランド）から汎用レジスタＲｓの内容（ソースオペランド）を減算し、結果をＲｄに格納します。

《　オペランド形式　と　実行ステート数　》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット				実行ｽﾃｰﾄ数
			第１バイト	第２バイト	第３バイト	第４バイト	
レジスタ直接	SUB.W	Rs，Rd	1　9	0　rs　0　rd			2

《　注意事項　》

2.2.56 ＳＵＢＳ

ＳＵＢＳ　(SUBtract with Sign extention)	アドレスデータ２進減算

《 オペレーション 》

Ｒｄ－１→Ｒｄ
Ｒｄ－２→Ｒｄ

《 アセンブラフォーマット 》

ＳＵＢＳ　＃１，Ｒｄ
ＳＵＢＳ　＃２，Ｒｄ

《 オペランドサイズ 》

ワード

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	－	－	－	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行前の値が保持されます。

Ｚ：実行前の値が保持されます。

Ｖ：実行前の値が保持されます。

Ｃ：実行前の値が保持されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）からイミディエイトの１または２を減算します。
ＳＵＢ命令と異なり、コンディションコードは実行前の値が保持されます。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ｽﾃｰﾄ数
			第１バイト		第２バイト			第３バイト	第４バイト		
レジスタ直接	SUBS	#1, Rd	1	B	0	0	rd				2
レジスタ直接	SUBS	#2, Rd	1	B	8	0	rd				2

《 注意事項 》

本命令では、バイトサイズのデータは扱えません。

2.2.57 ＳＵＢＸ

ＳＵＢＸ　(SUBtract with eXtend carry)　　**キャリ付減算**

《 オペレーション 》

Ｒｄ－（ＥＡｓ）－Ｃ→Ｒｄ

《 アセンブラフォーマット 》

ＳＵＢＸ　＜ＥＡｓ＞，Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H		N	Z	V	C
–	–	↕	–	↕	↕	↕	↕

Ｉ：実行前の値が保持されます。

Ｈ：ビット３にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき実行前の値が保持されます。それ以外のときは“０”にクリアされます。

Ｖ：オーバフローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｃ：ビット７にボローが発生したとき“１”にセットされ、それ以外のときは“０”にクリアされます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）からソースオペランドとキャリフラグの値を減算し、結果をＲｄに格納します。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット								実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト			
イミディエイト	SUBX	#xx:8，Rd	B	rd	IMM						2
レジスタ直接	SUBX	Rs，Rd	1	E	rs	rd					2

《 注意事項 》

2.2.58 ＸＯＲ

ＸＯＲ　(eXclusive OR logical)　　**排他的論理和**

《 オペレーション 》

Ｒｄ⊕（ＥＡｓ）→Ｒｄ

《 アセンブラフォーマット 》

ＸＯＲ　＜ＥＡｓ＞，Ｒｄ

《 オペランドサイズ 》

バイト

《 コンディションコード 》

I	H	N	Z	V	C
－	－	－	－	↕	↕	0	－

Ｉ：実行前の値が保持されます。

Ｈ：実行前の値が保持されます。

Ｎ：実行結果が負のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｚ：実行結果が０（ゼロ）のとき“１”にセットされ、それ以外のときは“０”にクリアされます。

Ｖ：常に“０”にクリアされます。

Ｃ：実行前の値が保持されます。

《 説　明 》

汎用レジスタＲｄの内容（デスティネーションオペランド）とソースオペランドの排他的論理和をとり、結果をＲｄに格納します。

《 オペランド形式　と　実行ステート数 》

ｱﾄﾞﾚｯｼﾝｸﾞﾓｰﾄﾞ	ﾆｰﾓﾆｯｸ	ｵﾍﾟﾗﾝﾄﾞ形式	インストラクションフォーマット						実行ｽﾃｰﾄ数
			第１バイト		第２バイト		第３バイト	第４バイト	
イミディエイト	XOR	#xx:8, Rd	D	rd	IMM				2
レジスタ直接	XOR	Rs, Rd	1	5	rs	rd			2

《 注意事項 》

2.2.59 ＸＯＲＣ

ＸＯＲＣ　(eXclusive OR Control register)	ＣＣＲとの排他的論理和

《　オペレーション　》

ＣＣＲ⊕＃ＩＭＭ→ＣＣＲ

《　アセンブラフォーマット　》

ＸＯＲＣ　＃ｘｘ：８，ＣＣＲ

《　オペランドサイズ　》

バイト

《　コンディションコード　》

I	H	N	Z	V	C
↕	↕	↕	↕	↕	↕

Ｉ：実行結果の対応するビットの値が格納されます。

Ｈ：実行結果の対応するビットの値が格納されます。

Ｎ：実行結果の対応するビットの値が格納されます。

Ｚ：実行結果の対応するビットの値が格納されます。

Ｖ：実行結果の対応するビットの値が格納されます。

Ｃ：実行結果の対応するビットの値が格納されます。

《　説　明　》

ＣＣＲの内容とイミディエイトデータとの排他的論理和をとり、結果をＣＣＲに格納します。ビット６およびビット４に対しても、他のビットと同様に操作することができます。本命令の実行終了時点では、ＮＭＩを含めてすべての割込みは受け付けられません。

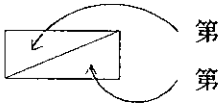
《　オペランド形式　と　実行ステート数　》

アドレッシングモード	ニーモニック	オペランド形式	インストラクションフォーマット				実行ステート数
			第１バイト	第２バイト	第３バイト	第４バイト	
イミディエイト	XORC	#xx:8, CCR	0　5	IMM			2

《　注意事項　》

2.3 オペレーションコードマップ

表2.1にオペレーションコードマップを示します。表2.1では、命令コードの第1バイト（第1ワードのビット15～8）についてのみ示しています。



第2バイトの最上位ビット（命令コードの第1ワードのビット7）が0の場合を示します。
第2バイトの最上位ビット（命令コードの第1ワードのビット7）が1の場合を示します。

表2.1 オペレーションコードマップ

HI＼LO	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	SLEEP	STC	LDC	ORC	XORC	ANDC	LDC	ADD		INC	ADDS	MOV		ADDX	DAA
1	SHLL / SHAL	SHLR / SHAR	ROTXL / ROTL	ROTXR / ROTR	OR	XOR	AND	NOT / NEG	SUB		DEC	SUBS	CMP		SUBX	DAS
2	MOV															
3																
4	BRA	BRN	BHI	BLS	BCC	BCS	BNE	BEQ	BVC	BVS	BPL	BMI	BGE	BLT	BGT	BLE
5	MULXU	DIVXU			RTS	BSR	RTE			JMP				JSR		
6	BSET	BNOT	BCLR	BTST				BST / BIST	MOV*							
7					BOR / BIOR	BXOR / BIXOR	BAND / BIAND	BLD / BILD		MOV		EEPMOV	ビット操作命令			
8	ADD															
9	ADDX															
A	CMP															
B	SUBX															
C	OR															
D	XOR															
E	AND															
F	MOV															

【注】 * MOVFPEおよびMOVTPEの命令コードの第1バイト、および第2バイトの最上位ビット（第1ワードのビット15～7）は、MOV 命令と共通です。
PUSH, POP命令の機械語はMOV命令と同一です。詳細は、「2.2 各命令の説明」を参照してください。

2.4 命令セット一覧表

表2.2 命令セット一覧(1)

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

表2.2　命令セット一覧(2)

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

表2.2　命令セット一覧(3)

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

表2.2　命令セット一覧(4)

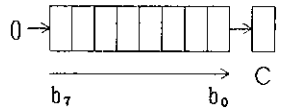
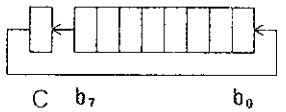
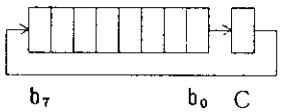
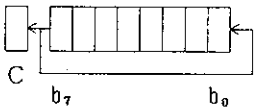
ニーモニック		サイズ	アドレッシングモード／命令長（バイト）									オペレーション	コンディションコード						実行ステート数*
			#xx:8/16	Rn	@Rn	@(d:16, Rn)	@-Rn/@Rn+	@aa:8/16	@(d:8, PC)	@@aa	—		I	H	N	Z	V	C	
SHAR	SHAR.B Rd	B		2									—	—	↕	↕	0	↕	2
SHLL	SHLL.B Rd	B		2									—	—	↕	↕	0	↕	2
SHLR	SHLR.B Rd	B		2									—	—	0	↕	0	↕	2
ROTXL	ROTXL.B Rd	B	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

表2.2　命令セット一覧(5)

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

表2.2　命令セット一覧(6)

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

表2.2　命令セット一覧(7)

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

表2.2　命令セット一覧(8)

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

表2.2　命令セット一覧(9)

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

【注】　＊：実行ステート数は、オペコードおよびオペランドデータが内蔵メモリに存在する場合の値です。それ以外の場合は、「2.5　命令実行ステート数」を参照してください。
①：ビット11から桁上がりまたはビット11へ桁下がりが発生したとき“1”にセットされ、それ以外のとき“0”にクリアされます。
②：演算結果がゼロのとき、演算前の値を保持し、それ以外のとき“0”にクリアされます。
③：補正結果に桁上がりが発生したとき“1”にセットされ、それ以外のとき演算前の値を保持します。
④：実行ステート数は、R4Lの設定値がnのとき4n+9となります。
⑤：Eクロック同期転送命令の実行ステート数は一定ではありません。
⑥：除数が負のとき“1”にセットされ、それ以外のとき“0”にクリアされます。
⑦：除数がゼロのとき“1”にセットされ、それ以外のとき“0”にクリアされます。

2.5　命令実行ステート数

Ｈ８／３００ＣＰＵの各命令についての実行状態と実行ステート数の計算方法を示します。表 2.4 に命令の実行状態として、命令実行中に行われる命令フェッチ、データリード／ライト等のサイクル数を示し、表 2.3 に各々のサイクルに必要なステート数を示します。命令の実行ステート数は次の計算式で計算されます。

$$実行ステート数 = I \cdot S_I + J \cdot S_J + K \cdot S_K + L \cdot S_L + M \cdot S_M + N \cdot S_N$$

■　実行ステート数計算例

（例）モード１、スタック領域を外部空間に設定、外部デバイスアクセス時１ウェイト挿入とした場合

１．ＢＳＥＴ　＃０，@FFC7

表 2.4 より

$I = L = 2$，　$J = K = M = N = 0$

表 2.3 より

$S_I = 8$，$S_L = 3$

実行ステート数 $= 2 \times 8 + 2 \times 3 = 22$

２．ＪＳＲ　@@30

表 2.4 より

$I = 2$，　$J = K = 1$，$L = M = N = 0$

表 2.3 より

$S_I = S_J = S_K = 8$

実行ステート数 $= 2 \times 8 + 1 \times 8 + 1 \times 8 = 32$

表 2.3　実行状態（サイクル）に要するステート数

実行状態（サイクル）		アクセス対象		
		内蔵メモリ	内蔵周辺ﾓｼﾞｭｰﾙ	外部デバイス
命令フェッチ	S_I	2	6	6 + 2m
分岐アドレスリード	S_J			
スタック操作	S_K			
バイトデータアクセス	S_L		3	3 + m*
ワードデータアクセス	S_M		6	6 + 2m
内部動作	S_N	1		

＜記号説明＞　m：外部デバイスアクセス時のウェイトステート数

【注】* MOVFPE,MOVTPE では９～16となります（「4.3　Eクロックインタフェース」を参照してください）。

表 2.4　命令の実行状態（サイクル数）(1)

命　令	ニーモニック	命令フェッチ	分岐アドレス リード	スタック操作	バイトデータ アクセス	ワードデータ アクセス	内部動作
		I	J	K	L	M	N
ADD	ADD.B #xx:8, Rd	1					
	ADD.B Rs, Rd	1					
	ADD.W Rs, Rd	1					
ADDS	ADDS.W #1/2, Rd	1					
ADDX	ADDX.B #xx:8, Rd	1					
	ADDX.B Rs, Rd	1					
AND	AND.B #xx:8, Rd	1					
	AND.B Rs, Rd	1					
ANDC	ANDC #xx:8, CCR	1					
BAND	BAND #xx:3, Rd	1					
	BAND #xx:3, @Rd	2			1		
	BAND #xx:3, @aa:8	2			1		
Bcc	BRA d:8 (BT d:8)	2					
	BRN d:8 (BF d:8)	2					
	BHI d:8	2					
	BLS d:8	2					
	BCC d:8 (BHS d:8)	2					
	BCS d:8 (BLO d:8)	2					
	BNE d:8	2					
	BEQ d:8	2					
	BVC d:8	2					
	BVS d:8	2					
	BPL d:8	2					
	BMI d:8	2					
	BGE d:8	2					
	BLT d:8	2					
	BGT d:8	2					
	BLE d:8	2					
BCLR	BCLR #xx:3, Rd	1					
	BCLR #xx:3, @Rd	2			2		
	BCLR #xx:3, @aa:8	2			2		
	BCLR Rn, Rd	1					
	BCLR Rn, @Rd	2			2		
	BCLR Rn, @aa:8	2			2		
BIAND	BIAND #xx:3, Rd	1					
	BIAND #xx:3, @Rd	2			1		
	BIAND #xx:3, @aa:8	2			1		

表 2.4　命令の実行状態（サイクル数）(2)

命　令	ニーモニック	命令フェッチ	分岐アドレスリード	スタック操作	バイトデータアクセス	ワードデータアクセス	内部動作
		I	J	K	L	M	N
BILD	BILD #xx:3, Rd	1					
	BILD #xx:3, @Rd	2			1		
	BILD #xx:3, @aa:8	2			1		
BIOR	BIOR #xx:3, Rd	1					
	BIOR #xx:3, @Rd	2			1		
	BIOR #xx:3, @aa:8	2			1		
BIST	BIST #xx:3, Rd	1					
	BIST #xx:3, @Rd	2			2		
	BIST #xx:3, @aa:8	2			2		
BIXOR	BIXOR #xx:3, Rd	1					
	BIXOR #xx:3, @Rd	2			1		
	BIXOR #xx:3, @aa:8	2			1		
BLD	BLD #xx:3, Rd	1					
	BLD #xx:3, @Rd	2			1		
	BLD #xx:3, @aa:8	2			1		
BNOT	BNOT #xx:3, Rd	1					
	BNOT #xx:3, @Rd	2			2		
	BNOT #xx:3, @aa:8	2			2		
	BNOT Rn, Rd	1					
	BNOT Rn, @Rd	2			2		
	BNOT Rn, @aa:8	2			2		
BOR	BOR #xx:3, Rd	1					
	BOR #xx:3, @Rd	2			1		
	BOR #xx:3, @aa:8	2			1		
BSET	BSET #xx:3, Rd	1					
	BSET #xx:3, @Rd	2			2		
	BSET #xx:3, @aa:8	2			2		
	BSET Rn, Rd	1					
	BSET Rn, @Rd	2			2		
	BSET Rn, @aa:8	2			2		
BSR	BSR d:8	2		1			
BST	BST #xx:3, Rd	1					
	BST #xx:3, @Rd	2			2		
	BST #xx:3, @aa:8	2			2		

表 2.4　命令の実行状態（サイクル数）(3)

命　令	ニーモニック	命令フェッチ	分岐アドレスリード	スタック操作	バイトデータアクセス	ワードデータアクセス	内部動作
		I	J	K	L	M	N
BTST	BTST #xx:3, Rd	1					
	BTST #xx:3, @Rd	2			1		
	BTST #xx:3, @aa:8	2			1		
	BTST Rn, Rd	1					
	BTST Rn, @Rd	2			1		
	BTST Rn, @aa:8	2			1		
BXOR	BXOR #xx:3, Rd	1					
	BXOR #xx:3, @Rd	2			1		
	BXOR #xx:3, @aa:8	2			1		
CMP	CMP.B #xx:8, Rd	1					
	CMP.B Rs, Rd	1					
	CMP.W Rs, Rd	1					
DAA	DAA.B Rd	1					
DAS	DAS.B Rd	1					
DEC	DEC.B Rd	1					
DIVXU	DIVXU.B Rs, Rd	1					12
EEPMOV	EEPMOV	2			2n+2*1		1
INC	INC.B Rd	1					
JMP	JMP @Rn	2					
	JMP @aa:16	2					2
	JMP @@aa:8	2	1				2
JSR	JSR @Rn	2		1			
	JSR @aa:16	2		1			2
	JSR @@aa:8	2	1	1			
LDC	LDC #xx:8, CCR	1					
	LDC Rs, CCR	1					
MOV	MOV.B #xx:8, Rd	1					
	MOV.B Rs, Rd	1					
	MOV.B @Rs, Rd	1			1		
	MOV.B @(d:16,Rs), Rd	2			1		
	MOV.B @Rs+, Rd	1			1		2
	MOV.B @aa:8, Rd	1			1		
	MOV.B @aa:16, Rd	2			1		
	MOV.B Rs, @Rd	1			1		
	MOV.B Rs, @(d:16,Rd)	2			1		
	MOV.B Rs, @-Rd	1			1		2

表 2.4　命令の実行状態（サイクル数）(4)

命　令	ニーモニック	命令フェッチ	分岐アドレス リード	スタック操作	バイトデータ アクセス	ワードデータ アクセス	内部動作
		I	J	K	L	M	N
MOV	MOV.B Rs, @aa:8	1			1		
	MOV.B Rs, @aa:16	2			1		
	MOV.W #xx:16, Rd	2					
	MOV.W Rs, Rd	1					
	MOV.W @Rs, Rd	1				1	
	MOV.W @(d:16,Rs), Rd	2				1	
	MOV.W @Rs+, Rd	1				1	2
	MOV.W @aa:16, Rd	2				1	
	MOV.W Rs, @Rd	1				1	
	MOV.W Rs, @(d:16,Rd)	2				1	
	MOV.W Rs, @-Rd	1				1	2
	MOV.W Rs, @aa:16	2				1	
MOVFPE	MOVFPE @aa:16, Rd	2			1 *2		
MOVTPE	MOVTPE Rs, @aa:16	2			1 *2		
MULXU	MULXU.B Rs, Rd	1					12
NEG	NEG.B Rd	1					
NOP	NOP	1					
NOT	NOT.B Rd	1					
OR	OR.B #xx:8, Rd	1					
	OR.B Rs, Rd	1					
ORC	ORC #xx:8, CCR	1					
ROTL	ROTL.B Rd	1					
ROTR	ROTR.B Rd	1					
ROTXL	ROTXL.B Rd	1					
ROTXR	ROTXR.B Rd	1					
RTE	RTE	2		2			2
RTS	RTS	2		1			2
SHAL	SHAL.B Rd	1					
SHAR	SHAR.B Rd	1					
SHLL	SHLL.B Rd	1					
SHLR	SHLR.B Rd	1					
SLEEP	SLEEP	1					
STC	STC CCR, Rd	1					
SUB	SUB.B Rs, Rd	1					
	SUB.W Rs, Rd	1					
SUBS	SUBS.W #1/2, Rd	1					

表 2.4　命令の実行状態（サイクル数）(5)

命　令	ニーモニック	命令フェッチ	分岐アドレスリード	スタック操作	バイトデータアクセス	ワードデータアクセス	内部動作
		I	J	K	L	M	N
SUBX	SUBX.B #xx:8, Rd	1					
	SUBX.B Rs, Rd	1					
XOR	XOR.B #xx:8, Rd	1					
	XOR.B Rs, Rd	1					
XORC	XORC #xx:8, CCR	1					

【注】*1　nはＲ４Ｌの設定値です。ソース側、デスティネーション側のアクセスが、それぞれ（ｎ＋１）回行われます。
*2　データアクセスに必要なステート数は、９～16です。

3. 処　理　状　態

ＣＰＵの処理状態には、プログラム実行状態、例外処理状態、低消費電力状態の３種類があります。さらに、低消費電力状態には、スリープモード、ソフトウェアスタンバイモード、ハードウェアスタンバイモードがあります。処理状態の分類を図3.1に、各状態間の遷移を図3.2に示します。

ただし、詳細は当該ＬＳＩのハードウェアマニュアルを参照してください。

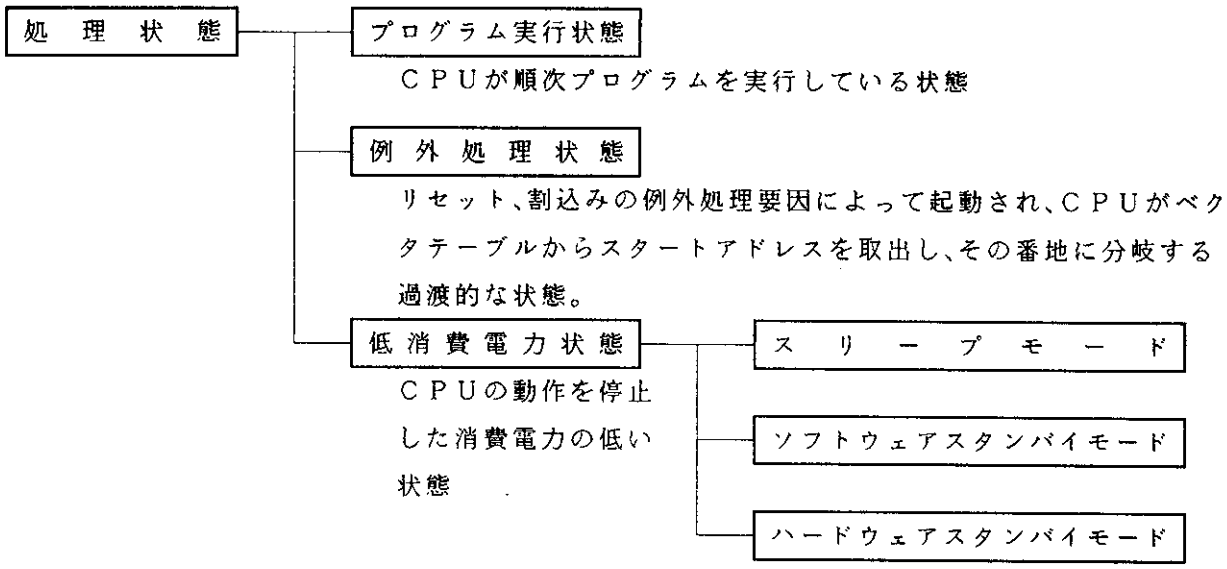
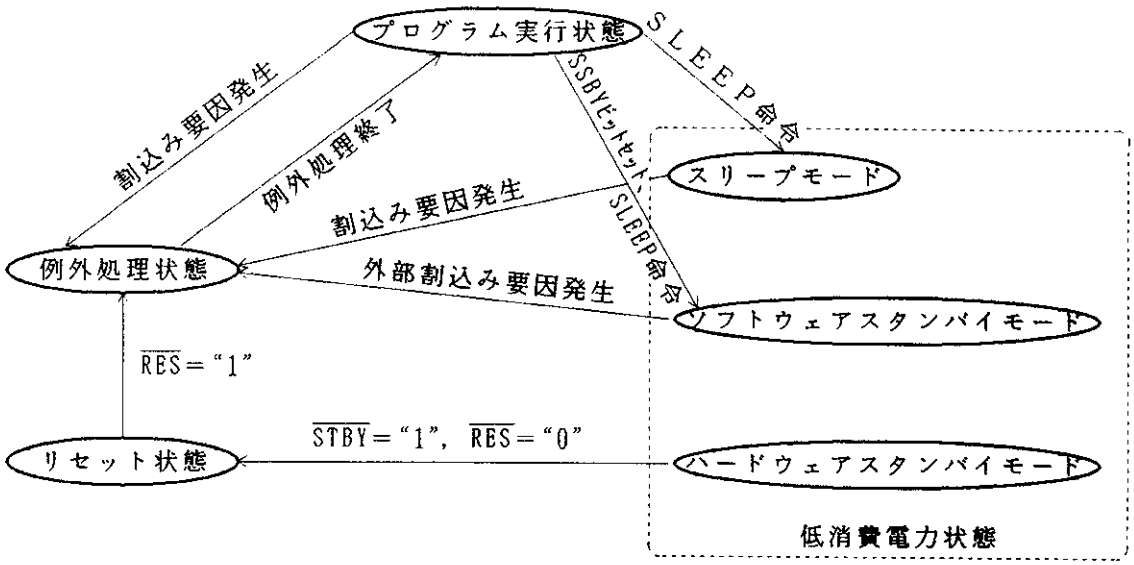


図3.1　処理状態の分類



【注】 1.　ハードウェアスタンバイモードを除くすべての状態においてRES端子が“Low”レベルになるとリセット状態に遷移します。

2.　すべての状態においてSTBY端子を“Low”レベルにすると、ハードウェアスタンバイモードに遷移します。

図3.2　状態遷移図

3.1　プログラム実行状態

ＣＰＵがプログラムを順次実行している状態です。

3.2　例外処理状態

リセット、割込みの例外処理要因によって起動され、ＣＰＵが通常の処理状態の流れを変え、例外処理ベクタテーブルからスタートアドレスを取出し、その番地に分岐する過渡的な状態です。割込み例外処理では、ＳＰ（Ｒ７）を参照して、ＰＣおよびＣＣＲの退避を行います。

3.2.1　例外処理の種類と優先度

例外処理には、リセットと割込みがあります。表 3.1 に、例外処理の種類と優先度を示します。

表 3.1　例外処理の種類と優先度

優先度	例外処理要因	例外処理検出タイミング	例外処理開始タイミング
高 ↑ 低	リセット	クロック同期	$\overline{RES}$端子が"Low"レベルから"High"レベルに変化すると、ただちに例外処理を開始します。
	割込み	命令の実行終了時*	割込み要求が発生すると、命令の実行終了時または例外処理の終了時に例外処理を開始します。

【注】 * ＡＮＤＣ、ＯＲＣ、ＸＯＲＣ、ＬＤＣ命令の実行終了時点またはリセット例外処理の終了時点では、割込み要因の検出を行いません。

3.2.2　例外処理要因とベクタテーブル

例外処理要因は、図 3.3 に示すように分類されます。

例外処理要因とベクタ番号ならびにベクタアドレスの詳細は当該ＬＳＩのハードウェアマニュアルを参照してください。

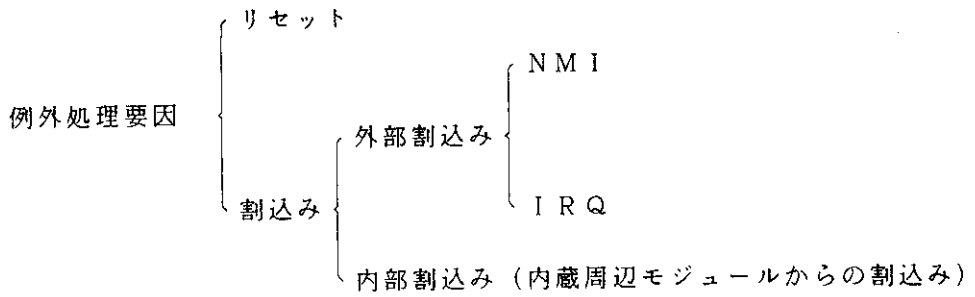
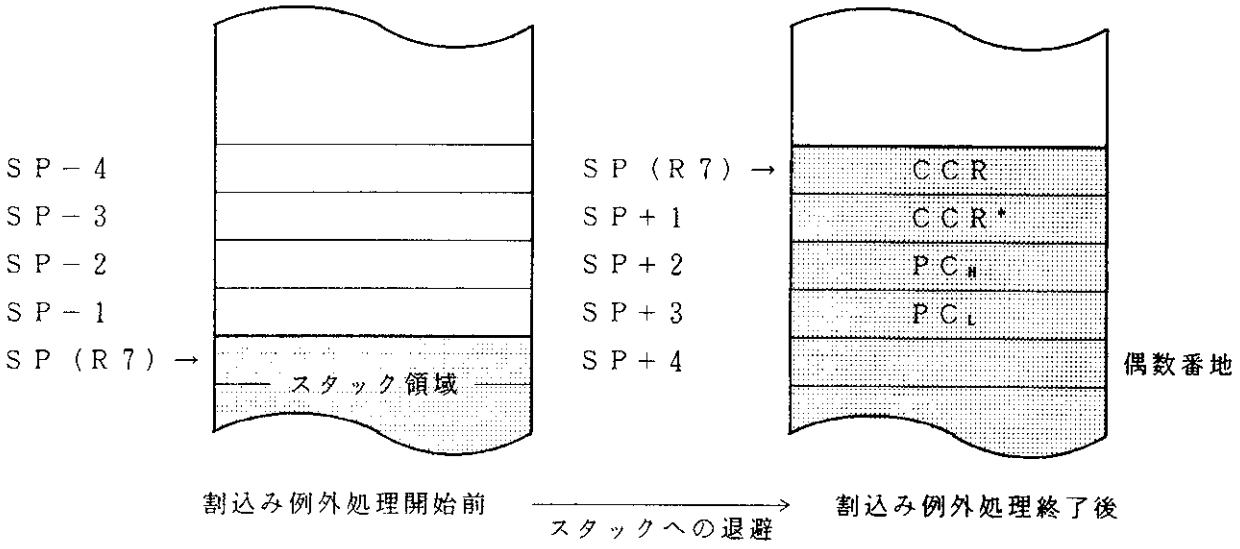


図 3.3　例外処理要因の分類

3.2.3　例外処理の動作

リセット例外処理は、最も優先度の高い例外処理です。$\overline{\text{RES}}$端子を"Low"レベルにしてリセット状態にした後、$\overline{\text{RES}}$端子を"High"レベルにすると、リセット例外処理が起動されます。リセット例外処理が起動されると、ＣＰＵは、例外処理ベクタテーブルからスタートアドレスを取り出し、その番地からプログラムの実行を開始します。リセット例外処理実行中および終了後は、ＮＭＩを含めたすべての割込みが禁止されます。

割込み例外処理が起動されると、ＣＰＵはＳＰ（Ｒ７）を参照してＰＣとＣＣＲをスタックに退避します。その後、ＣＣＲのＩビットを“１”にセットし、例外処理ベクタテーブルからスタートアドレスを取り出し、その番地からプログラムの実行を開始します。例外処理終了後のスタックの状態を図3.4に示します。



<記号説明>

ＰＣH：プログラムカウンタ（ＰＣ）の上位８ビット
ＰＣL：プログラムカウンタ（ＰＣ）の下位８ビット
ＣＣＲ：コンディションコードレジスタ
ＳＰ　：スタックポインタ

【注】　＊　リターン時には無視されます。
1.　ＰＣはリターン後に実行する最初の命令のアドレスです。
2.　レジスタの退避／復帰は必ずワードサイズで、偶数アドレスから行ってください。

図3.4　割込み例外処理終了後のスタック状態

3.3　リセット状態

$\overline{\text{RES}}$端子が"Low"レベルになると、実行中の処理はすべて中止され、ＣＰＵはリセット状態になります。リセットによってＣＣＲのＩビットが"1"にセットされます。リセット状態ではＮＭＩを含めたすべての割込みが禁止されます。

外部から$\overline{\text{RES}}$端子を"Low"レベルから"High"レベルにすると、リセット例外処理が開始されます。

3.4　低消費電力状態

低消費電力状態はＣＰＵの動作を停止して、消費電力を下げる状態です。スリープモード、ソフトウェアスタンバイモード、ハードウェアスタンバイモードの３つのモードがあります。詳細は当該ＬＳＩのハードウェアマニュアルを参照してください。

3.4.1　スリープモード

スリープモードは、ＳＬＥＥＰ命令を実行することによって遷移するモードです。ＣＰＵの動作は、ＳＬＥＥＰ命令実行直後で停止します。ＣＰＵの内部レジスタの内容は保持されます。

3.4.2　ソフトウェアスタンバイモード

ソフトウェアスタンバイモードは、ＳＳＢＹ（ソフトウェアスタンバイ）ビットを"１"にセット後、ＳＬＥＥＰ命令を実行することによって遷移するモードです。

ＣＰＵおよびクロックをはじめ内蔵周辺モジュールのすべての動作が停止します。内蔵周辺モジュールはリセット状態になりますが、規定の電圧が与えられている限り、ＣＰＵの内部レジスタの内容および内蔵ＲＡＭの内容は保持されます。またＩ／Ｏポートの状態も保持されます。

3.4.3　ハードウェアスタンバイモード

ハードウェアスタンバイモードは、$\overline{\text{STBY}}$端子を"Low"レベルにすることによって遷移するモードです。ソフトウェアスタンバイモードと同様に、ＣＰＵおよびすべてのクロックは停止し、内蔵周辺モジュールはリセット状態になりますが、規定の電圧が与えられている限り内蔵ＲＡＭの内容は保持されます。

4. 基本動作タイミング

ＣＰＵは、システムクロック（φ）をタイムベースに動作しており、システムクロックの立上がりから次の立上がりまでの１単位をステートと呼びます。メモリサイクルまたはバスサイクルは、２または３ステートで構成され、内蔵メモリ、内蔵周辺モジュール、外部デバイスによってそれぞれ異なるアクセスを行います。

4.1 内蔵メモリ（ＲＡＭ、ＲＯＭ）

内蔵メモリのアクセスは、高速処理を行うために２ステートアクセスを行います。このとき、データバス幅は16ビットで、バイトおよびワードサイズアクセスが可能です。内蔵メモリアクセスサイクルを図4.1に示します。

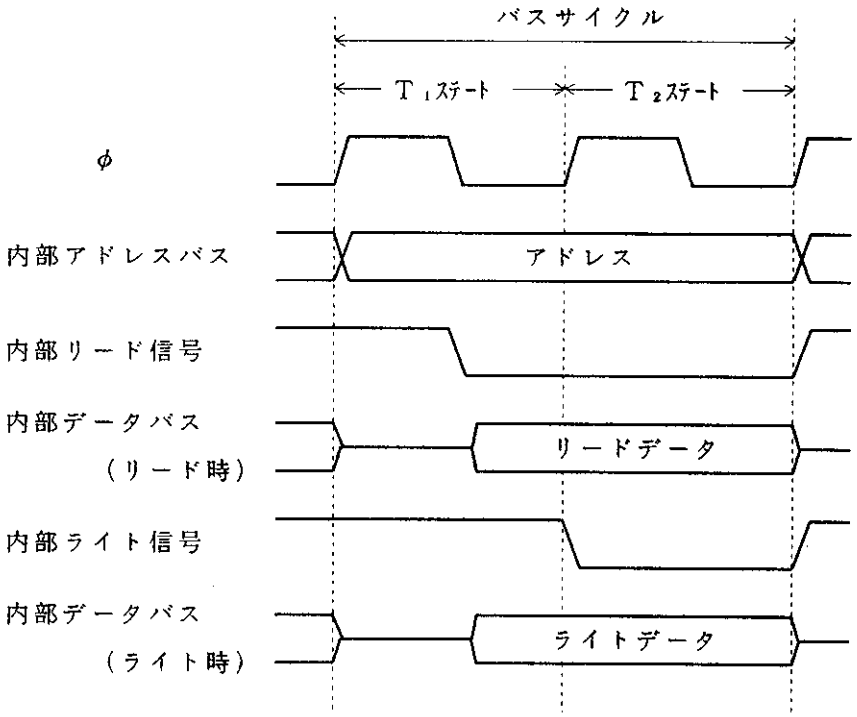


図4.1 内蔵メモリアクセスサイクル

4.2　内蔵周辺モジュール／外部デバイス

内蔵周辺モジュールおよび外部デバイスのアクセスは、3ステートで行われます。このとき、データバス幅は8ビットで、ワードデータおよび命令コードは、1バイトずつ2回に分けてアクセスされます。内蔵周辺モジュールアクセスサイクルを図4.2に、外部デバイスアクセスタイミングを図4.3に示します。

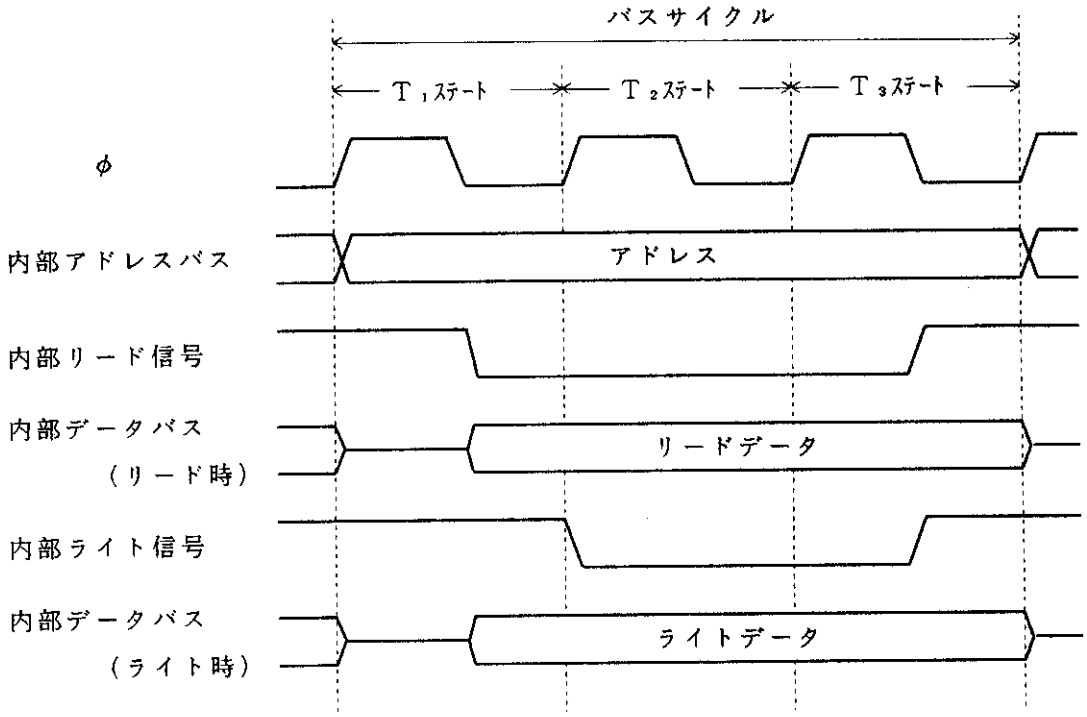


図4.2　内蔵周辺モジュールアクセスサイクル

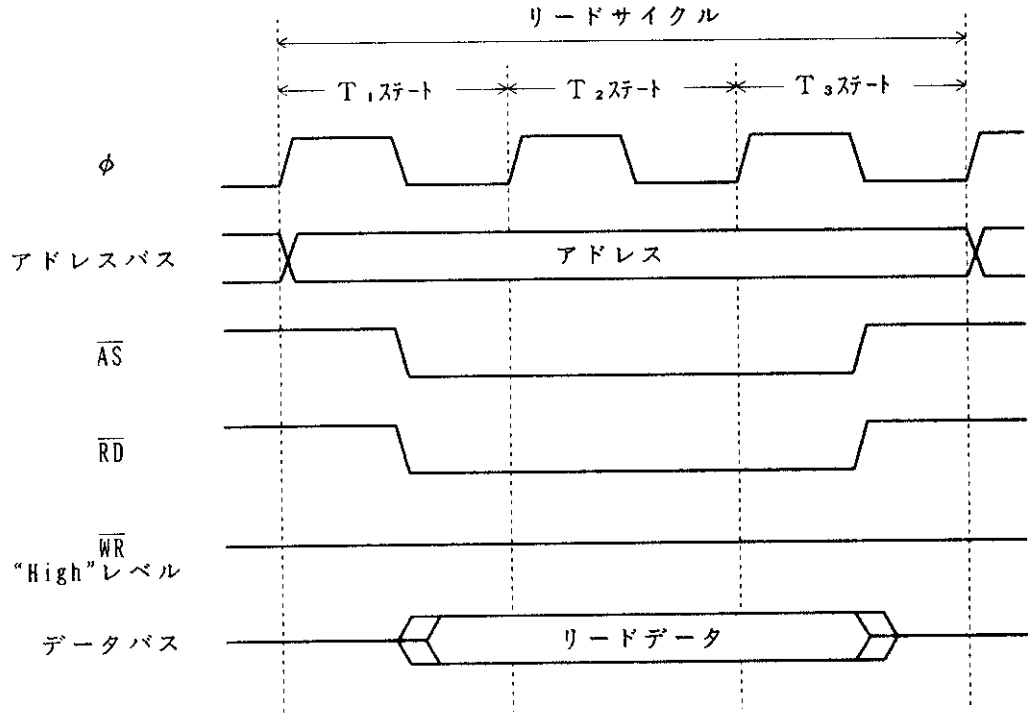


図4.3(a)　外部デバイスアクセスタイミング（リード時）

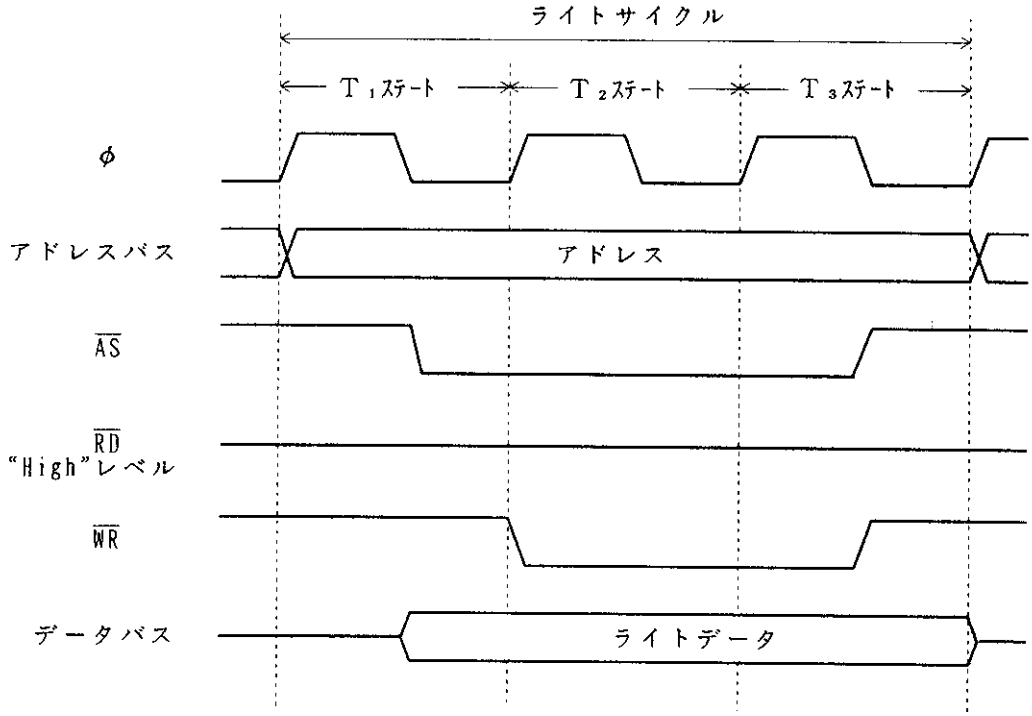


図 4.3 (b)　外部デバイスアクセスタイミング（ライト時）

4.3　Eクロックインタフェース

H8／300シリーズの中でEクロック出力端子を備えたLSIは、外部拡張モード時、Eクロック同期転送命令（MOVFPE，MOVTPE）を使用でき、Eクロック入力を必要とする周辺LSIとインタフェースをとることができます。Eクロックはシステムクロック（φ）を 1/8に分周したもので、外部拡張モード時、出力することができます。

CPUがEクロック同期転送命令を実行すると、アドレスバス、$\overline{AS}$, $\overline{IOS}$ の各端子は通常アクセスと同様に出力されますが、データバスおよび$\overline{RD}$, $\overline{WR}$の各端子はEクロックの立下がりを検出した後，アクティブとなります。このタイミングを図**4.4**、図**4.5**に示します。Eクロック同期転送命令の実行サイクルは9～16ステートで、一定にはなりません。

またEクロック同期転送命令の実行サイクル中は$\overline{WAIT}$端子によるウェイトステート(Tw)の挿入はできません。

詳細は当該LSIのハードウェアマニュアルを参照してください。

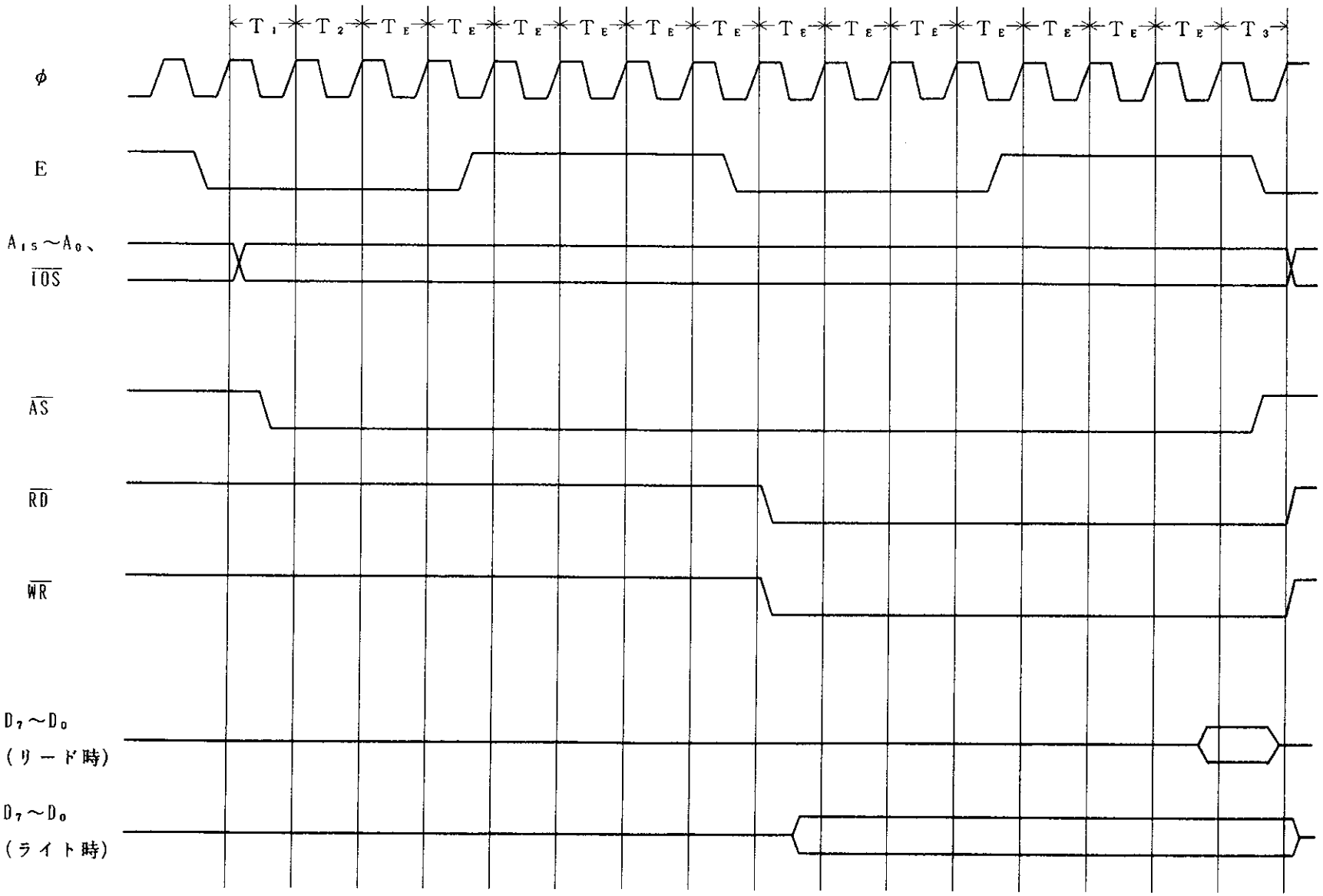


図 4.4　拡張モード時Eクロック同期転送命令実行サイクル（最大同期）

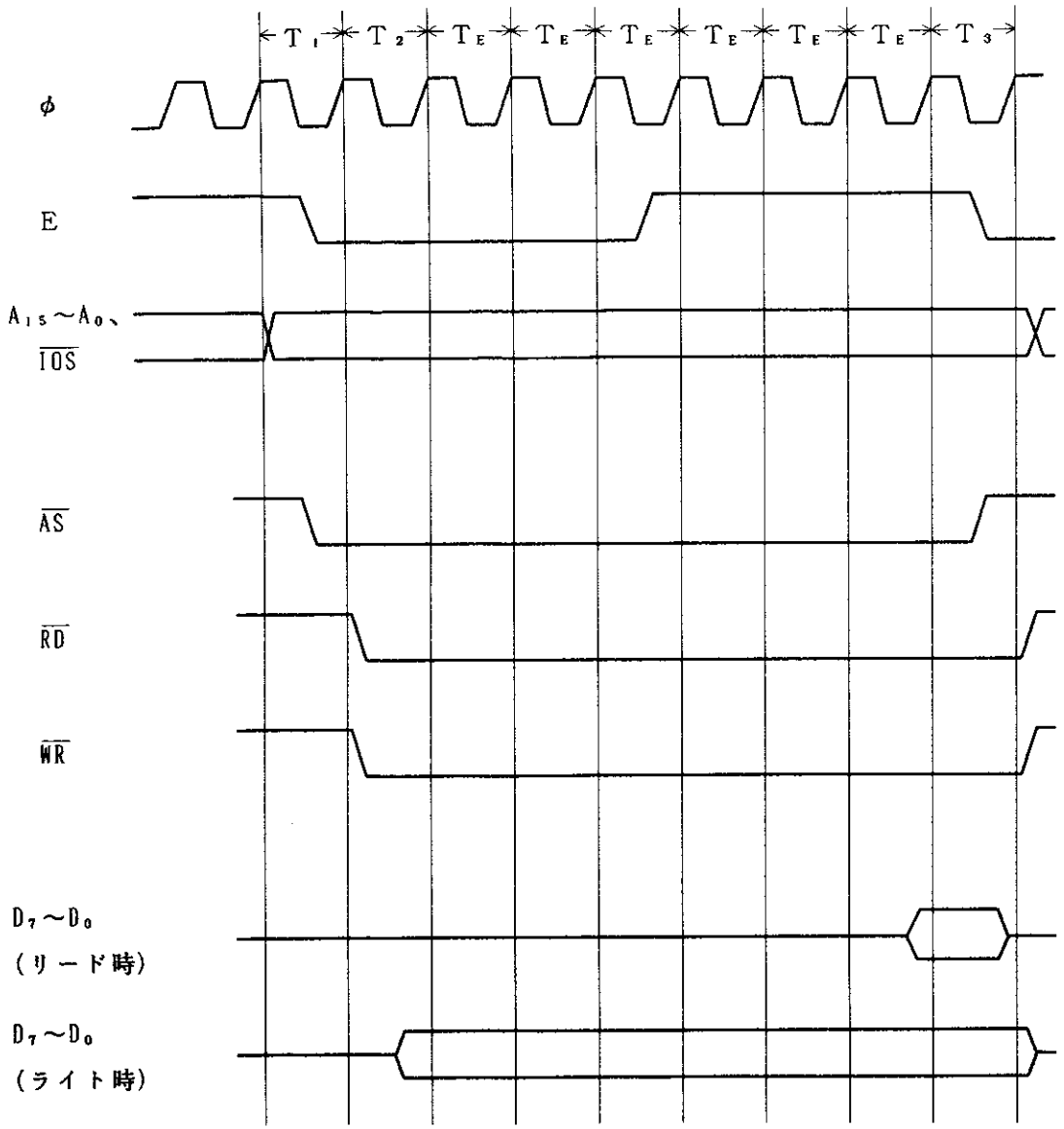


図 4.5　拡張モード時Eクロック同期転送命令実行サイクル（最小同期）

H8/300シリーズ プログラミングマニュアル

発行年月　平成元年7月　第1版
　　平成5年3月　第3版
発　　行　株式会社 日立製作所
　　半導体事業本部統括営業本部
編　　集　株式会社 超LSIメディア
　　技術ドキュメントグループ

H8/300 シリーズ
プログラミングマニュアル

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-602-026B