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「三菱電機」、「三菱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日を以って株式会社日立製作所及び三菱電機株式会社のマイコン、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「三菱電機」、「三菱電機株式会社」、「三菱半導体」、「三菱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。

注:「高周波・光素子事業、パワーデバイス事業については三菱電機にて引き続き事業運営を行います。」

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

User's Manual

32

M32R ファミリ

ソフトウェアマニュアル

ルネサス32ビットシングルチップマイクロコンピュータ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.10　2000.03

安全設計に関するお願い

・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

・本資料は、お客様が用途に応じた適切な三菱半導体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三菱電機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。

・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、三菱電機は責任を負いません。

・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、三菱電機は、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。三菱半導体製品のご購入に当たりましては、事前に三菱電機または特約店へ最新の情報をご確認頂きますとともに、三菱電機半導体情報ホームページ（http://www.semicon.melco.co.jp/）などを通じて公開される情報に常にご注意ください。

・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、三菱電機はその責任を負いません。

・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。三菱電機は、適用可否に対する責任は負いません。

・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、三菱電機または特約店へご照会ください。

・本資料の転載、複製については、文書による三菱電機の事前の承諾が必要です。

・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三菱電機または特約店までご照会ください。

はじめに

このたび、CMOS32ビットマイクロコンピュータ M32Rファミリのソフトウエアについて、マニュアルを作成いたしましたのでご案内申し上げます。
このソフトウエアマニュアルは、ユーザの皆様にM32Rファミリのソフトウエアをよく理解していただき、その機能を最大限に生かしていただくために作成いたしました。M32RファミリのCPUの特長や命令体系について詳細に説明しておりますので、広くご活用ください。

なお、M32Rファミリ各機種のハードウエア及び開発サポートツールにつきましては、各ユーザーズマニュアル又は各操作説明書をご併用くださいますようお願い申し上げます。

PDF ファイル 改訂履歴	M32R ファミリ ソフトウエアマニュアル

Rev. No.	改訂内容	Rev. date
1.0	**PDF**ファイル初版発行 製本第二版(**HU-078B**；**1998**年**5**月発行)の内容に、テクニカルニュース**No.M32R-12-9807**「**M32R**ファミリソフトウエアマニュアル正誤表(**Rev.B**)」を反映。	990521
1.1	テクニカルニュース**No.M32R-21-0002**「**M32R**ファミリソフトウエアマニュアル正誤表(**Rev.C**)」を反映。 ・「付録**1 16**進命令コード対応表」の追加 ・「付録**5** 注意事項」の追加	000316

目次

第１章

CPUプログラミングモデル

1.1 CPUレジスタ

M32RファミリCPU(以下M32R CPU)には16本の汎用レジスタ、5本の制御レジスタ、アキュムレータ及びプログラムカウンタがあります。アキュムレータは56ビット、その他のレジスタはすべて32ビット構成になっています。

1.2 汎用レジスタ

汎用レジスタは32ビット幅で16本(R0～R15)あり、データやベースアドレスの保持などに使用します。R14はリンクレジスタとして、R15はスタックポインタとして使用されます。リンクレジスタはサブルーチン呼び出し命令実行の際、戻り先番地の格納に使われます。またスタックポインタは、プロセッサ状態語レジスタ(PSW)のスタックモード(SM)ビットの値に応じて割り込み用スタックポインタ(SPI)と、ユーザ用スタックポインタ(SPU)とに切り替わります。

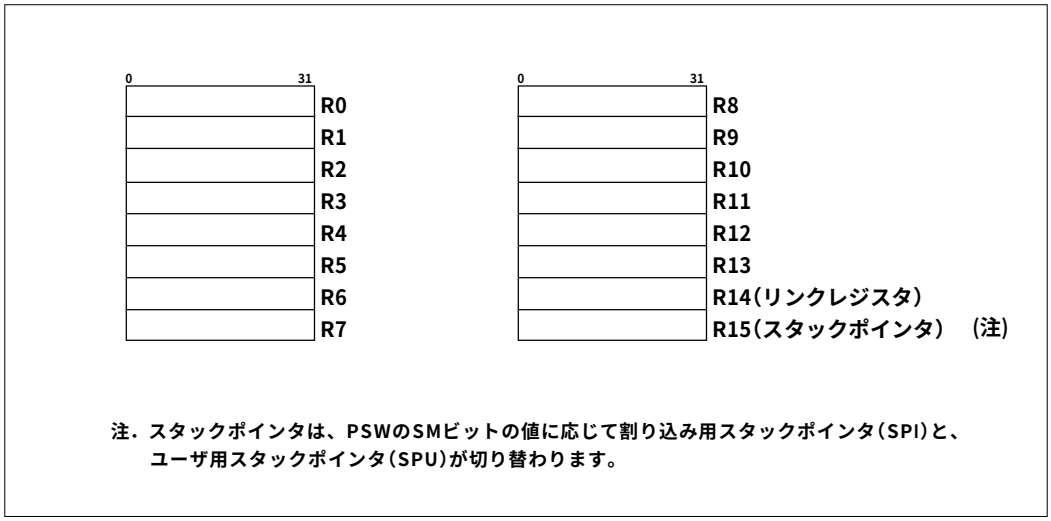


図1.2.1 汎用レジスタ

1.3 制御レジスタ

制御レジスタ(CR)には、プロセッサ状態語レジスタ(PSW)，条件ビットレジスタ(CBR)，割り込み用スタックポインタ(SPI)，ユーザ用スタックポインタ(SPU)，バックアップPC(BPC)の5つがあります。

これら制御レジスタの設定や読み出しには、専用の「MVTC命令」と「MVFC命令」を使用します。

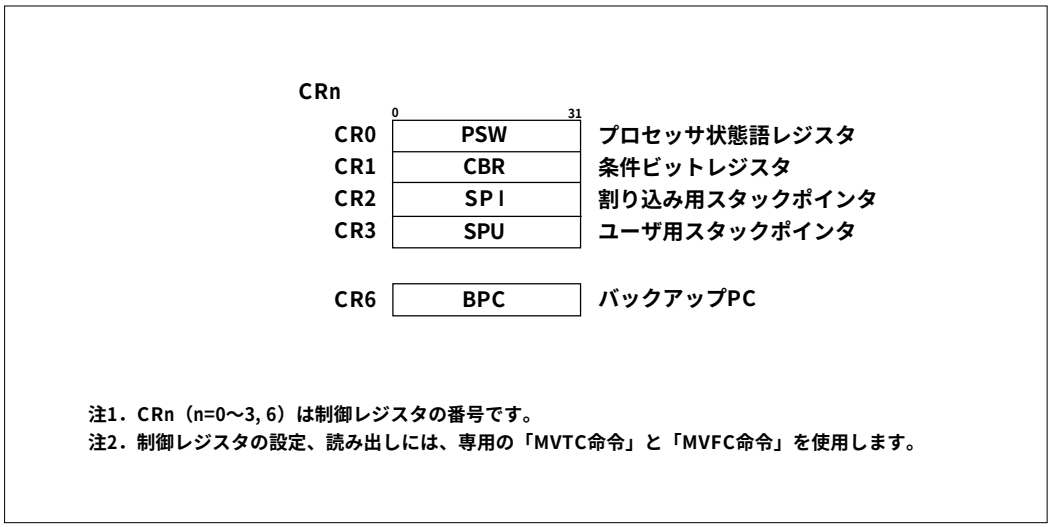


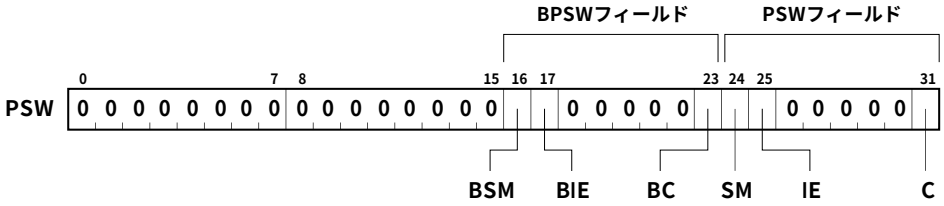
図1.3.1 制御レジスタ

1.3.1 プロセッサ状態語レジスタ：PSW（CR0）

プロセッサ状態語レジスタ（PSW）は、M32R CPUのステータスを表示するレジスタで、通常使用するPSWフィールドと、EIT発生時にPSWフィールドを退避するためのBPSWフィールドからなります。

PSWフィールドは、スタックモードビット(SM)，割り込みイネーブルビット(IE)，条件ビット(C)の各ビットで構成されています。

また、BPSWフィールドはバックアップSMビット(BSM)，バックアップIEビット(BIE)，バックアップCビット(BC)で構成されています。

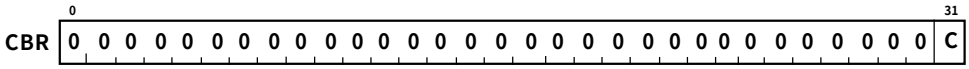


D	ビット名	機能	初	R（注）	W（注）
16	BSM（バックアップSM）	EIT受け付け時に、SMビットの値が保存される	不定	○	○
17	BIE（バックアップIE）	EIT受け付け時に、IEビットの値が保存される	不定	○	○
23	BC（バックアップC）	EIT受け付け時に、Cビットの値が保存される	不定	○	○
24	SM（スタックモード）	0：割り込み用スタックポインタを使用 1：ユーザ用スタックポインタを使用	0	○	○
25	IE（割り込みイネーブル）	0：割り込みを受け付けない 1：割り込みを受け付ける	0	○	○
31	C（条件ビット）	命令の実行に応じて演算結果のキャリー、ボロー、オーバフローの有無を示す	0	○	○

注. 初＝リセット直後の初期状態，R＝○は読み出し可能を，W＝○は書き込み可能を表します。

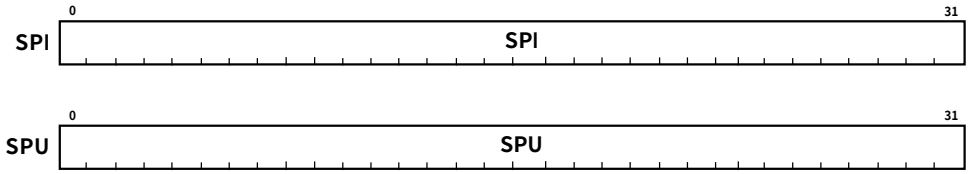
1.3.2 条件ビットレジスタ：CBR(CR1)

条件ビットレジスタ(CBR)は、PSWのうち条件ビット(C)を抜き出して別レジスタとしたものです。PSWの条件ビット(C)に書き込まれた値はこのレジスタに反映されます。このレジスタは読み出しのみ可能です(「MVTC命令」で書き込みを行っても無視されます)。



1.3.3 割り込み用スタックポインタ：SPI(CR2)
ユーザ用スタックポインタ：SPU(CR3)

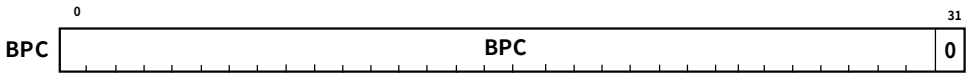
割り込み用スタックポインタ(SPI)、ユーザ用スタックポインタ(SPU)は、現在のスタックポインタのアドレスを保持します。これらのレジスタは、汎用レジスタR15としてアクセスできます。このときR15をSPIとして使用するかSPUとして使用するかは、PSWのスタックモードビット(SM)によって切り替わります。



1.3.4 バックアップPC：BPC(CR6)

バックアップPC(BPC)は、EIT発生時にプログラムカウンタ(PC)の値を退避するためのレジスタです。ビット31は"0"に固定です。

EIT発生時には発生したEITによってEIT発生時または次命令のPC値がセットされ、「RTE命令」実行時にBPCの値はPCに戻されます。ただし復帰時にPCの下位2ビットは常に"00"になります(常にワード境界に復帰します)。

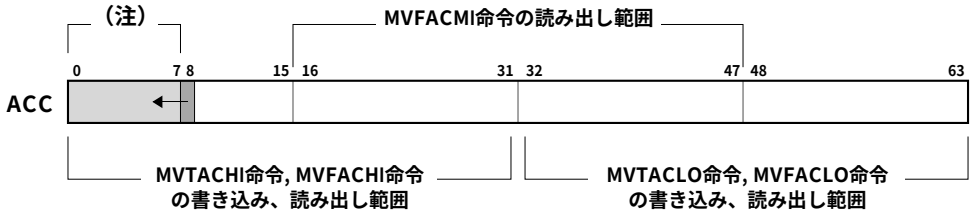


1.4 アキュムレータ

アキュムレータ(ACC)は、DSP機能用命令で使用される56ビットのレジスタです。
読み出し時や書き込み時には64ビットのレジスタとして扱われ、読み出し時にはビット8の値が符号拡張されます。書き込み時にはビット0～7は無視されます。また、アキュムレータは乗算命令「MUL」でも使用され、この命令実行の際はアキュムレータの値が破壊されるので注意してください。

アキュムレータへの書き込みには「MVTACHI命令」と「MVTACLO命令」を使用します。「MVTACHI命令」は上位側32ビット(ビット0～31)に、「MVTACLO命令」は下位側32ビット(ビット32～63)にデータを書き込みます。

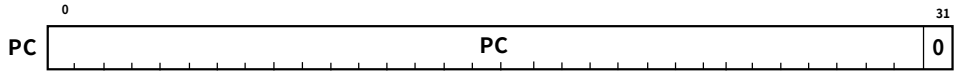
読み出しには「MVFACHI命令」、「MVFACLO命令」および「MVFACMI命令」を使用します。「MVFACHI命令」は上位側32ビット(ビット0～31)を、「MVFACLO命令」は下位側32ビット(ビット32～63)を、また「MVFACMI命令」は中央の32ビット(ビット16～47)のデータをそれぞれ読み出します。



注. ビット0～7は、ビット8の値を符号拡張した値が常に読み出されます。この部分への書き込みは無視されます。

1.5 プログラムカウンタ

プログラムカウンタ(PC)は32ビットのカウンタで、現在実行中の命令アドレスを保持します。M32R CPUの命令は偶数アドレスから始まるため、LSB(ビット31)は"0"になります。



1.6 データフォーマット

1.6.1 データタイプ

M32R CPUの命令セットで扱えるデータタイプは、符号付き、または符号なしの8, 16, 32ビット整数です。符号付き整数の値は2の補数で表現されます。

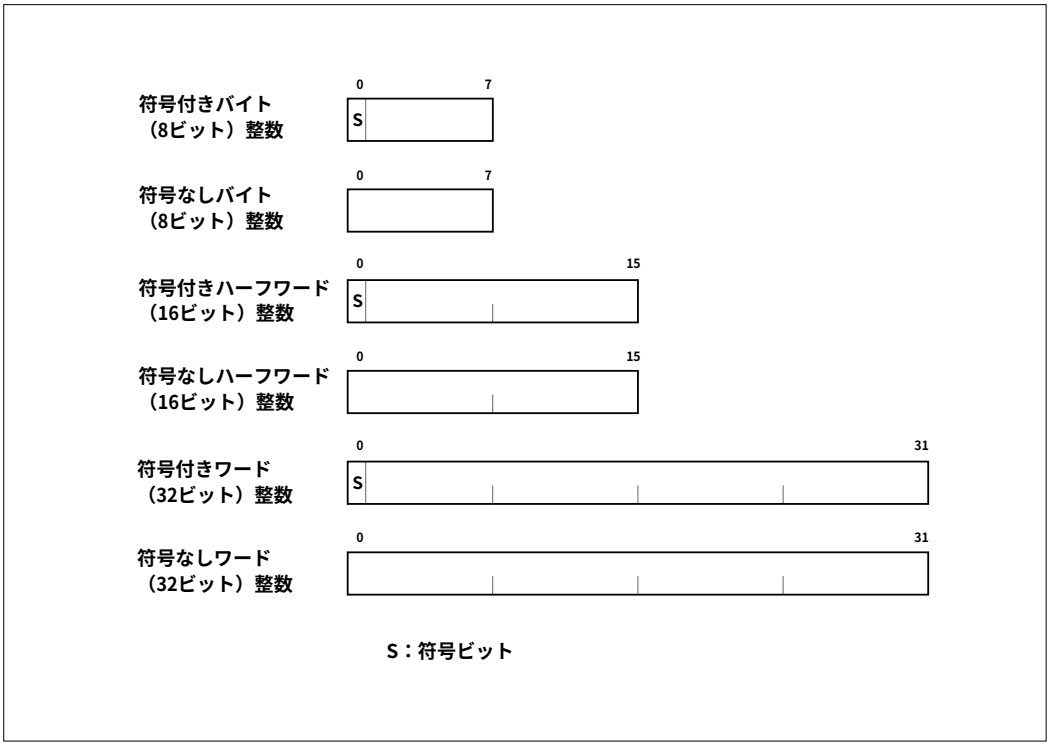


図1.6.1 データタイプ

1.6.2 データフォーマット

(1) M32R CPU レジスタ上のデータフォーマット

M32R CPUのレジスタ上でのデータサイズは常にワード(32ビット)です。

メモリ上のバイト(8ビット)、ハーフワード(16ビット)のデータをロードする場合は、ワード(32ビット)データに符号拡張(LDB，LDH命令)またはゼロ拡張(LDUB，LDUH命令)後、レジスタに格納されます。

M32R CPUのレジスタ上のデータをメモリにストアする場合は、ST命令ではレジスタ上の32ビットデータ、STH命令ではLSB側の16ビットデータ、またSTB命令ではLSB側8ビットデータをそれぞれメモリにストアします。

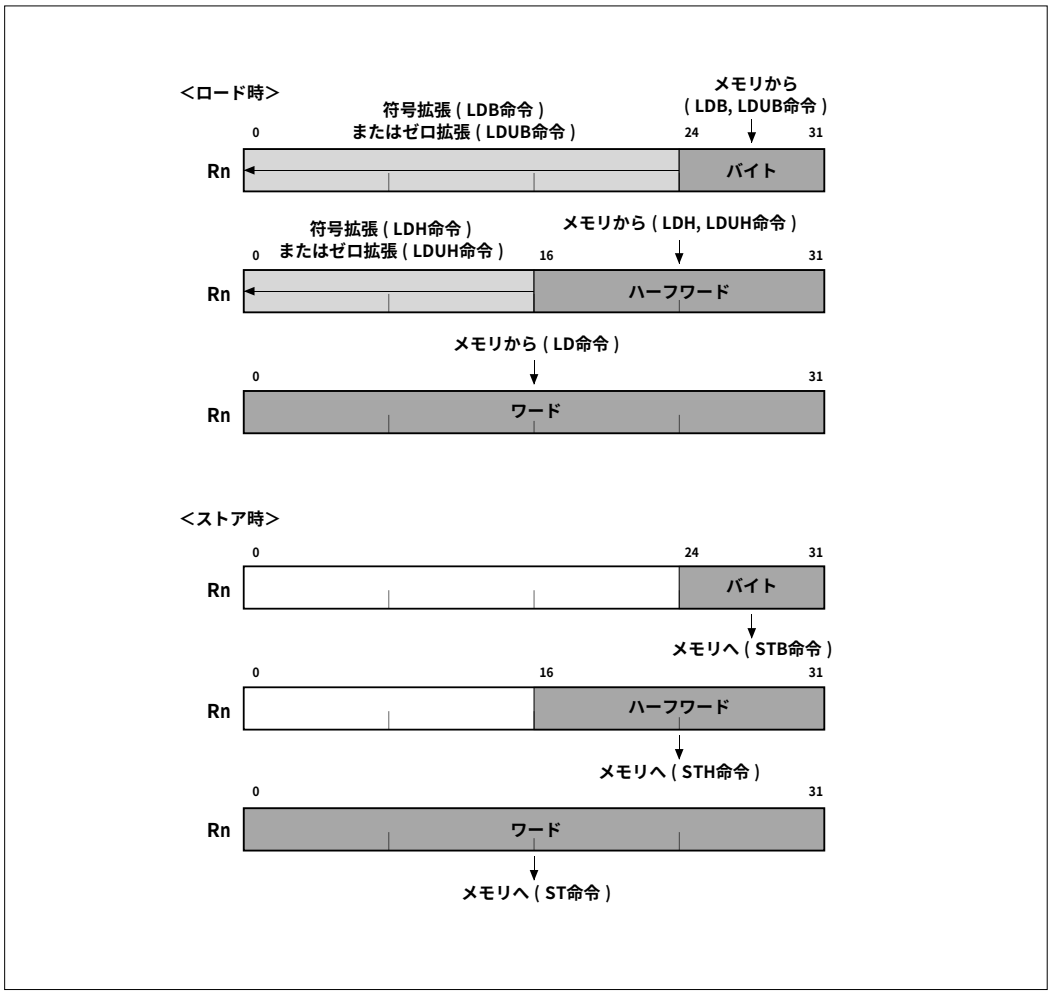


図1.6.2 レジスタ上のデータフォーマット

(2) メモリ上のデータフォーマット

メモリ上でのデータサイズはバイト(8ビット)、ハーフワード(16ビット)、ワード(32ビット)の3種類です。バイトデータは任意のアドレスに配置できますが、ハーフワードデータはハーフワード境界(アドレスの最下位ビットが"0"の番地)、またワードデータはワード境界(アドレスの下位2ビットが"00"の番地)に配置されなければなりません。この境界をまたぐメモリデータをアクセスしようとするとアドレス例外が発生します。

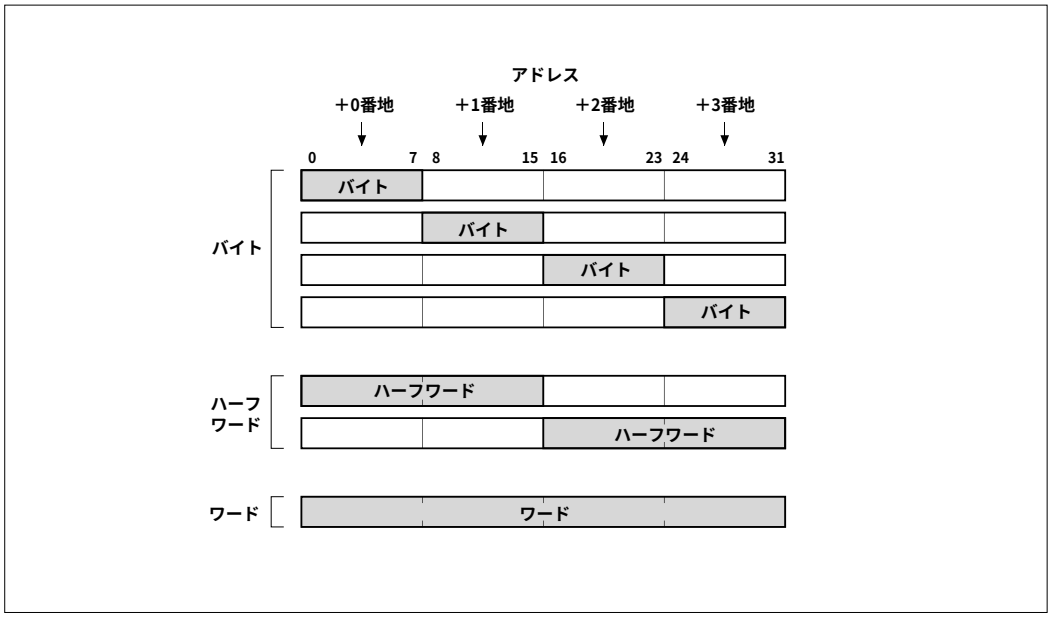


図1.6.3 メモリ上のデータフォーマット

1.7 アドレッシングモード

M32R CPUのアドレッシングモードには、以下のものがあります。

(1) レジスタ直接〔表記：RまたはCR〕

操作の対象として汎用レジスタまたは制御レジスタを指定するもの。

(2) レジスタ間接〔表記：@R〕

レジスタの値をアドレスとするもの(すべてのロード／ストア命令で指定可能)。

(3) レジスタ相対間接〔表記：@(disp,R)〕

(レジスタの値)＋(16ビットのディスプレースメントを32ビットに符号拡張した値)をアドレスとするもの。

(4) レジスタ間接＋レジスタ更新

- レジスタ値を＋4する**〔表記：@R+〕**

  更新前のレジスタ値をアドレスとするもの(LD命令でのみ指定可能)。

- レジスタ値を＋4する**〔表記：@+R〕**

  更新後のレジスタ値をアドレスとするもの(ST命令でのみ指定可能)。

- レジスタ値を－4する**〔表記：@-R〕**

  更新後のレジスタ値をアドレスとするもの(ST命令でのみ指定可能)。

(5) イミディエート〔表記：#imm 〕

4, 5, 8, 16 または 24ビットの即値(値の扱いは各命令の詳細説明参照)。

(6) PC相対〔表記：pcdisp 〕

(PCの値)＋(8ビット, 16ビットまたは24ビットのディスプレースメントを32ビットに符号拡張して左へ2ビットシフトした値)をアドレスとするもの。

第２章

命令セット

2.1 命令セット概要

M32R CPUの命令は全部で83あります。RISCアーキテクチャを採用しており、メモリアクセスは基本的にロード命令とストア命令で行います。また、各種の演算はレジスタ間演算で実行します。さらに、ロード＆アドレス更新、ストア＆アドレス更新といった複合命令もサポートしています。

2.1.1 ロード／ストア命令

メモリ～レジスタ間のデータ転送を行います。

LD	Load
LDB	Load byte
LDUB	Load unsigned byte
LDH	Load halfword
LDUH	Load unsigned halfword
LOCK	Load locked
ST	Store
STB	Store byte
STH	Store halfword
UNLOCK	Store unlocked

ロード／ストア命令におけるアドレッシングモードは、次の３種類を指定可能です。

(1) レジスタ間接

レジスタの値をアドレスとするもの(すべてのロード／ストア命令で指定可能)。

(2) レジスタ相対間接

(レジスタの値)＋(16ビットのディスプレースメントを32ビットに符号拡張した値)をアドレスとするもの(LOCK,UNLOCK命令以外の命令で指定可能)。

(3) レジスタ間接＋レジスタ更新

- レジスタ値を＋4する
  更新前のレジスタ値をアドレスとするもの(LD命令でのみ指定可能)
- レジスタ値を＋4する
  更新後のレジスタ値をアドレスとするもの(ST命令でのみ指定可能)
- レジスタ値を－4する
  更新後のレジスタ値をアドレスとするもの(ST命令でのみ指定可能)

いずれのアドレッシングモードを使用した場合でも、メモリ上のデータフォーマットの規則を守る必要があります。ハーフワード、ワードサイズのデータをアクセスする場合には、それぞれハーフワードアライメント、ワードアライメントのとれたアドレスを指定します(ハーフワードサイズの場合にはアドレスの下位２ビットは"00"か"10"、ワードサイズの場合にはアドレスの下位２ビットは"00")。アライメントのとれていないアドレスを指定するとアドレス例外が発生します。

ロード命令でバイトデータ、ハーフワードデータをアクセスした場合には、上位ビットが符号拡張またはゼロ拡張された32ビットデータとしてレジスタに格納されます。

2.1.2 転送命令

レジスタ～レジスタ間またはレジスタ～イミディエート(即値)の転送を行います。

LD24	Load 24-bit immediate
LDI	Load immediate
MV	Move register
MVFC	Move from control register
MVTC	Move to control register
SETH	Set high-order 16-bit

2.1.3 演算命令

レジスタ～レジスタ間で比較、算術論理演算、乗除算、シフトなどを行います。

■比較

CMP	Compare
CMPI	Compare immediate
CMPU	Compare unsigned
CMPUI	Compare unsigned immediate

■算術演算

ADD	Add
ADD3	Add 3-operand
ADDI	Add immediate
ADDV	Add with overflow checking
ADDV3	Add 3-operand with overflow checking
ADDX	Add with carry
NEG	Negate
SUB	Subtract
SUBV	Subtract with overflow checking
SUBX	Subtract with borrow

■論理演算

AND	AND
AND3	AND 3-operand
NOT	Logical NOT
OR	OR
OR3	OR 3-operand
XOR	Exclusive OR
XOR3	Exclusive OR 3-operand

■乗除算

DIV	Divide
DIVU	Divide unsigned
MUL	Multiply
REM	Remainder
REMU	Remainder unsigned

■シフト

SLL	Shift left logical
SLL3	Shift left logical 3-operand
SLLI	Shift left logical immediate
SRA	Shift right arithmetic
SRA3	Shift right arithmetic 3-operand
SRAI	Shift right arithmetic immediate
SRL	Shift right logical
SRL3	Shift right logical 3-operand
SRLI	Shift right logical immediate

2.1.4 分岐命令

プログラムの流れを変えるための命令です。

BC	Branch on C-bit
BEQ	Branch on equal to
BEQZ	Branch on equal to zero
BGEZ	Branch on greater than or equal to zero
BGTZ	Branch on greater than zero
BL	Branch and link
BLEZ	Branch on less than or equal to zero
BLTZ	Branch on less than zero
BNC	Branch on not C-bit
BNE	Branch on not equal to
BNEZ	Branch on not equal to zero
BRA	Branch
JL	Jump and link
JMP	Jump
NOP	No operation

分岐先として指定できるのはワードアライメントのとれたアドレス(ワード境界)だけです。

BRA, BL, BC, BNC命令のアドレッシングモードは、8ビットか24ビットのイミディエート値を指定できます。またBEQ, BNE, BEQZ, BNEZ, BGTZ, BLTZ, BGEZ, BLEZ命令のアドレッシングモードは、16ビットのイミディエート値です。

JMP, JL命令は、レジスタの値が分岐先アドレスとなります。ただし、レジスタの下位2ビットの値は無視されます。

その他の分岐命令は、(分岐命令のPC値)＋(符号拡張されたイミディエート値を左に2ビットシフトした値)が分岐先アドレスとなります。ただし、加算が行われる際のPC値の下位2ビットは"00"にクリアされます。たとえば図2.1.1で「命令A」または「命令B」が分岐命令で、「命令G」に分岐する場合、いずれもイミディエート値は4になります。

サブルーチンコール用のJL, BL命令は、分岐と同時に戻り先PC値がR14に格納されます。R14に格納される値は(分岐命令のPC値＋4)で、PC値の下位2ビットは"00"にクリアされます。

たとえば図2.1.1で「命令A」または「命令B」がJL, BL命令であった場合、いずれも戻り先は「命令C」となります。

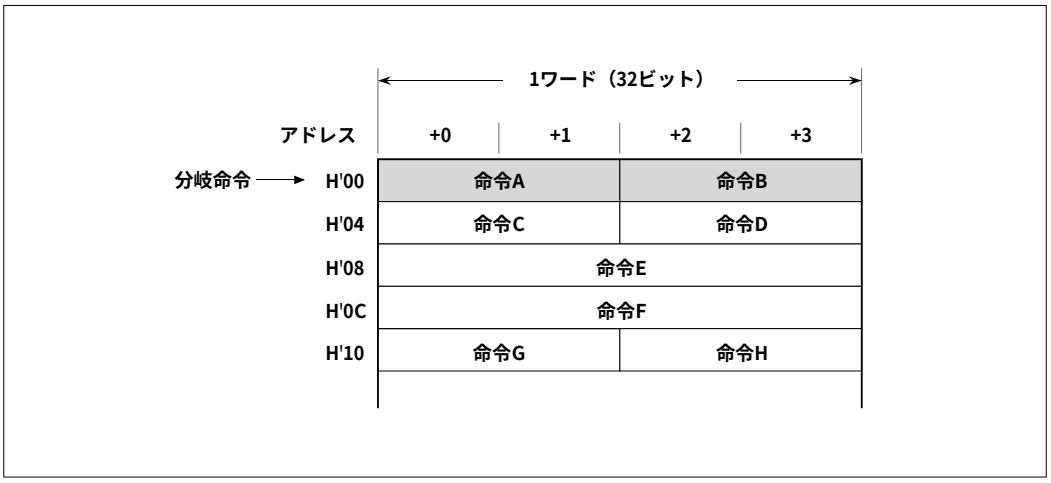


図2.1.1　分岐命令の分岐先

2.1.5 EIT関連命令

EIT関連命令は、M32RのEIT事象(Exception：例外, Interrupt：割り込み, Trap：トラップ)のための命令です。EIT関連命令にはトラップの起動命令とEIT処理からの復帰命令があります。

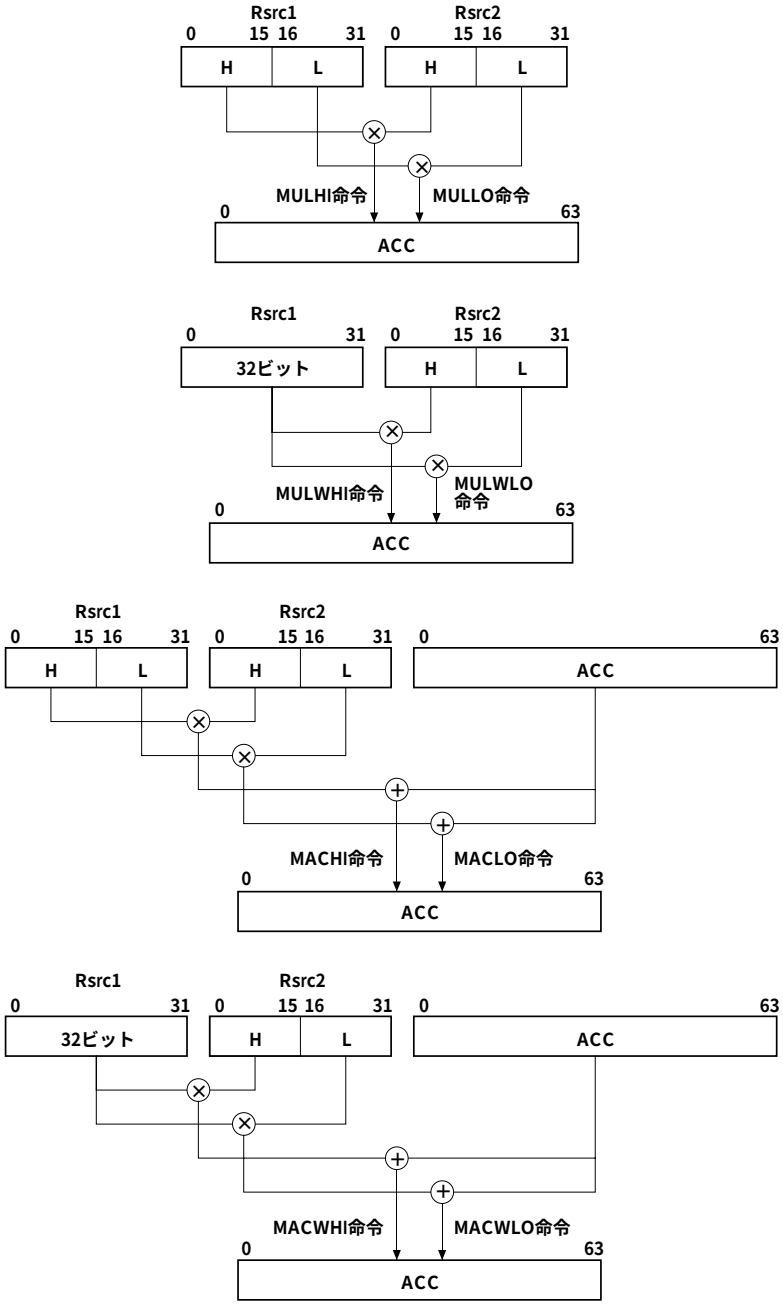
TRAP	Trap
RTE	Return from EIT

2.1.6 DSP機能用命令

32ビット×16ビット、16ビット×16ビットの乗算や積和演算を行います。また、アキュムレータ内のデータの丸めやアキュムレータ～汎用レジスタ間の転送を行います。

MACHI	Multiply-accumulate high-order halfwords
MACLO	Multiply-accumulate low-order halfwords
MACWHI	Multiply-accumulate word and high-order halfword
MACWLO	Multiply-accumulate word and low-order halfword
MULHI	Multiply high-order halfwords
MULLO	Multiply low-order halfwords
MULWHI	Multiply word and high-order halfword
MULWLO	Multiply word and low-order halfword
MVFACHI	Move high-order word from accumulator
MVFACLO	Move low-order word from accumulator
MVFACMI	Move middle-order word from accumulator
MVTACHI	Move high-order word to accumulator
MVTACLO	Move low-order word to accumulator
RAC	Round accumulator
RACH	Round accumulator halfword

次ページにこれらの命令の動作概要を示します。



注. 実際のDSP機能用命令の演算では結果の格納位置の調整や符号拡張が行われます。
詳しくは第3章「命令」を参照ください。

図2.1.2 DSP機能用命令の動作 1（乗算、積和演算）

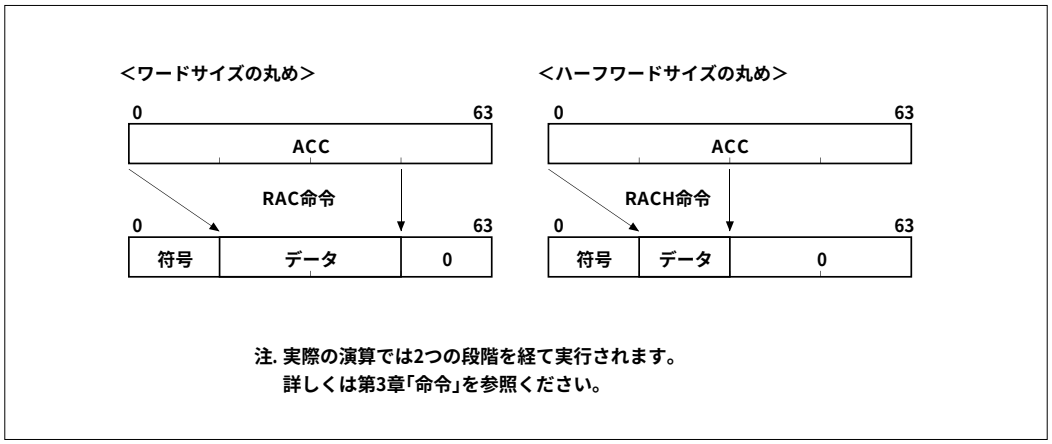


図2.1.3　DSP機能用命令の動作 2（丸め操作）

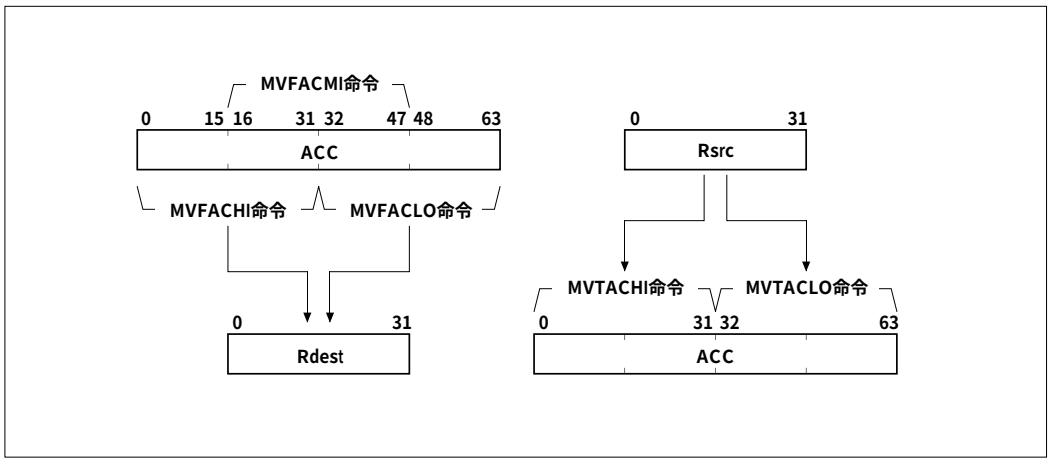


図2.1.4　DSP機能用命令の動作 3（アキュムレータ～レジスタ間転送）

2.2 命令フォーマット

M32R CPUの命令フォーマットは2種類あり、1つは32ビットワード境界内に格納された2つの16ビット命令、もう1つは単一の32ビット命令です(図2.2.1参照)。

M32R CPUの基本命令フォーマットを図2.2.2に示します。

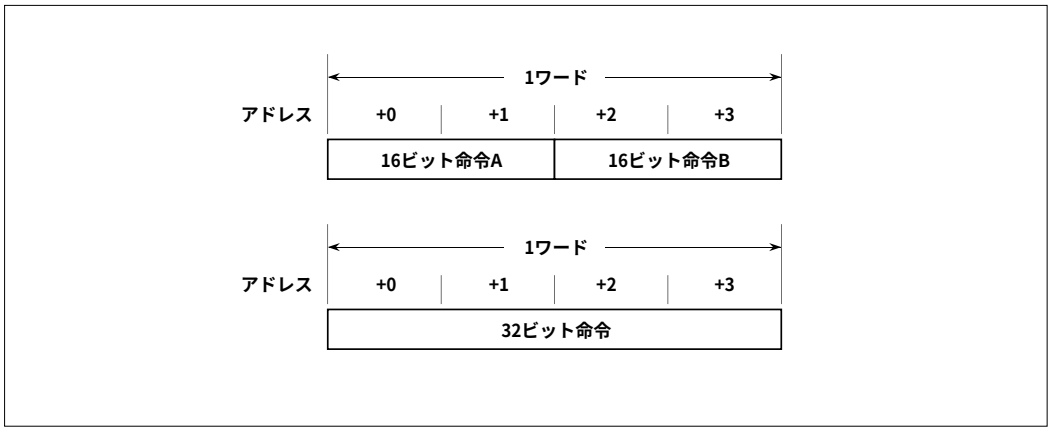


図2.2.1 16ビット命令と32ビット命令

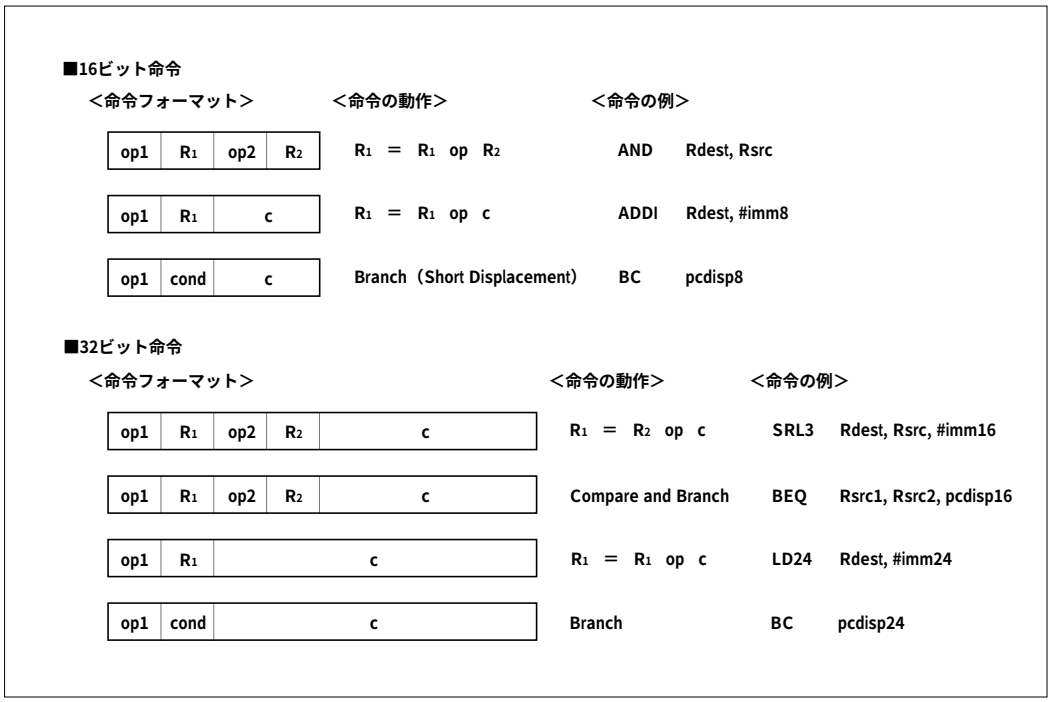


図2.2.2 M32R CPUの基本命令フォーマット

32ビット命令のMSB(Most Significant Bit)は常に"1"です。16ビット命令の場合、上位のハーフワード境界に存在する命令のMSBは常に"0"ですが、それにつづく16ビット命令は、その命令のMSBの値によって処理が異なります。

図2.2.3において「命令B」のMSBが"0"の場合、「命令A」と「命令B」はシーケンシャルに実行されますが、「命令B」のMSBが"1"の場合は、「命令A」と「命令B」はパラレルに実行されます。

ただし現状のインプリメンテーションレベルでは「NOP命令」のみ、このパラレル実行をサポートします(例としてワードアライメント調整のためのNOP命令は、アセンブラが自動的にMSBを1にしたNOP命令に変えるため、M32R CPUは見かけ上クロックを消費しない命令としてこれを処理します)。

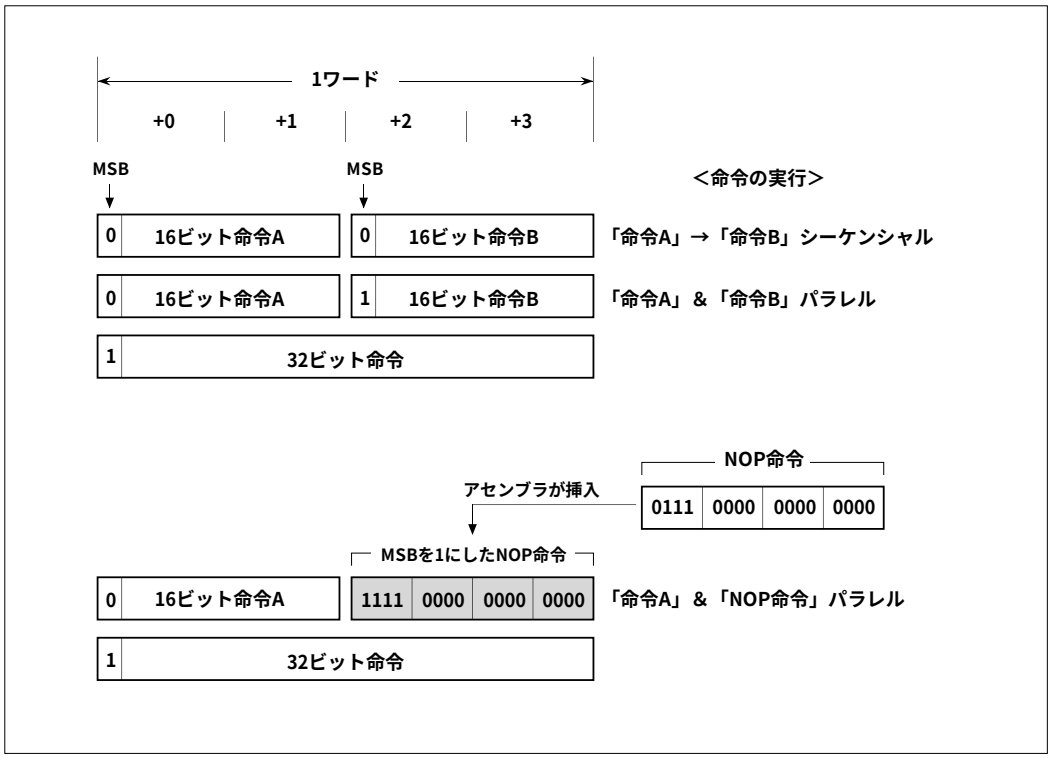


図2.2.3 16ビット命令の処理

第３章

命令

3.1 命令の記述方法

命令詳細説明で記述される各項目は以下のとおりです。

【ニーモニック】

ニーモニックは、命令とそれに続く命令のオペランド(操作対象)の記述で構成されます。

表3.1.1 オペランド一覧

表記(注)	アドレッシングモード	命令の操作対象
R	レジスタ直接	M32Rの汎用レジスタ(R0～R15)の内容
CR	制御レジスタ	M32Rの制御レジスタ(CR0=PSW, CR1=CBR, CR2=SPI, CR3=SPU, CR6=BPC)の内容
@R	レジスタ間接	レジスタの値をアドレスとするメモリの内容
@(disp,R)	レジスタ相対間接	(レジスタの値)＋(16ビットのディスプレースメントを32ビットに符号拡張した値)をアドレスとするメモリの内容
@R+	レジスタ間接 ＋レジスタ更新	レジスタ値を＋4する(更新前のレジスタ値をアドレスとするメモリの内容)
@+R	レジスタ間接 ＋レジスタ更新	レジスタ値を＋4する(更新後のレジスタ値をアドレスとするメモリの内容)
@-R	レジスタ間接 ＋レジスタ更新	レジスタ値を－4する(更新後のレジスタ値をアドレスとするメモリの内容)
#imm	イミディエート	即値(値の扱いは各命令の詳細説明参照)
pcdisp	PC相対	(PCの値)＋(8ビット, 16ビットまたは24ビットのディスプレースメントを32ビットに符号拡張して左へ2ビットシフトした値)をアドレスとするメモリの内容

注. オペランド表記において、Rsrc, Rdestと表記した場合のsrc, destには任意の汎用レジスタ番号(0～15)が入ります。
また、CRsrc, CRdestと表記した場合のsrc, destには任意の制御レジスタ番号(0～3, 6)が入ります。

【動作】

命令の動作概要とC言語に準じた表記での動作説明です。

表3.1.2 動作表記法1(演算子)

演算子	意味
+	加算(二項演算子)
-	減算(二項演算子)
*	乗算(二項演算子)
/	除算(二項演算子)
%	剰余算(二項演算子)
++	インクリメント(単項演算子)
--	デクリメント(単項演算子)

表3.1.3　動作表記法２（演算子）

演算子	意味
-	符号の反転（単項演算子）
=	右辺から左辺への代入（代入演算子）
+=	左辺と右辺の変数を加算し、左辺に代入（代入演算子）
-=	左辺の変数から右辺の変数を減算し、左辺に代入（代入演算子）
>	より大（関係演算子）
<	より小（関係演算子）
>=	より大か等しい（関係演算子）
<=	より小か等しい（関係演算子）
==	等しい（関係演算子）
!=	等しくない（関係演算子）
&&	AND（論理演算子）
||	OR（論理演算子）
!	NOT（論理演算子）
?:	条件式をつくる（条件演算子）

表3.1.4　動作表記法3（ビット演算子）

演算子	意味
<<	ビットを左にシフト
>>	ビットを右にシフト
&	ビット積（AND）
|	ビット和（OR）
^	ビット排他的論理和（EXOR）
~	ビットの反転

表3.1.5　データタイプ

表現	種類	符号の有無	ビット長	数値の範囲
signed char	整数	あり	8	－128～＋127
signed short	整数	あり	16	－32,768～＋32,767
signed int	整数	あり	32	－2,147,483,648～＋2,147,483,647
unsigned char	整数	なし	8	0～255
unsigned short	整数	なし	16	0～655,535
unsigned int	整数	なし	32	0～4,294,967,295
signed64bit	整数	あり	64	符号付き64ビット整数（アキュムレータ操作時）

【機能】

各命令の機能の詳細を説明しています。また、その命令の実行で起こるPSWレジスタ中の条件ビット(C)の変化を記述しています。

【発生EIT】

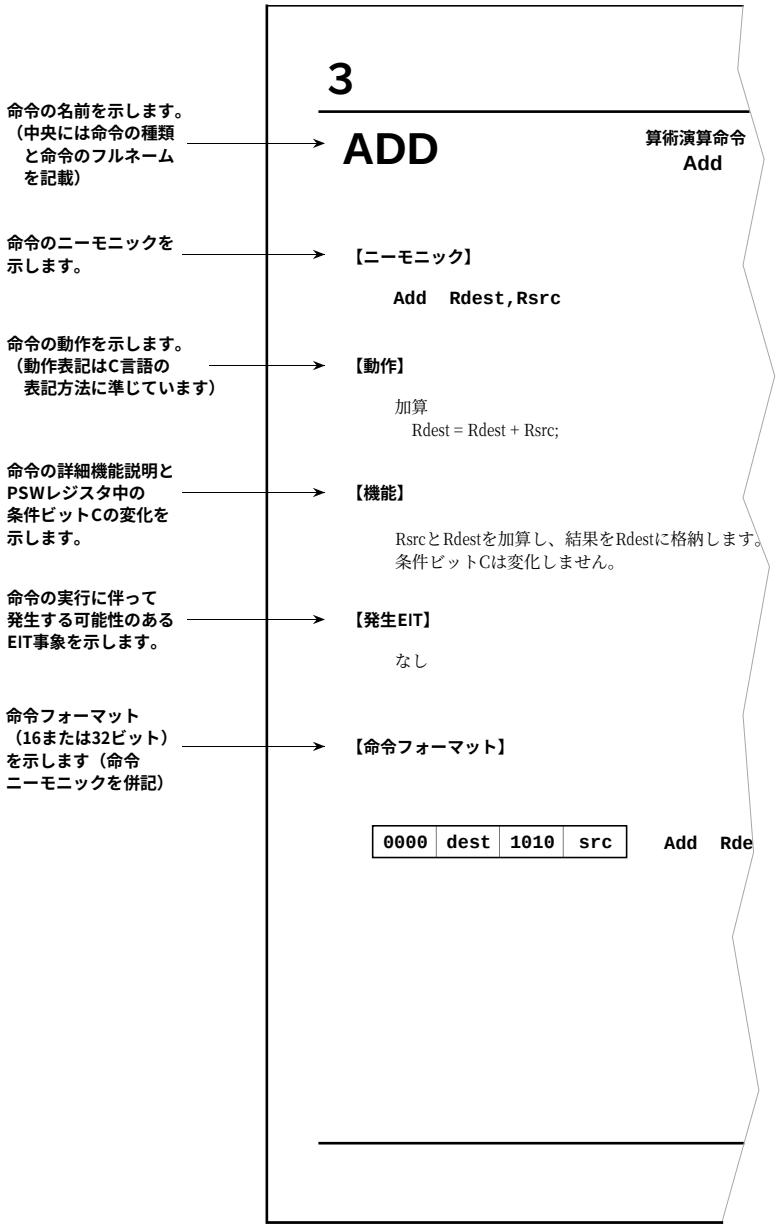
各命令の実行で発生する可能性のあるEIT事象(例外、割り込み、トラップ)を示しています。命令実行で発生する可能性があるのはアドレス例外とトラップです。

【命令フォーマット】

16ビットまたは32ビットの命令ビットパターンを示します。src, destフィールドには対応するレジスタ番号が、また imm, dispにはそれぞれイミディエート(即値)、ディスプレースメントが入ります(代入可能な数値の大きさはフィールドの幅で決まります)。命令フォーマットについては、2.2「 命令フォーマット」を参照ください。

3.2 命令詳細説明

次ページよりM32R CPUの各命令の詳細説明を示します。各命令はアルファベット順に掲載されています。各ページの構成は、下記のとおりです。



ADD

算術演算命令

Add

ADD

【ニーモニック】

```
ADD  Rdest,Rsrc
```

【動作】

加算

Rdest = Rdest + Rsrc;

【機能】

RsrcとRdestを加算し、結果をRdestに格納します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1010	src

ADD Rdest,Rsrc

ADD3

算術演算命令

Add 3-operand

ADD3

【ニーモニック】

ADD3 Rdest,Rsrc,#imm16

【動作】

加算

Rdest = Rsrc + (signed short) imm16;

【機能】

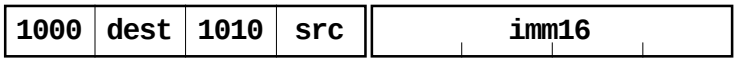
Rsrcに16ビット即値を加算し、結果をRdestに格納します。16ビット即値は、演算前に32ビットに符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



ADD3 Rdest,Rsrc,#imm16

ADDI

算術演算命令

Add immediate

ADDI

【ニーモニック】

```
ADDI  Rdest,#imm8
```

【動作】

加算

Rdest = Rdest + (signed char) imm8;

【機能】

Rdestに8ビット即値を加算し、結果をRdestに格納します。8ビット即値は、演算前に32ビットに符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0100	dest	imm8

ADDI Rdest,#imm8

ADDV

算術演算命令

Add with overflow checking

ADDV

【ニーモニック】

```
ADDV  Rdest,Rsrc
```

【動作】

加算

```
Rdest = ( signed ) Rdest + ( signed ) Rsrc;
C = overflow ? 1 : 0;
```

【機能】

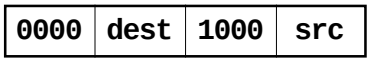
RdestとRsrcを加算し、結果をRdestに格納します。

条件ビット（C）は加算結果がオーバフローした場合にセットされ、それ以外の場合はクリアされます。

【発生EIT】

なし

【命令フォーマット】



ADDV Rdest,Rsrc

ADDV3 *算術演算命令* Add 3-operand with overflow checking ADDV3

【ニーモニック】

```
ADDV3  Rdest,Rsrc,#imm16
```

【動作】

加算

```
Rdest = ( signed ) Rsrc + ( signed ) ( ( signed short ) imm16 );
C = overflow ? 1 : 0;
```

【機能】

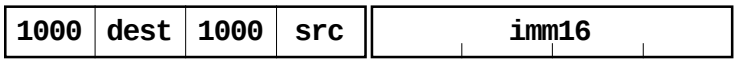
Rsrcと16ビット即値を加算し、結果をRdestに格納します。16ビット即値は演算前に32ビットに符号拡張されます。

条件ビット（C）は加算結果がオーバフローした場合にセットされ、それ以外の場合はクリアされます。

【発生EIT】

なし

【命令フォーマット】




```
ADDV3 Rdest,Rsrc,#imm16
```

ADDX

算術演算命令

Add with carry

ADDX

【ニーモニック】

```
ADDX  Rdest,Rsrc
```

【動作】

加算

Rdest = (unsigned) Rdest + (unsigned) Rsrc + C;
C = carry_out ? 1 : 0;

【機能】

Rdestに（Rsrc＋条件ビット（C）の値）を加算し、結果をRdestに格納します。
条件ビット（C）は加算結果が32ビット符号なし整数で表せないときにセットされ、それ以外の場合はクリアされます。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1001	src

ADDX Rdest,Rsrc

AND

論理演算命令

AND

AND

【ニーモニック】

```
AND  Rdest,Rsrc
```

【動作】

論理積
Rdest = Rdest & Rsrc;

【機能】

RdestとRsrcの対応するビットの論理積をとり、結果をRdestに格納します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1100	src

AND Rdest,Rsrc

AND3

論理演算命令

AND 3-operand

AND3

【ニーモニック】

```
AND3  Rdest,Rsrc,#imm16
```

【動作】

論理積

Rdest = Rsrc & (unsigned short) imm16;

【機能】

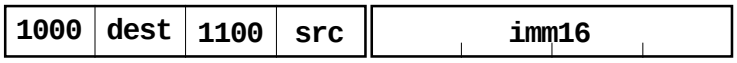
Rsrcと32ビットにゼロ拡張された16ビット即値の対応するビットの論理積をとり、結果をRdestに格納します。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
AND3  Rdest,Rsrc,#imm16
```

BC

分岐命令

Branch on C-bit

BC

【ニーモニック】

① **BC pcdisp8**
② **BC pcdisp24**

【動作】

条件付き分岐
① if (C==1) PC = (PC & 0xfffffffc) + (((signed char) pcdisp8) << 2);
② if (C==1) PC = (PC & 0xfffffffc) + (sign_extend (pcdisp24) << 2);
where
#define sign_extend(x) (((signed) ((x)<< 8)) >>8)

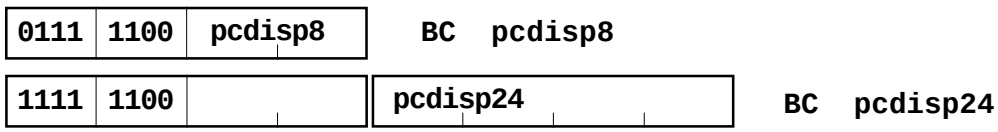
【機能】

条件ビット（C）が1のとき、指定されたラベルへ分岐します。命令フォーマットには2種類ありますが、アセンブラレベルで記述する場合はオペランドに分岐先のラベルを書けば、通常最短のフォーマットが選ばれます。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



BEQ

分岐命令

Branch on equal to

BEQ

【ニーモニック】

```
BEQ  Rsrc1,Rsrc2,pcdisp16
```

【動作】

条件付き分岐

```
if ( Rsrc1 == Rsrc2 )  PC = ( PC & 0xfffffffc ) + ( ( ( signed short ) pcdisp16 ) << 2);
```

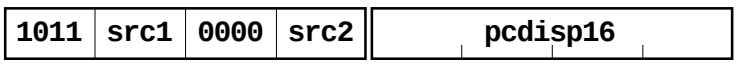
【機能】

Rsrc1とRsrc2が等しいとき、指定されたラベルへ分岐します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
BEQ  Rsrc1,Rsrc2,pcdisp16
```

BEQZ

分岐命令

Branch on equal to zero

BEQZ

【ニーモニック】

```
BEQZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

```
if ( Rsrc == 0 )  PC = ( PC & 0xfffffffc ) + ( ( ( signed short ) pcdisp16 ) << 2);
```

【機能】

Rsrcが0のとき、指定されたラベルへ分岐します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
BEQZ  Rsrc,pcdisp16
```

BGEZ

分岐命令

Branch on greater than or equal to zero

BGEZ

【ニーモニック】

```
BGEZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

if ((signed) Rsrc >= 0) PC = (PC & 0xfffffffc) + (((signed short) pcdisp16) << 2);

【機能】

Rsrcの内容を符号付き32ビット値としてみたとき、0または0より大きい場合に指定されたラベルへ分岐します。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
BGEZ  Rsrc,pcdisp16
```

BGTZ

分岐命令

Branch on greater than zero

BGTZ

【ニーモニック】

```
BGTZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

if ((signed) Rsrc > 0) PC = (PC & 0xfffffffc) + (((signed short) pcdisp16) << 2);

【機能】

Rsrcの内容を符号付き32ビット値としてみたとき、0より大きい場合に指定されたラベルへ分岐します。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
BGTZ  Rsrc,pcdisp16
```

BL

分岐命令

Branch and link

BL

【ニーモニック】

① **BL pcdisp8**
② **BL pcdisp24**

【動作】

サブルーチン呼び出し（PC相対）
① R14 = (PC & 0xfffffffc) + 4;
PC = (PC & 0xfffffffc) + (((signed char) pcdisp8) << 2);
② R14 = (PC & 0xfffffffc) + 4;
PC = (PC & 0xfffffffc) + (sign_extend (pcdisp24) << 2);
where
#define sign_extend(x) (((signed) ((x)<< 8)) >>8)

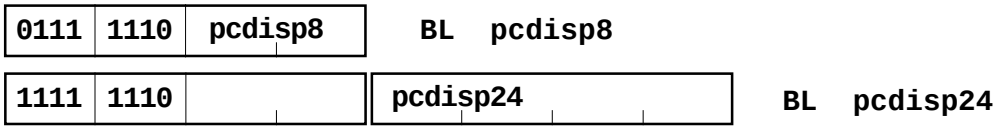
【機能】

無条件分岐を行い、戻り先をR14に格納します。命令フォーマットには2種類ありますが、アセンブラレベルで記述する場合はオペランドに分岐先のラベルを書けば、通常最短のフォーマットが選ばれます。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



BLEZ

分岐命令

Branch on less than or equal to zero

BLEZ

【ニーモニック】

```
BLEZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

if ((signed) Rsrc <= 0) PC = (PC & 0xfffffffc) + (((signed short) pcdisp16) << 2);

【機能】

Rsrcの内容を符号付き32ビット値としてみたとき、0または0より小さい場合に指定されたラベルへ分岐します。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
BLEZ  Rsrc,pcdisp16
```

BLTZ

分岐命令

Branch on less than zero

BLTZ

【ニーモニック】

BLTZ Rsrc,pcdisp16

【動作】

条件付き分岐

if ((signed) Rsrc < 0) PC = (PC & 0xfffffffc) + (((signed short) pcdisp16) << 2);

【機能】

Rsrcの内容を符号付き32ビット値としてみたとき、0より小さい場合に指定されたラベルへ分岐します。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

BLTZ Rsrc,pcdisp16

BNC 分岐命令 Branch on not C-bit BNC

【ニーモニック】

① **BNC pcdisp8**
② **BNC pcdisp24**

【動作】

条件付き分岐
① if (C==0) PC = (PC & 0xfffffffc) + (((signed char) pcdisp8) << 2);
② if (C==0) PC = (PC & 0xfffffffc) + (sign_extend (pcdisp24) << 2);
where
#define sign_extend(x) (((signed) ((x)<< 8)) >>8)

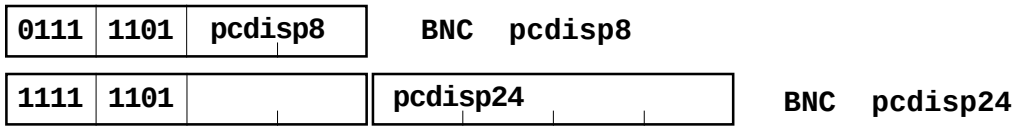
【機能】

条件ビット（C）が0のとき、指定されたラベルへ分岐します。命令フォーマットには2種類ありますが、アセンブラレベルで記述する場合はオペランドに分岐先のラベルを書けば、通常最短のフォーマットが選ばれます。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



BNE

分岐命令

Branch on not equal to

BNE

【ニーモニック】

```
BNE  Rsrc1,Rsrc2,pcdisp16
```

【動作】

条件付き分岐

```
if ( Rsrc1 != Rsrc2 )  PC = ( PC & 0xfffffffc ) + ( ( ( signed short ) pcdisp16 ) << 2);
```

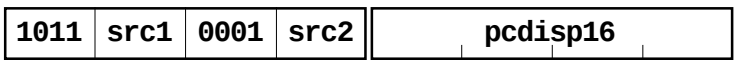
【機能】

Rsrc1とRsrc2が等しくないとき、指定されたラベルへ分岐します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
BNE  Rsrc1,Rsrc2,pcdisp16
```

BNEZ

分岐命令

Branch on not equal to zero

BNEZ

【ニーモニック】

```
BNEZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

```
if ( Rsrc != 0 )  PC = ( PC & 0xfffffffc ) + ( ( ( signed short ) pcdisp16 ) << 2);
```

【機能】

Rsrcが0でないとき、指定されたラベルへ分岐します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
BNEZ Rsrc,pcdisp16
```

BRA

分岐命令
Branch

BRA

【ニーモニック】

① **BRA pcdisp8**
② **BRA pcdisp24**

【動作】

無条件分岐
① PC = (PC & 0xfffffffc) + (((signed char) pcdisp8) << 2);
② PC = (PC & 0xfffffffc) + (sign_extend (pcdisp24) << 2);
where
#define sign_extend(x) (((signed) ((x)<< 8)) >>8)

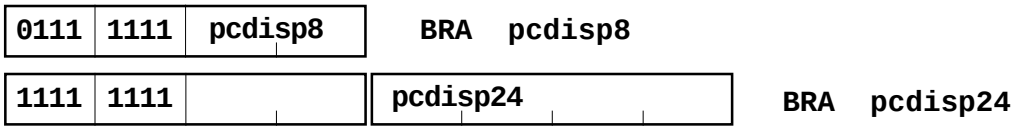
【機能】

指定されたラベルへ無条件分岐します。命令フォーマットには2種類ありますが、アセンブラレベルで記述する場合はオペランドに分岐先のラベルを書けば、通常最短のフォーマットが選ばれます。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



CMP 比較命令 Compare CMP

【ニーモニック】

```
CMP  Rsrc1,Rsrc2
```

【動作】

比較

C = ((signed) Rsrc1 < (signed) Rsrc2) ? 1:0;

【機能】

Rsrc1とRsrc2を比較し、Rsrc1がRsrc2よりも小さいとき条件ビット（C）が1にセットされます。オペランドは符号付き32ビット値として扱われます。

【発生EIT】

なし

【命令フォーマット】

0000	src1	0100	src2

CMP Rsrc1,Rsrc2

CMPI 比較命令 Compare immediate CMPI

【ニーモニック】

```
CMPI  Rsrc,#imm16
```

【動作】

比較

C = ((signed) Rsrc < (signed) ((signed short) imm16)) ? 1:0;

【機能】

Rsrcと16ビット即値を比較し、Rsrcが16ビット即値よりも小さいとき条件ビット（C）が1にセットされます。オペランドは符号付き32ビット値として扱われます。16ビット即値は比較の前に32ビットデータに符号付き拡張されます。

【発生EIT】

なし

【命令フォーマット】


```
CMPI  Rsrc,#imm16
```

CMPU

比較命令
Compare unsigned

CMPU

【ニーモニック】

```
CMPU  Rsrc1,Rsrc2
```

【動作】

比較
C = ((unsigned) Rsrc1 < (unsigned) Rsrc2) ? 1:0;

【機能】

Rsrc1とRsrc2を比較し、Rsrc1がRsrc2よりも小さいとき条件ビット（C）が1にセットされます。オペランドは符号なし32ビット値として扱われます。

【発生EIT】

なし

【命令フォーマット】

0000	src1	0101	src2

CMPU Rsrc1,Rsrc2

CMPUI 比較命令 Compare unsigned immediate CMPUI

【ニーモニック】

```
CMPUI  Rsrc,#imm16
```

【動作】

比較

C = ((unsigned) Rsrc < (unsigned) ((signed short) imm16)) ? 1:0;

【機能】

Rsrcと16ビット即値を比較し、Rsrcが16ビット即値よりも小さいとき条件ビット（C）が1にセットされます。オペランドは符号なし32ビット値として扱われます。16ビット即値は比較の前に32ビットデータに符号付き拡張されます。

【発生EIT】

なし

【命令フォーマット】


```
CMPUI Rsrc,#imm16
```

DIV

乗除算命令
Divide

DIV

【ニーモニック】

```
DIV  Rdest,Rsrc
```

【動作】

符号付き除算
Rdest = (signed) Rdest / (signed) Rsrc;

【機能】

RdestをRsrcで割り算し、商をRdestに格納します。オペランドは符号付き32ビット値として扱われ、商は0方向に丸められます。
条件ビット（C）は変化しません。
Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0000	src	0000	0000	0000	0000

```
DIV  Rdest,Rsrc
```

DIVU

乗除算命令

Divide unsigned

DIVU

【ニーモニック】

```
DIVU  Rdest,Rsrc
```

【動作】

符号なし除算

Rdest = (unsigned) Rdest / (unsigned) Rsrc;

【機能】

RdestをRsrcで割り算し、商をRdestに格納します。オペランドは符号なし32ビット値として扱われ、商は0方向に丸められます。

条件ビット（C）は変化しません。

Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0001	src	0000	0000	0000	0000

```
DIVU  Rdest,Rsrc
```

JL

分岐命令

Jump and link

JL

【ニーモニック】

```
JL  Rsrc
```

【動作】

サブルーチン呼び出し（レジスタ直接）
R14 = (PC & 0xfffffffc) + 4;
PC = Rsrc & 0xfffffffc;

【機能】

Rsrcで指定された番地へ無条件分岐を行い、戻り先をR14に格納します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	1110	1100	src

JL Rsrc

JMP

分岐命令
Jump

JMP

【ニーモニック】

```
JMP  Rsrc
```

【動作】

無条件分岐
PC = Rsrc & 0xfffffffc;

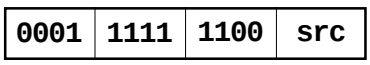
【機能】

Rsrcで指定された番地へ無条件分岐を行います。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



JMP Rsrc

LD

ロード／ストア命令

Load

LD

【ニーモニック】

① **LD Rdest,@Rsrc**
② **LD Rdest,@Rsrc+**
③ **LD Rdest,@(disp16,Rsrc)**

【動作】

メモリからレジスタへのロード
① Rdest = *(signed int *) Rsrc;
② Rdest = *(signed int *) Rsrc, Rsrc += 4;
③ Rdest = *(signed int *) (Rsrc + (signed short) disp16);

【機能】

① Rsrcで指定された番地のメモリ内容をRdestに格納します。
② Rsrcで指定された番地のメモリ内容をRdestに格納し、その後でRsrcを4インクリメントします。
③ Rsrcと16ビットのディスプレースメントで指定された番地のメモリ内容を、Rdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

アドレス例外（AE）

【命令フォーマット】

0010	dest	1100	src	LD Rdest,@Rsrc
0010	dest	1110	src	LD Rdest,@Rsrc+
1010	dest	1100	src	disp16

LD Rdest,@(disp16,Rsrc)

LD24 転送命令 Load 24-bit immediate LD24

【ニーモニック】

```
LD24  Rdest,#imm24
```

【動作】

24ビット即値データの転送
Rdest = imm24 & 0x00ffffff;

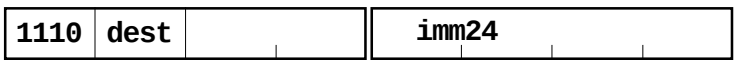
【機能】

24ビットの即値をRdestに格納します。24ビットの即値は32ビットにゼロ拡張されます。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
LD24  Rdest,#imm24
```

LDB

ロード／ストア命令
Load byte

LDB

【ニーモニック】

① **LDB Rdest,@Rsrc**
② **LDB Rdest,@(disp16,Rsrc)**

【動作】

メモリからレジスタへのロード
① Rdest = *(signed char *) Rsrc;
② Rdest = *(signed char *) (Rsrc + (signed short) disp16);

【機能】

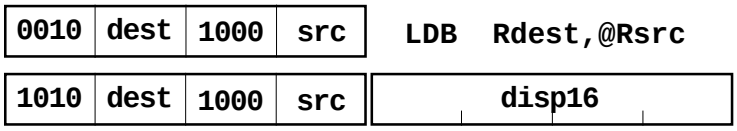
① Rsrcで指定された番地のメモリのバイトデータを、符号拡張してRdestに格納します。
② Rsrcと16ビットのディスプレースメントで指定された番地のメモリのバイトデータを、符号拡張してRdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



LDB Rdest,@(disp16,Rsrc)

LDH

ロード／ストア命令
Load halfword

LDH

【ニーモニック】

① **LDH Rdest,@Rsrc**
② **LDH Rdest,@(disp16,Rsrc)**

【動作】

メモリからレジスタへのロード
① Rdest = *(signed short *) Rsrc;
② Rdest = *(signed short *) (Rsrc + (signed short) disp16);

【機能】

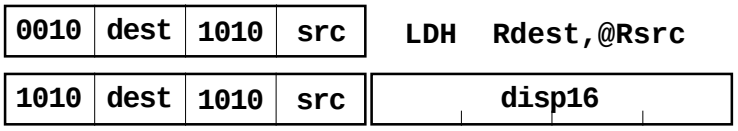
① Rsrcで指定された番地のメモリのハーフワードデータを、符号拡張してRdestに格納します。
② Rsrcと16ビットのディスプレースメントで指定された番地のメモリのハーフワードデータを、符号拡張してRdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

アドレス例外（AE）

【命令フォーマット】



LDH Rdest,@(disp16,Rsrc)

LDI

転送命令

Load immediate

LDI

【ニーモニック】

① **LDI Rdest,#imm8**
② **LDI Rdest,#imm16**

【動作】

即値データの転送
① Rdest = (signed char) imm8;
② Rdest = (signed short) imm16;

【機能】

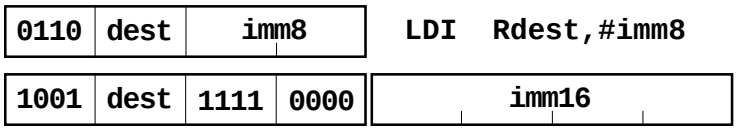
①8ビットの即値をRdestに格納します。8ビットの即値は32ビットに符号拡張されます。
②16ビットの即値をRdestに格納します。16ビットの即値は32ビットに符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



LDI Rdest,#imm16

LDUB

ロード／ストア命令
Load unsigned byte

LDUB

【ニーモニック】

① **LDUB Rdest,@Rsrc**
② **LDUB Rdest,@(disp16,Rsrc)**

【動作】

メモリからレジスタへのロード
① Rdest = *(unsigned char *) Rsrc;
② Rdest = *(unsigned char *) (Rsrc + (signed short) disp16);

【機能】

① Rsrcで指定された番地のメモリのバイトデータを、ゼロ拡張してRdestに格納します。
② Rsrcと16ビットのディスプレースメントで指定された番地のメモリのバイトデータを、ゼロ拡張してRdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0010	dest	1001	src	LDUB Rdest,@Rsrc
1010	dest	1001	src	disp16

LDUB Rdest,@(disp16,Rsrc)

LDUH

ロード／ストア命令

Load unsigned halfword

LDUH

【ニーモニック】

① **LDUH Rdest,@Rsrc**
② **LDUH Rdest,@(disp16,Rsrc)**

【動作】

メモリからレジスタへのロード
① Rdest = *(unsigned short *) Rsrc;
② Rdest = *(unsigned short *) (Rsrc + (signed short) disp16);

【機能】

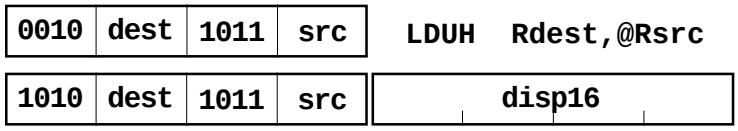
① Rsrcで指定された番地のメモリのハーフワードデータを、ゼロ拡張してRdestに格納します。
② Rsrcと16ビットのディスプレースメントで指定された番地のメモリのハーフワードデータを、ゼロ拡張してRdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

アドレス例外（AE）

【命令フォーマット】



LDUH Rdest,@(disp16,Rsrc)

LOCK *ロード／ストア命令* Load locked LOCK

【ニーモニック】

```
LOCK  Rdest,@Rsrc
```

【動作】

ロック付きロード
LOCK = 1, Rdest = *(signed int *) Rsrc;

【機能】

Rsrcで指定された番地のメモリのワードデータを、Rdestに格納します。
条件ビット（C）は変化しません。
この命令は、通常のロードを行う以外にLOCKビットのセットも行います。
LOCKビットが1のときには、外部バスマスタからのアクセス要求は受け付けません。
LOCKビットのクリアは、UNLOCK命令によって行われます。
LOCKビットはCPUの内部にあり、LOCK命令とUNLOCK命令による操作を除き、ユーザがこのビットを直接アクセスすることはできません。

【発生EIT】

アドレス例外（AE）

【命令フォーマット】

0010	dest	1101	src

LOCK Rdest,@Rsrc

MACHI

DSP機能用命令

Multiply-accumulate high-order halfwords

MACHI

【ニーモニック】

```
MACHI  Rsrc1,Rsrc2
```

【動作】

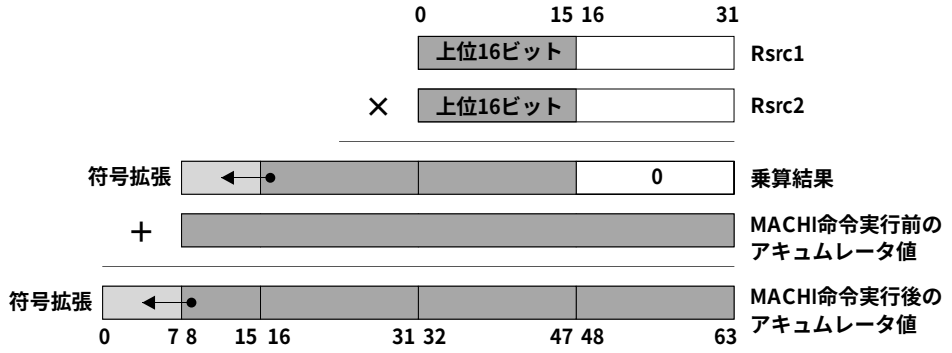
積和演算

accumulator += ((signed) (Rsrc1 & 0xffff0000) * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1の上位16ビットと、Rsrc2の上位16ビットの乗算を行い、乗算結果とアキュムレータの下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはアキュムレータのビット47にあわせ、アキュムレータのビット8～15に対応する部分は符号拡張して加算します。加算結果はアキュムレータに格納されます。Rsrc1の上位16ビットと、Rsrc2の上位16ビットは符号付き整数として扱われます。

条件ビット（C）は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0100	src2

MACHI Rsrc1,Rsrc2

MACLO *DSP機能用命令* Multiply-accumulate low-order halfwords MACLO

【ニーモニック】

```
MACLO  Rsrc1,Rsrc2
```

【動作】

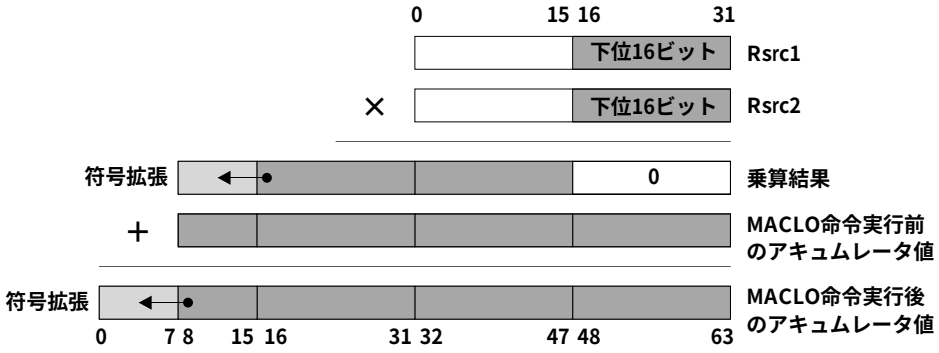
積和演算

accumulator += ((signed) (Rsrc1 << 16) * (signed short) Rsrc2) ;

【機能】

Rsrc1の下位16ビットと、Rsrc2の下位16ビットの乗算を行い、乗算結果とアキュムレータの下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはアキュムレータのビット47にあわせ、アキュムレータのビット8～15に対応する部分は符号拡張して加算します。加算結果はアキュムレータに格納されます。Rsrc1の下位16ビットと、Rsrc2の下位16ビットは符号付き整数として扱われます。

条件ビット（C）は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0101	src2

MACLO Rsrc1,Rsrc2

MACWHI

DSP機能用命令

Multiply-accumulate word and high-order halfword

MACWHI

【ニーモニック】

```
MACWHI  Rsrc1,Rsrc2
```

【動作】

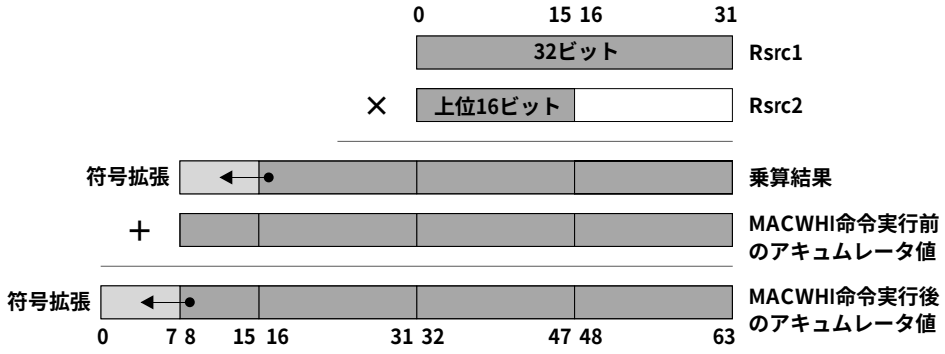
積和演算

accumulator += ((signed) Rsrc1 * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1（32ビット）と、Rsrc2の上位16ビットの乗算を行い、乗算結果とアキュムレータの下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはアキュムレータの最下位ビットにあわせ、アキュムレータのビット8～15に対応する部分は符号拡張して加算します。加算結果はアキュムレータに格納されます。Rsrc1（32ビット）と、Rsrc2の上位16ビットは符号付き整数として扱われます。

条件ビット（C）は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0110	src2

MACWHI Rsrc1,Rsrc2

MACWLO

DSP機能用命令

Multiply-accumulate word and low-order halfword

MACWLO

【ニーモニック】

```
MACWLO  Rsrc1,Rsrc2
```

【動作】

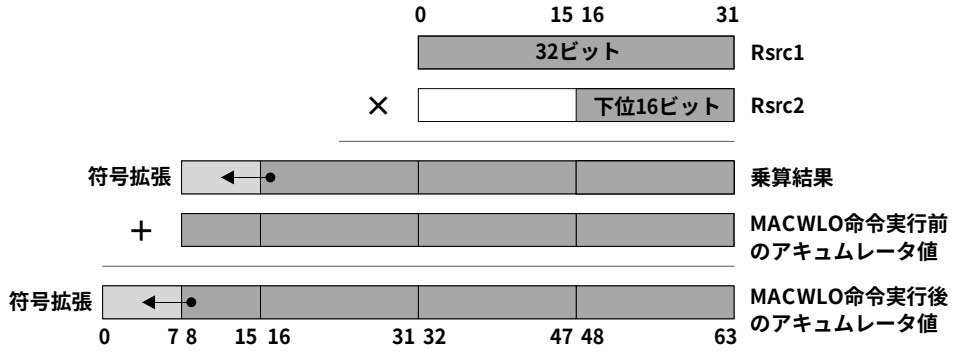
積和演算

accumulator += ((signed) Rsrc1 * (signed short) Rsrc2) ;

【機能】

Rsrc1（32ビット）と、Rsrc2の下位16ビットの乗算を行い、乗算結果とアキュムレータの下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはアキュムレータの最下位ビットにあわせ、アキュムレータのビット8～15に対応する部分は符号拡張して加算します。加算結果はアキュムレータに格納されます。Rsrc1（32ビット）と、Rsrc2の下位16ビットは符号付き整数として扱われます。

条件ビット（C）は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0111	src2

MACWLO Rsrc1,Rsrc2

MUL

乗除算命令
Multiply

MUL

【ニーモニック】

```
MUL  Rdest,Rsrc
```

【動作】

乗算

```
{ signed64bit tmp;
  tmp = ( signed64bit ) Rdest * ( signed64bit ) Rsrc;
  Rdest = ( signed int ) tmp;
}
```

【機能】

RdestとRsrcの乗算を行い、結果をRdestに格納します。オペランドは符号付き整数として扱われます。

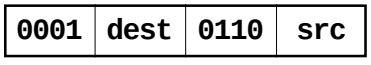
この命令の実行によりアキュムレータの内容は破壊されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
MUL  Rdest,Rsrc
```

MULHI

DSP機能用命令

Multiply high-order halfwords

MULHI

【ニーモニック】

```
MULHI  Rsrc1,Rsrc2
```

【動作】

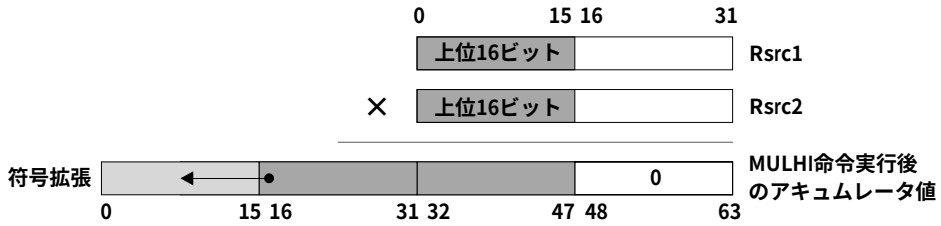
乗算

accumulator = ((signed) (Rsrc1 & 0xffff0000) * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1の上位16ビットと、Rsrc2の上位16ビットの乗算を行い、その結果をアキュムレータに格納します。ただし、乗算結果の最下位ビットはアキュムレータのビット47にあわせ、アキュムレータのビット0～15に対応する部分は符号拡張されます。また、アキュムレータのビット48～63は、ゼロクリアされます。Rsrc1の上位16ビットと、Rsrc2の上位16ビットは符号付き整数として扱われます。

条件ビット（C）は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0000	src2

MULHI Rsrc1,Rsrc2

MULLO *DSP機能用命令* Multiply low-order halfwords MULLO

【ニーモニック】

```
MULLO  Rsrc1,Rsrc2
```

【動作】

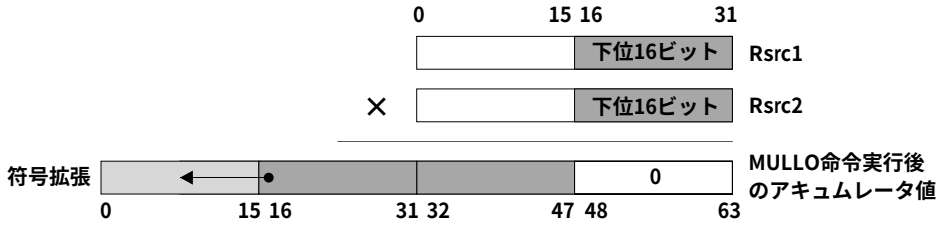
乗算

accumulator = ((signed) (Rsrc1 << 16) * (signed short) Rsrc2);

【機能】

Rsrc1の下位16ビットと、Rsrc2の下位16ビットの乗算を行い、その結果をアキュムレータに格納します。ただし、乗算結果の最下位ビットはアキュムレータのビット47にあわせ、アキュムレータのビット0～15に対応する部分は符号拡張されます。また、アキュムレータのビット48～63は、ゼロクリアされます。Rsrc1の下位16ビットと、Rsrc2の下位16ビットは符号付き整数として扱われます。

条件ビット（C）は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0001	src2

MULLO Rsrc1,Rsrc2

MULWHI

DSP機能用命令

Multiply word and high-order halfword

MULWHI

【ニーモニック】

```
MULWHI  Rsrc1,Rsrc2
```

【動作】

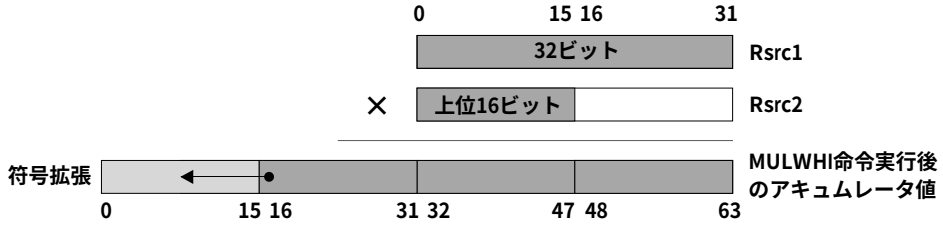
乗算

accumulator = ((signed) Rsrc1 * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1（32ビット）と、Rsrc2の上位16ビットの乗算を行い、結果をアキュムレータに格納します。ただし、乗算結果の最下位ビットはアキュムレータの最下位ビットにあわせ、アキュムレータのビット0～15に対応する部分は符号拡張されます。Rsrc1（32ビット）と、Rsrc2の上位16ビットは符号付き整数として扱われます。

条件ビット（C）は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0010	src2

MULWHI Rsrc1,Rsrc2

MULWLO

DSP機能用命令

Multiply
word and low-order halfword

MULWLO

【ニーモニック】

```
MULWLO  Rsrc1,Rsrc2
```

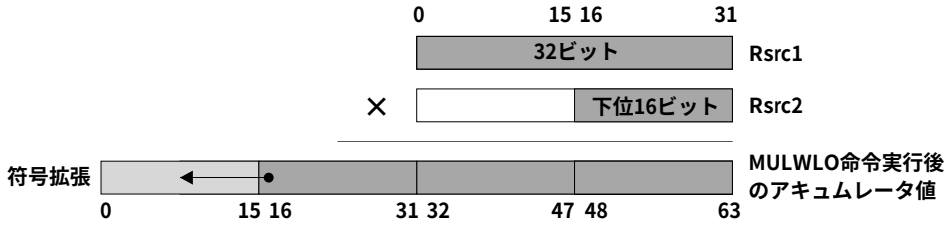
【動作】

乗算
accumulator = ((signed) Rsrc1 * (signed short) Rsrc2);

【機能】

Rsrc1（32ビット）と、Rsrc2の下位16ビットの乗算を行い、結果をアキュムレータに格納します。ただし、乗算結果の最下位ビットはアキュムレータの最下位ビットにあわせ、アキュムレータのビット0～15に対応する部分は符号拡張されます。Rsrc1（32ビット）と、Rsrc2の下位16ビットは符号付き整数として扱われます。

条件ビット（C）は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0011	src2

MULWLO Rsrc1,Rsrc2

MV

転送命令

Move register

MV

【ニーモニック】

```
MV  Rdest,Rsrc
```

【動作】

レジスタ間転送
Rdest = Rsrc;

【機能】

Rsrcの内容をRdestに転送します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	1000	src

MV Rdest,Rsrc

MVFACHI

DSP機能用命令

Move high-order word from accumulator

MVFACHI

【ニーモニック】

```
MVFACHI  Rdest
```

【動作】

アキュムレータ～レジスタ間転送

```
Rdest = ( signed int ) ( accumulator >> 32 ) ;
```

【機能】

アキュムレータの上位32ビットの内容をRdestに転送します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	1111	0000

MVFACHI Rdest

MVFACLO *DSP機能用命令* MVFACLO

Move low-order word from accumulator

【ニーモニック】

```
MVFACLO  Rdest
```

【動作】

アキュムレータ～レジスタ間転送
Rdest = (signed int) accumulator ;

【機能】

アキュムレータの下位32ビットの内容をRdestに転送します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	1111	0001

MVFACLO Rdest

MVFACMI

DSP機能用命令

Move middle-order word from accumulator

MVFACMI

【ニーモニック】

```
MVFACMI  Rdest
```

【動作】

アキュムレータ～レジスタ間転送

```
Rdest = ( signed int ) ( accumulator >> 16 ) ;
```

【機能】

アキュムレータのビット16～47の内容をRdestに転送します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	1111	0010

MVFACMI Rdest

MVFC

転送命令

Move from control register

MVFC

【ニーモニック】

```
MVFC  Rdest,CRsrc
```

【動作】

レジスタ～制御レジスタ間転送
Rdest = CRsrc ;

【機能】

制御レジスタCRsrcの内容をRdestに転送します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	1001	src

MVFC Rdest,CRsrc

MVTACHI

DSP機能用命令

Move high-order word to accumulator

MVTACHI

【ニーモニック】

```
MVTACHI  Rsrc
```

【動作】

アキュムレータ～レジスタ間転送

accumulator [0 : 31] = Rsrc ;

【機能】

Rsrcの内容をアキュムレータの上位32ビット（ビット0～31）に転送します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	src	0111	0000

MVTACHI Rsrc

MVTACLO

DSP機能用命令

Move low-order word to accumulator

MVTACLO

【ニーモニック】

```
MVTACLO  Rsrc
```

【動作】

アキュムレータ～レジスタ間転送

accumulator [32 : 63] = Rsrc ;

【機能】

Rsrcの内容をアキュムレータの下位32ビット（ビット32～63）に転送します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	src	0111	0001

MVTACLO Rsrc

MVTC

転送命令

Move to control register

MVTC

【ニーモニック】

```
MVTC  Rsrc,CRdest
```

【動作】

レジスタ～制御レジスタ間転送
CRdest = Rsrc ;

【機能】

Rsrcの内容を制御レジスタCRdestに転送します。CRdestがPSW（CR0）のとき、条件ビット（C）はその値に依存し、それ以外の場合は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	1010	src

MVTC Rsrc,CRdest

NEG

算術演算命令
Negate

NEG

【ニーモニック】

```
NEG  Rdest,Rsrc
```

【動作】

符号反転
Rdest = 0 －(signed) Rsrc ;

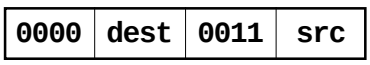
【機能】

Rsrcの内容を符号付き32ビット値として扱い、符号反転して結果をRdestに格納します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



NEG Rdest,Rsrc

NOP

分岐命令

No operation

NOP

【ニーモニック】

NOP

【動作】

ノーオペレーション
/* */

【機能】

処理はなにも行いません。次の命令から継続して実行されます。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	0000	0000	0000

NOP

NOT

論理演算命令

Logical NOT

NOT

【ニーモニック】

```
NOT  Rdest,Rsrc
```

【動作】

論理否定

Rdest = ~ Rsrc ;

【機能】

Rsrcの各ビットの内容を反転して、結果をRdestに格納します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1011	src

NOT Rdest,Rsrc

OR

論理演算命令

OR

OR

【ニーモニック】

```
OR  Rdest,Rsrc
```

【動作】

論理和
Rdest = Rdest ¦ Rsrc ;

【機能】

RdestとRsrcの対応する各ビットの論理和を計算し、結果をRdestに格納します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1110	src

OR Rdest,Rsrc

OR3

論理演算命令

OR 3-operand

OR3

【ニーモニック】

```
OR3  Rdest,Rsrc,#imm16
```

【動作】

論理和

Rdest = Rsrc ¦ (unsigned short) imm16 ;

【機能】

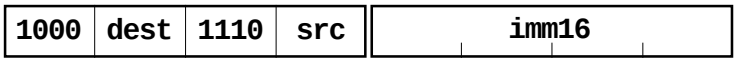
Rsrcと16ビット即値の対応する各ビットの論理和を計算し、結果をRdestに格納します。実行にあたって16ビット即値は32ビットにゼロ拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
OR3  Rdest,Rsrc,#imm16
```

RAC

DSP機能用命令

Round accumulator

RAC

【ニーモニック】

RAC

【動作】

飽和処理
```
{ signed64bit tmp;
tmp = ( signed64bit ) accumulator << 1;
tmp = tmp + 0x0000 0000 0000 8000;
if( 0x0000 7fff ffff 0000 < tmp )
   accumulator = 0x0000 7fff ffff 0000;
else if( tmp < 0xffff 8000 0000 0000 )
   accumulator = 0xffff 8000 0000 0000;
else
   accumulator = tmp & 0xffff ffff ffff 0000;
}
```

【機能】

アキュムレータの値に対してワードサイズで丸めを行い、その結果をアキュムレータに格納します。この命令は2つの処理から構成されています。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	**0000**	**1001**	**0000**

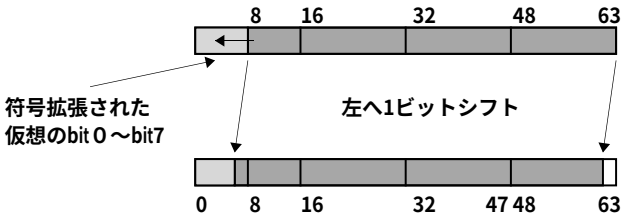
RAC

【機能補足説明】

RAC命令は以下のような処理で構成されています。

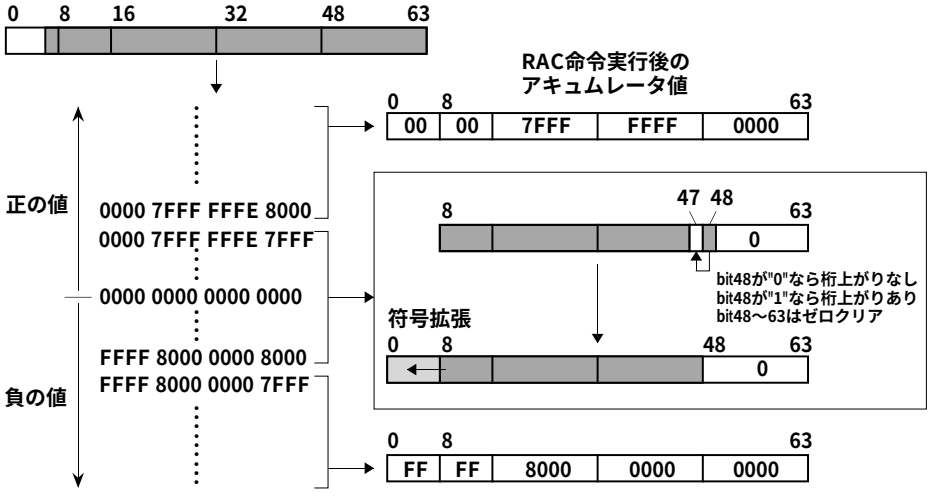
■処理1

アキュムレータの値を、1ビット左シフトします。



■処理2

1ビットの左シフトが反映された仮想のbit0～bit7と、シフト後のbit8～bit63を合わせた64ビットの値にしたがってアキュムレータの値が変化します。



RACH

DSP機能用命令

Round accumulator halfword

RACH

【ニーモニック】

RACH

【動作】

飽和処理
{ signed64bit tmp;
tmp = (signed64bit) accumulator << 1;
tmp = tmp + 0x0000 0000 8000 0000;
if(0x0000 7fff 0000 0000 < tmp)
  accumulator = 0x0000 7fff 0000 0000;
else if(tmp < 0xffff 8000 0000 0000)
  accumulator = 0xffff 8000 0000 0000;
else
  accumulator = tmp & 0xffff ffff 0000 0000;
}

【機能】

アキュムレータの値に対してハーフワードサイズで丸めを行い、その結果をアキュムレータに格納します。この命令は2つの処理から構成されています。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	0000	1000	0000

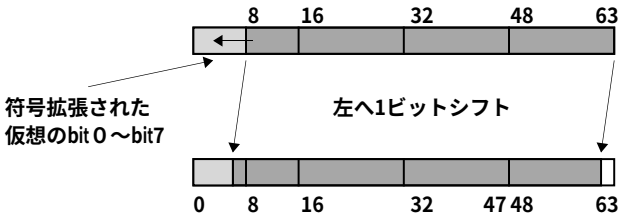
RACH

【機能補足説明】

RACH命令は以下のような処理で構成されています。

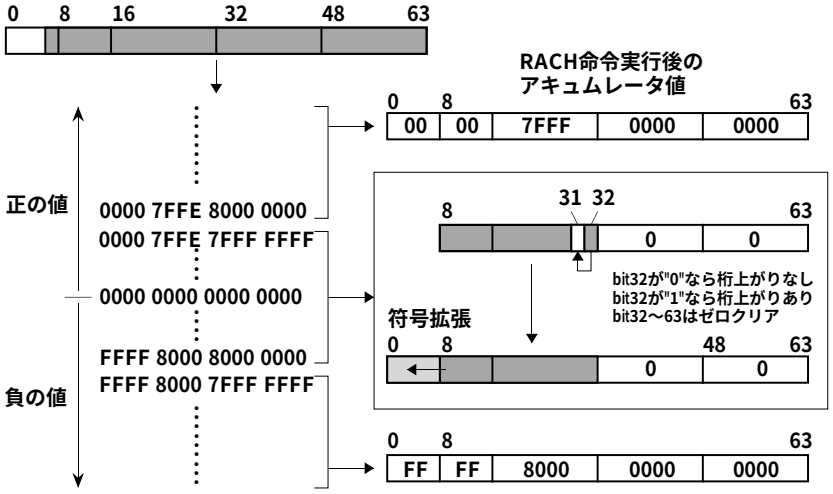
■処理1

アキュムレータの値を、1ビット左シフトします。



■処理2

1ビットの左シフトが反映された仮想のbit0～bit7と、シフト後のbit8～bit63を合わせた64ビットの値にしたがってアキュムレータの値が変化します。



REM

乗除算命令

Remainder

REM

【ニーモニック】

```
REM  Rdest,Rsrc
```

【動作】

符号付き剰余

Rdest = (signed) Rdest % (signed) Rsrc ;

【機能】

RdestをRsrcで除算し、剰余をRdestに格納します。オペランドは符号付き32ビット値として扱われます。商は0方向に丸められ、剰余の符号は被除数と等しくなります。

条件ビット（C）は変化しません。

Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0010	src	0000	0000	0000	0000

```
REM  Rdest,Rsrc
```

REMU 乗除算命令 Remainder unsigned REMU

【ニーモニック】

REMU Rdest,Rsrc

【動作】

符号なし剰余
Rdest = (unsigned) Rdest % (unsigned) Rsrc ;

【機能】

RdestをRsrcで除算し、剰余をRdestに格納します。オペランドは符号なし32ビット値として扱われます。
条件ビット（C）は変化しません。
Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0011	src	0000	0000	0000	0000

REMU Rdest,Rsrc

RTE

EIT関連命令

Return from EIT

RTE

【ニーモニック】

RTE

【動作】

EITハンドラからの復帰
SM = BSM ;
IE = BIE ;
C = BC ;
PC = BPC & 0xfffffffc ;

【機能】

PSWレジスタ中のBSM, BIE および BC ビットの値をそれぞれSM, IE および Cビットに復帰し、BPCで示される番地へ分岐します。

【発生EIT】

なし

【命令フォーマット】

0001	0000	1101	0110

RTE

SETH

転送命令

Set high-order 16-bit

SETH

【ニーモニック】

```
SETH  Rdest,#imm16
```

【動作】

転送命令

Rdest = (signed short) imm16 << 16 ;

【機能】

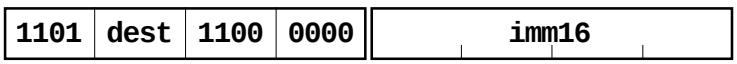
16ビット即値をRdestのMSB側16ビットに転送します。このときLSB側16ビットは0になります。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
SETH  Rdest,#imm16
```

SLL

シフト命令
Shift left logical

SLL

【ニーモニック】

```
SLL  Rdest,Rsrc
```

【動作】

論理左シフト
Rdest = Rdest << (Rsrc & 31) ;

【機能】

Rsrcで指定された値だけRdestの内容を左に論理シフトします。シフトしたLSB側には0が入ります。RsrcのLSB側5ビットのみ有効です。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	0100	src

SLL Rdest,Rsrc

SLL3

シフト命令

Shift left logical 3-operand

SLL3

【ニーモニック】

```
SLL3  Rdest,Rsrc,#imm16
```

【動作】

論理左シフト

```
Rdest = Rsrc  <<  ( imm16 & 31 ) ;
```

【機能】

16ビット即値で指定された値だけRsrcの内容を左に論理シフトします。シフトしたLSB側には0が入ります。16ビット即値のLSB側5ビットのみ有効です。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
SLL3  Rdest,Rsrc,#imm16
```

SLLI

シフト命令

Shift left logical immediate

SLLI

【ニーモニック】

```
SLLI  Rdest,#imm5
```

【動作】

論理左シフト

Rdest = Rdest << imm5 ;

【機能】

5ビット即値で指定された値だけRdestの内容を左に論理シフトます。シフトしたLSB側には0が入ります。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	010	imm5

SLLI Rdest,#imm5

SRA

シフト命令

Shift right arithmetic

SRA

【ニーモニック】

```
SRA  Rdest,Rsrc
```

【動作】

算術右シフト

```
Rdest = ( signed ) Rdest  >>  ( Rsrc & 31 ) ;
```

【機能】

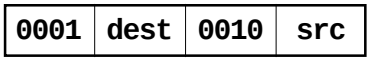
Rsrcで指定された値だけRdestの内容を右に算術シフトします。シフトして空いた側にはMSBにある符号ビットがコピーされて入り、結果はRdestに格納されます。RsrcはLSB側5ビットのみ有効です。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
SRA  Rdest,Rsrc
```

SRA3 シフト命令 Shift right arithmetic 3-operand SRA3

【ニーモニック】

```
SRA3  Rdest,Rsrc,#imm16
```

【動作】

算術右シフト

Rdest = (signed) Rsrc >> (imm16 & 31) ;

【機能】

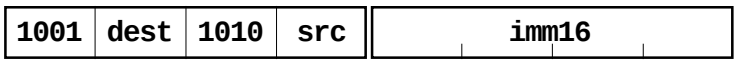
16ビット即値で指定された値だけRsrcの内容を右に算術シフトします。シフトして空いた側にはMSBにある符号ビットがコピーされて入り、結果はRdestに格納されます。16ビット即値のLSB側5ビットのみ有効です。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
SRA3  Rdest,Rsrc,#imm16
```

SRAI

シフト命令

Shift right arithmetic immediate

SRAI

【ニーモニック】

```
SRAI  Rdest,#imm5
```

【動作】

算術右シフト

Rdest = (signed) Rdest >> imm5 ;

【機能】

5ビット即値で指定された値だけRdestの内容を右に算術シフトします。シフトして空いた側にはMSBにある符号ビットがコピーされて入り、結果はRdestに格納されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	001	imm5

SRAI Rdest,#imm5

SRL

シフト命令
Shift right logical

SRL

【ニーモニック】

```
SRL   Rdest,Rsrc
```

【動作】

論理右シフト

```
Rdest = ( unsigned ) Rdest  >>  ( Rsrc & 31 ) ;
```

【機能】

Rsrcで指定された値だけRdestの内容を右に論理シフトします。シフトしたMSB側には0が入ります。RsrcのLSB側5ビットのみ有効です。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	0000	src

SRL Rdest,Rsrc

SRL3

シフト命令
Shift right logical 3-operand

SRL3

【ニーモニック】

```
SRL3  Rdest,Rsrc,#imm16
```

【動作】

論理右シフト
Rdest = (unsigned) Rsrc >> (imm16 & 31) ;

【機能】

16ビット即値で指定された値だけRsrcの内容を右に論理シフトします。シフトしたMSB側には0が入ります。16ビット即値のLSB側5ビットのみ有効です。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
SRL3  Rdest,Rsrc,#imm16
```

SRLI

シフト命令

Shift right logical immediate

SRLI

【ニーモニック】

```
SRLI  Rdest,#imm5
```

【動作】

論理右シフト

Rdest = (unsigned) Rdest >> (imm5 & 31) ;

【機能】

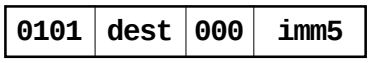
5ビット即値で指定された値だけRdestの内容を右にシフトします。シフトしたMSB側には0が入り、結果はRdestに格納されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



SRLI Rdest,#imm5

ST

ロード／ストア命令

Store

ST

【ニーモニック】

① **ST Rsrc1,@Rsrc2**
② **ST Rsrc1,@+Rsrc2**
③ **ST Rsrc1,@-Rsrc2**
④ **ST Rsrc1,@(disp16,Rsrc2)**

【動作】

レジスタからメモリへのストア
① * (signed int *) Rsrc2 = Rsrc1;
② Rsrc2 += 4, * (signed int *) Rsrc2 = Rsrc1;
③ Rsrc2 -= 4, * (signed int *) Rsrc2 = Rsrc1;
④ * (signed int *) (Rsrc2 + (signed short) disp16) = Rsrc1;

【機能】

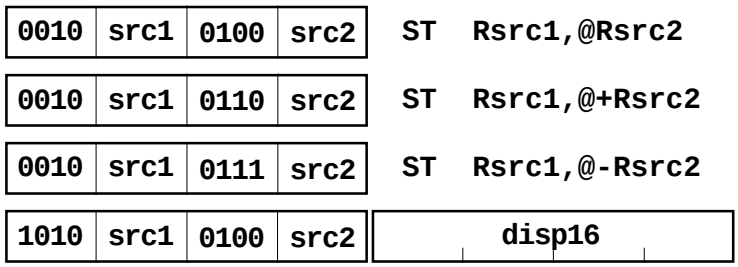
① Rsrc1の内容を、Rsrc2で指定する番地のメモリにストアします。
② Rsrc2を4インクリメントした後、Rsrc1の内容を、インクリメントしたRsrc2で指定する番地のメモリにストアします。
③ Rsrc2を4デクリメントした後、Rsrc1の内容を、デクリメントしたRsrc2で指定する番地のメモリにストアします。
④ Rsrc1の内容を、Rsrc2と16ビットのディスプレースメントで指定する番地のメモリにストアします。ディスプレースメントの値は、アドレス計算の前に符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

アドレス例外（AE）

【命令フォーマット】



ST Rsrc1,@(disp16,Rsrc2)

STB

ロード／ストア命令
Store byte

STB

【ニーモニック】

① **STB Rsrc1,@Rsrc2**
② **STB Rsrc1,@(disp16,Rsrc2)**

【動作】

レジスタからメモリへのストア
① * (signed char *) Rsrc2 = Rsrc1;
② * (signed char *) (Rsrc2 + (signed short) disp16) = Rsrc1;

【機能】

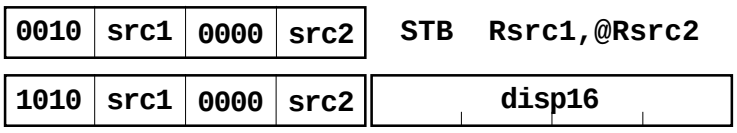
① Rsrc1のLSB側バイトデータを、Rsrc2で指定する番地のメモリにストアします。
② Rsrc1のLSB側バイトデータを、Rsrc2と16ビットディスプレースメントで指定する番地のメモリにストアします。ディスプレースメントは、アドレス計算の前に符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】



STB Rsrc1,@(disp16,Rsrc2)

STH

ロード／ストア命令
Store halfword

STH

【ニーモニック】

① **STH Rsrc1,@Rsrc2**
② **STH Rsrc1,@(disp16,Rsrc2)**

【動作】

レジスタからメモリへのストア
① * (signed short *) Rsrc2 = Rsrc1;
② * (signed short *) (Rsrc2 + (signed short) disp16) = Rsrc1;

【機能】

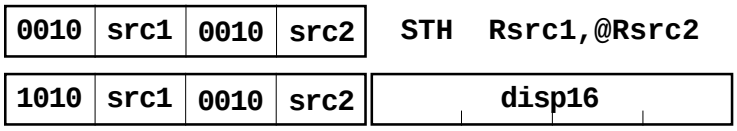
① Rsrc1のLSB側ハーフワードデータを、Rsrc2で指定する番地のメモリにストアします。
② Rsrc1のLSB側ハーフワードデータを、Rsrc2と16ビットディスプレースメントで指定する番地のメモリにストアします。ディスプレースメントは、アドレス計算の前に符号拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

アドレス例外（AE）

【命令フォーマット】



STH Rsrc1,@(disp16,Rsrc2)

SUB

算術演算命令

Subtract

SUB

【ニーモニック】

SUB Rdest,Rsrc

【動作】

減算
Rdest = Rdest - Rsrc;

【機能】

Rdestの内容からRsrcの内容を引き算し、結果をRdestに格納します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	0010	src

SUB Rdest,Rsrc

SUBV

算術演算命令

Subtract with overflow checking

SUBV

【ニーモニック】

```
SUBV  Rdest,Rsrc
```

【動作】

減算（オーバフローチェック付き）

```
Rdest = ( signed ) Rdest - ( signed ) Rsrc;
C = overflow ? 1 : 0;
```

【機能】

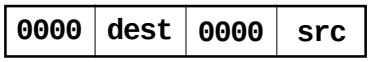
Rdestの内容からRsrcの内容を引き算し、結果をRdestに格納します。

条件ビット（C）は引き算の結果がオーバフローしたときセットされ、それ以外はクリアされます。

【発生EIT】

なし

【命令フォーマット】



SUBV Rdest,Rsrc

SUBX

算術演算命令

Subtract with borrow

SUBX

【ニーモニック】

```
SUBX  Rdest,Rsrc
```

【動作】

減算（ボロー付き）

```
Rdest = ( unsigned ) Rdest - ( unsigned ) Rsrc - C;
C = borrow ? 1 : 0;
```

【機能】

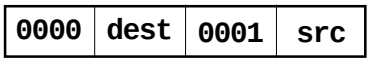
Rdestの内容から（Rsrcの内容＋条件ビット（C））の値を引き算し、結果をRdestに格納します。

条件ビット（C）は引き算の結果が符号なし32ビット整数として表現できないときセットされ、それ以外はクリアされます。

【発生EIT】

なし

【命令フォーマット】



SUBX Rdest,Rsrc

TRAP

EIT関連命令
Trap

TRAP

【ニーモニック】

```
TRAP  #imm4
```

【動作】

トラップの発生
```
BPC = PC + 4;
BSM = SM;
BIE = IE;
BC = C;
IE = 0;
C = 0;
call_trap_handler ( imm4 );
```

【機能】

指定された番号のトラップを発生します。
PSWレジスタ中のSM, IEおよびCビットの値をそれぞれBSM, BIEおよびBCビットに退避し、IEおよびCビットをそれぞれ"0"に更新します。

【発生EIT】

トラップ（TRAP）

【命令フォーマット】

0001	0000	1111	imm4

```
TRAP  #imm4;
```

UNLOCK　ロード／ストア命令 Store unlocked　UNLOCK

【ニーモニック】

```
UNLOCK  Rsrc1,@Rsrc2
```

【動作】

ロック解除付きストア
if (LOCK == 1) { * (signed int *) Rsrc2 = Rsrc1; }
LOCK = 0;

【機能】

LOCKビットが1のとき、Rsrc1をRsrc2で指定された番地のメモリにストアして、LOCKビットをクリアします。
条件ビット（C）は変化しません。
LOCKビットが0のときは、ストア動作を行いません。
LOCKビットはCPUの内部にあり、LOCK命令とUNLOCK命令による操作を除き、ユーザがこのビットを直接アクセスすることはできません。

【発生EIT】

アドレス例外（AE）

【命令フォーマット】

0010	src1	0101	src2

UNLOCK Rsrc1,@Rsrc2

XOR

論理演算命令
Exclusive OR

XOR

【ニーモニック】

```
XOR  Rdest,Rsrc
```

【動作】

排他的論理和
Rdest = Rdest ^ Rsrc;

【機能】

RdestとRsrcの対応するビットの排他的論理和を計算し、Rdestに格納します。
条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1101	src

XOR Rdest,Rsrc

XOR3

論理演算命令

Exclusive OR 3-operand

XOR3

【ニーモニック】

```
XOR3  Rdest,Rsrc,#imm16
```

【動作】

排他的論理和

```
Rdest = Rsrc ^ ( unsigned short ) imm16;
```

【機能】

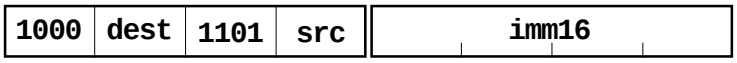
Rsrcと16ビット即値の対応するビットの排他的論理和を計算し、Rdestに格納します。16ビット即値は演算前に32ビットにゼロ拡張されます。

条件ビット（C）は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
XOR3  Rdest,Rsrc,#imm16
```

＊空きページです＊

付録

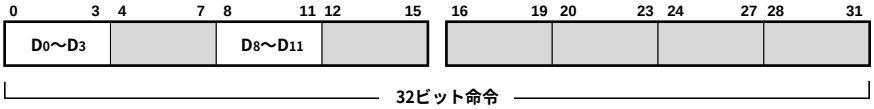
付録1 16進命令コード対応表
付録2 命令セット一覧
付録3 パイプライン処理機構
付録4 命令処理時間
付録5 注意事項

付録1　16進命令コード対応表

M32Rファミリの各命令のビットパターンとニーモニックの対応を以下に示します。

付表1.1　M32Rファミリ　命令コード対応表

	D8～D11	0000	0001	0010	0011	0100	0101	0110	0111
	D0～D3 / 16進表記	0	1	2	3	4	5	6	7
16ビット命令	0000	0	**SUBV** Rdest,Rsrc	**SUBX** Rdest,Rsrc	**SUB** Rdest,Rsrc	**NEG** Rdest,Rsrc	**CMP** Rsrc1,Rsrc2	**CMPU** Rsrc1,Rsrc2		
	0001	1	**SRL** Rdest,Rsrc		**SRA** Rdest,Rsrc		**SLL** Rdest,Rsrc		**MUL** Rdest,Rsrc	
	0010	2	**STB** Rsrc1,@Rsrc2		**STH** Rsrc1,@Rsrc2		**ST** Rsrc1,@Rsrc2	**UNLOCK** Rsrc1,@Rsrc2	**ST** Rsrc1,@+Rsrc2	**ST** Rsrc1,@-Rsrc2
	0011	3	**MULHI** Rsrc1,Rsrc2	**MULLO** Rsrc1,Rsrc2	**MULWHI** Rsrc1,Rsrc2	**MULWLO** Rsrc1,Rsrc2	**MACHI** Rsrc1,Rsrc2	**MACLO** Rsrc1,Rsrc2	**MACWHI** Rsrc1,Rsrc2	**MACWLO** Rsrc1,Rsrc2
	0100	4	**ADDI** Rdest,#imm8							
	0101	5	**SRLI** Rdest,#imm5		**SRAI** Rdest,#imm5		**SLLI** Rdest,#imm5			**MVTACHI, MVTACLO (★2)**
	0110	6	**LDI** Rdest,#imm8							
	0111	7	**NOP (★1)**	**BC, BNC, BL, BRA (★1)**						
32ビット命令	1000	8					**CMPI** Rsrc,#imm16	**CMPUI** Rsrc,#imm16		
	1001	9	**DIV** Rdest,Rsrc	**DIVU** Rdest,Rsrc	**REM** Rdest,Rsrc	**REMU** Rdest,Rsrc				
	1010	A	**STB** Rsrc1,@(disp16,Rsrc2)		**STH** Rsrc1,@(disp16,Rsrc2)		**ST** Rsrc1,@(disp16,Rsrc2)			
	1011	B	**BEQ** Rsrc1,Rsrc2,pcdisp16	**BNE** Rsrc1,Rsrc2,pcdisp16						
	1100	C								
	1101	D								
	1110	E	**LD24** Rdest,#imm24							
	1111	F	**BC, BNC, BL, BRA (★1)**							



1000	1001	1010	1011	1100	1101	1110	1111	D8～D11 / 16進表記	D0～D3	
8	9	A	B	C	D	E	F			
ADDV Rdest,Rsrc	**ADDX** Rdest,Rsrc	**ADD** Rdest,Rsrc	**NOT** Rdest,Rsrc	**AND** Rdest,Rsrc	**XOR** Rdest,Rsrc	**OR** Rdest,Rsrc		0	0000	16ビット命令
MV Rdest,Rsrc	**MVFC** Rdest,CRsrc	**MVTC** Rsrc,CRdest		**JL, JMP** (★1)	**RTE**		**TRAP** #imm4	1	0001	
LDB Rdest,@Rsrc	**LDUB** Rdest,@Rsrc	**LDH** Rdest,@Rsrc	**LDUH** Rdest,@Rsrc	**LD** Rdest,@Rsrc	**LOCK** Rdest,@Rsrc	**LD** Rdest,@Rsrc+		2	0010	
								3	0011	
ADDI Rdest,#imm8								4	0100	
RACH	**RAC**						**MVFACHI, MVFACLO, MVFACMI** (★2)	5	0101	
LDI Rdest,#imm8								6	0110	
BC, BNC, BL, BRA (★1)								7	0111	
ADDV3 Rdest,Rsrc,#imm16		**ADD3** Rdest,Rsrc,#imm16		**AND3** Rdest,Rsrc,#imm16	**XOR3** Rdest,Rsrc,#imm16	**OR3** Rdest,Rsrc,#imm16		8	1000	32ビット命令
SRL3 Rdest,Rsrc,#imm16		**SRA3** Rdest,Rsrc,#imm16		**SLL3** Rdest,Rsrc,#imm16			**LDI** Rdest,#imm16	9	1001	
LDB Rdest,@(disp16,Rsrc)	**LDUB** Rdest,@(disp16,Rsrc)	**LDH** Rdest,@(disp16,Rsrc)	**LDUH** Rdest,@(disp16,Rsrc)	**LD** Rdest,@(disp16,Rsrc)				A	1010	
BEQZ Rsrc,pcdisp16	**BNEZ** Rsrc,pcdisp16	**BLTZ** Rsrc,pcdisp16	**BGEZ** Rsrc,pcdisp16	**BLEZ** Rsrc,pcdisp16	**BGTZ** Rsrc,pcdisp16			B	1011	
								C	1100	
				SETH Rdest,#imm16				D	1101	
LD24 Rdest,#imm24								E	1110	
BC, BNC, BL, BRA (★1)								F	1111	

注. 表中の★1～2で示された命令は、D0～D3、およびD8～D11以外に、下記に示すビットのパターンにより命令が決定されます。各命令のビットパターンの詳細は、3.2「命令詳細説明」の該当命令のページを参照してください。

★1. D4～D7、★2. D12～D15

付録2　命令セット一覧

M32Rファミリの命令セット一覧を以下に示します(アルファベット順)。

ニーモニック		動作	条件ビット(C)
ADD	Rdest,Rsrc	Rdest = Rdest + Rsrc	—
ADD3	Rdest,Rsrc,#imm16	Rdest = Rsrc + (sh)imm16	—
ADDI	Rdest,#imm8	Rdest = Rdest + (sb)imm8	—
ADDV	Rdest,Rsrc	Rdest = Rdest + Rsrc	変化
ADDV3	Rdest,Rsrc,#imm16	Rdest = Rsrc + (sh)imm16	変化
ADDX	Rdest,Rsrc	Rdest = Rdest + Rsrc + C	変化
AND	Rdest,Rsrc	Rdest = Rdest & Rsrc	—
AND3	Rdest,Rsrc,#imm16	Rdest = Rsrc & (uh)imm16	—
BC	pcdisp8	if(C) PC=PC+((sb)pcdisp8<<2)	—
BC	pcdisp24	if(C) PC=PC+((s24)pcdisp24<<2)	—
BEQ	Rsrc1,Rsrc2,pcdisp16	if(Rsrc1 == Rsrc2) PC=PC+((sh)pcdisp16<<2)	—
BEQZ	Rsrc,pcdisp16	if(Rsrc == 0) PC=PC+((sh)pcdisp16<<2)	—
BGEZ	Rsrc,pcdisp16	if(Rsrc >= 0) PC=PC+((sh)pcdisp16<<2)	—
BGTZ	Rsrc,pcdisp16	if(Rsrc > 0) PC=PC+((sh)pcdisp16<<2)	—
BL	pcdisp8	R14=PC+4,PC=PC+((sb)pcdisp8<<2)	—
BL	pcdisp24	R14=PC+4,PC=PC+((s24)pcdisp24<<2)	—
BLEZ	Rsrc,pcdisp16	if(Rsrc <= 0) PC=PC+((sh)pcdisp16<<2)	—
BLTZ	Rsrc,pcdisp16	if(Rsrc < 0) PC=PC+((sh)pcdisp16<<2)	—
BNC	pcdisp8	if(!C) PC=PC+((sb)pcdisp8<<2)	—
BNC	pcdisp24	if(!C) PC=PC+((s24)pcdisp24<<2)	—
BNE	Rsrc1,Rsrc2,pcdisp16	if(Rsrc1 != Rsrc2) PC=PC+((sh)pcdisp16<<2)	—
BNEZ	Rsrc,pcdisp16	if(Rsrc != 0) PC=PC+((sh)pcdisp16<<2)	—
BRA	pcdisp8	PC=PC+((sb)pcdisp8<<2)	—
BRA	pcdisp24	PC=PC+((s24)pcdisp24<<2)	—
CMP	Rsrc1,Rsrc2	(s)Rsrc1 < (s)Rsrc2	変化
CMPI	Rsrc,#imm16	(s)Rsrc < (sh)imm16	変化
CMPU	Rsrc1,Rsrc2	(u)Rsrc1 < (u)Rsrc2	変化
CMPUI	Rsrc,#imm16	(u)Rsrc < (u)((sh)imm16)	変化
DIV	Rdest,Rsrc	Rdest = (s)Rdest / (s)Rsrc	—
DIVU	Rdest,Rsrc	Rdest = (u)Rdest / (u)Rsrc	—
JL	Rsrc	R14 = PC+4, PC = Rsrc	—
JMP	Rsrc	PC = Rsrc	—
LD	Rdest,@(disp16,Rsrc)	Rdest = *(s *)(Rsrc+(sh)disp16)	—
LD	Rdest,@Rsrc	Rdest = *(s *)Rsrc	—
LD	Rdest,@Rsrc+	Rdest = *(s *)Rsrc, Rsrc += 4	—

ニーモニック		動作	条件ビット(C)
LD24	Rdest,#imm24	Rdest = imm24 & 0x00ffffff	—
LDB	Rdest,@(disp16,Rsrc)	Rdest = *(sb *)(Rsrc+(sh)disp16)	—
LDB	Rdest,@Rsrc	Rdest = *(sb *)Rsrc	—
LDH	Rdest,@(disp16,Rsrc)	Rdest = *(sh *)(Rsrc+(sh)disp16)	—
LDH	Rdest,@Rsrc	Rdest = *(sh *)Rsrc	—
LDI	Rdest,#imm16	Rdest = (sh)imm16	—
LDI	Rdest,#imm8	Rdest = (sb)imm8	—
LDUB	Rdest,@(disp16,Rsrc)	Rdest = *(ub *)(Rsrc+(sh)disp16)	—
LDUB	Rdest,@Rsrc	Rdest = *(ub *)Rsrc	—
LDUH	Rdest,@(disp16,Rsrc)	Rdest = *(uh *)(Rsrc+(sh)disp16)	—
LDUH	Rdest,@Rsrc	Rdest = *(ub *)Rsrc	—
LOCK	Rdest,@Rsrc	LOCK = 1, Rdest = *(s *)Rsrc	—
MACHI	Rsrc1,Rsrc2	accumulator += (s)(Rsrc1 & 0xffff0000) * (s)((s)Rsrc2>>16)	—
MACLO	Rsrc1,Rsrc2	accumulator += (s)(Rsrc1<<16) * (sh)Rsrc2	—
MACWHI	Rsrc1,Rsrc2	accumulator += (s)Rsrc1 * (s)((s)Rsrc2>>16)	—
MACWLO	Rsrc1,Rsrc2	accumulator += (s)Rsrc1 * (sh)Rsrc2	—
MUL	Rdest,Rsrc	Rdest = (s)Rdest * (s)Rsrc	—
MULHI	Rsrc1,Rsrc2	accumulator = (s)(Rsrc1 & 0xffff0000) * (s)((s)Rsrc2>>16)	—
MULLO	Rsrc1,Rsrc2	accumulator = (s)(Rsrc1<<16) * (sh)Rsrc2	—
MULWHI	Rsrc1,Rsrc2	accumulator = (s)Rsrc1 * (s)((s)Rsrc2>>16)	—
MULWLO	Rsrc1,Rsrc2	accumulator = (s)Rsrc1 * (sh)Rsrc2	—
MV	Rdest,Rsrc	Rdest = Rsrc	—
MVFACHI	Rdest	Rdest = accumulater >> 32	—
MVFACLO	Rdest	Rdest = accumulator	—
MVFACMI	Rdest	Rdest = accumulator >> 16	—
MVFC	Rdest,CRsrc	Rdest = CRsrc	—
MVTACHI	Rsrc	accumulator[0:31] = Rsrc	—
MVTACLO	Rsrc	accumulator[32:63] = Rsrc	—
MVTC	Rsrc,CRdest	CRdest = Rsrc	変化
NEG	Rdest,Rsrc	Rdest = 0 - Rsrc	—
NOP		/*no-operation*/	—
NOT	Rdest,Rsrc	Rdest = ~Rsrc	—
OR	Rdest,Rsrc	Rdest = Rdest | Rsrc	—
OR3	Rdest,Rsrc,#imm16	Rdest = Rsrc | (uh)imm16	—
RAC		Round the 32-bit value in the accumulator	—
RACH		Round the 16-bit value in the accumulator	—
REM	Rdest,Rsrc	Rdest = (s)Rdest % (s)Rsrc	—
REMU	Rdest,Rsrc	Rdest = (u)Rdest % (u)Rsrc	—
RTE		PC = BPC & 0xfffffffc, PSW[SM,IE,C] = PSW[BSM,BIE,BC]	変化

ニーモニック		動作	条件ビット(C)
SETH	Rdest,#imm16	Rdest = imm16 << 16	—
SLL	Rdest,Rsrc	Rdest = Rdest << (Rsrc & 31)	—
SLL3	Rdest,Rsrc,#imm16	Rdest = Rsrc << (imm16 & 31)	—
SLLI	Rdest,#imm5	Rdest = Rdest << imm5	—
SRA	Rdest,Rsrc	Rdest = (s)Rdest >> (Rsrc & 31)	—
SRA3	Rdest,Rsrc,#imm16	Rdest = (s)Rsrc >> (imm16 & 31)	—
SRAI	Rdest,#imm5	Rdest = (s)Rdest >> imm5	—
SRL	Rdest,Rsrc	Rdest = (u)Rdest >> (Rsrc & 31)	—
SRL3	Rdest,Rsrc,#imm16	Rdest = (u)Rsrc >> (imm16 & 31)	—
SRLI	Rdest,#imm5	Rdest = (u)Rdest >> imm5	—
ST	Rsrc1,@(disp16,Rsrc2)	*(s *)(Rsrc2+(sh)disp16) = Rsrc1	—
ST	Rsrc1,@+Rsrc2	Rsrc2 += 4, *(s *)Rsrc2 = Rsrc1	—
ST	Rsrc1,@-Rsrc2	Rsrc2 -= 4, *(s *)Rsrc2 = Rsrc1	—
ST	Rsrc1,@Rsrc2	*(s *)Rsrc2 = Rsrc1	—
STB	Rsrc1,@(disp16,Rsrc2)	*(sb *)(Rsrc2+(sh)disp16) = Rsrc1	—
STB	Rsrc1,@Rsrc2	*(sb *)Rsrc2 = Rsrc1	—
STH	Rsrc1,@(disp16,Rsrc2)	*(sh *)(Rsrc2+(sh)disp16) = Rsrc1	—
STH	Rsrc1,@Rsrc2	*(sh *)Rsrc2 = Rsrc1	—
SUB	Rdest,Rsrc	Rdest = Rdest - Rsrc	—
SUBV	Rdest,Rsrc	Rdest = Rdest - Rsrc	変化
SUBX	Rdest,Rsrc	Rdest = Rdest - Rsrc - C	変化
TRAP	#n	PSW[BSM,BIE,BC] = PSW[SM,IE,C] PSW[SM,IE,C] = PSW[SM,0,0] Call trap-handler number-n	変化
UNLOCK	Rsrc1,@Rsrc2	if(LOCK) { *(s *)Rsrc2 = Rsrc1; } LOCK=0	—
XOR	Rdest,Rsrc	Rdest = Rdest ^ Rsrc	—
XOR3	Rdest,Rsrc,#imm16	Rdest = Rsrc ^ (uh)imm16	—

where:

```
typedef singed int      s;  /* 32 bit signed integer (word)*/
typedef unsigned int    u;  /* 32 bit unsigned integer (word)*/
typedef signed short    sh; /* 16 bit signed integer (halfword)*/
typedef unsigned short  uh; /* 16 bit unsigned integer (halfword)*/
typedef signed char     sb; /*  8 bit signed integer (byte)*/
typedef unsigned char   ub; /*  8 bit unsigned integer (byte)*/
```

付録3　パイプライン処理機構

付録3.1　パイプライン処理機構の概要

M32R CPUは、5段のパイプラインステージで構成されています。以下に各ステージの概要を示します。

(1) IFステージ(命令フェッチステージ)

メモリから命令をフェッチ(IF)します。M32R CPUは命令キューを備えており、D(デコード)ステージのデコード処理完了とは無関係に、命令キューがいっぱいになるまでフェッチを続けます。

あらかじめ命令キューに命令がある場合は、命令キューから読み出した命令を命令デコーダに渡します。

(2) Dステージ(デコードステージ)

Dステージ前半は、命令のデコード処理(DEC1)を行います。後半は命令デコード処理の続き(DEC2)と、レジスタのフェッチ(RF)を行います。

(3) Eステージ(実行ステージ)

演算やアドレス計算など(OP)を行います。

直前の命令の演算結果を必要とする処理の場合は、Eステージの前半でバイパス(BYP)を行います(バイパス処理機構については、付図3.5をご参照ください)。

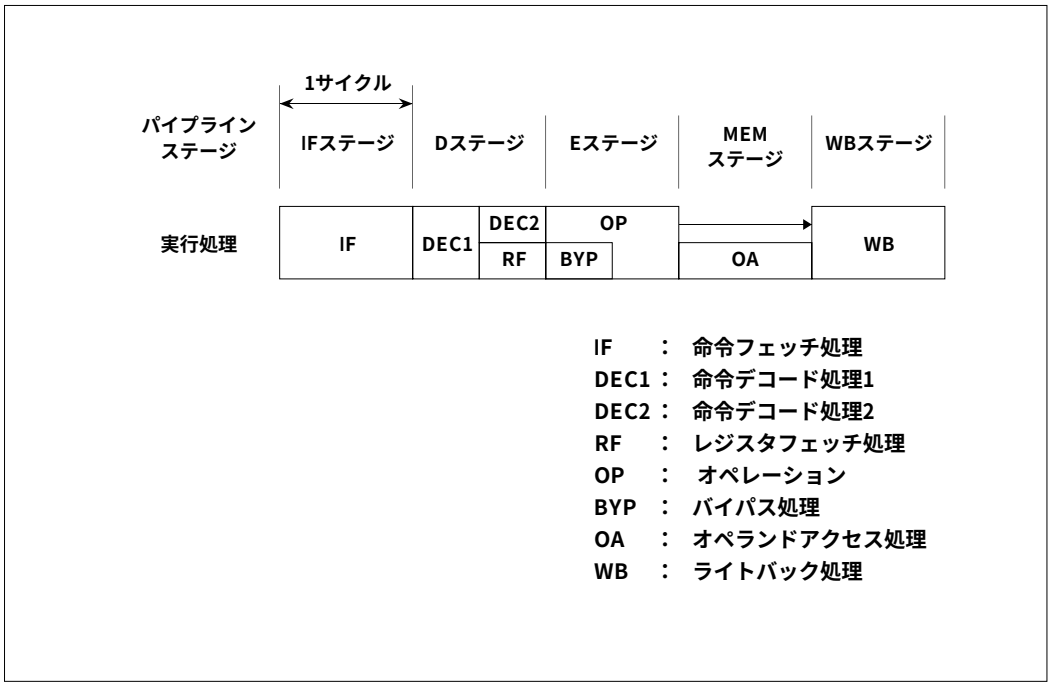
(4) MEMステージ(メモリアクセスステージ)

オペランドのアクセス(OA)を行います。ロード／ストア命令実行時のみこのステージが使われます。

(5) WBステージ(ライトバックステージ)

演算結果やフェッチしてきたデータをレジスタ、またはアキュムレータに書き込みます(WB)。

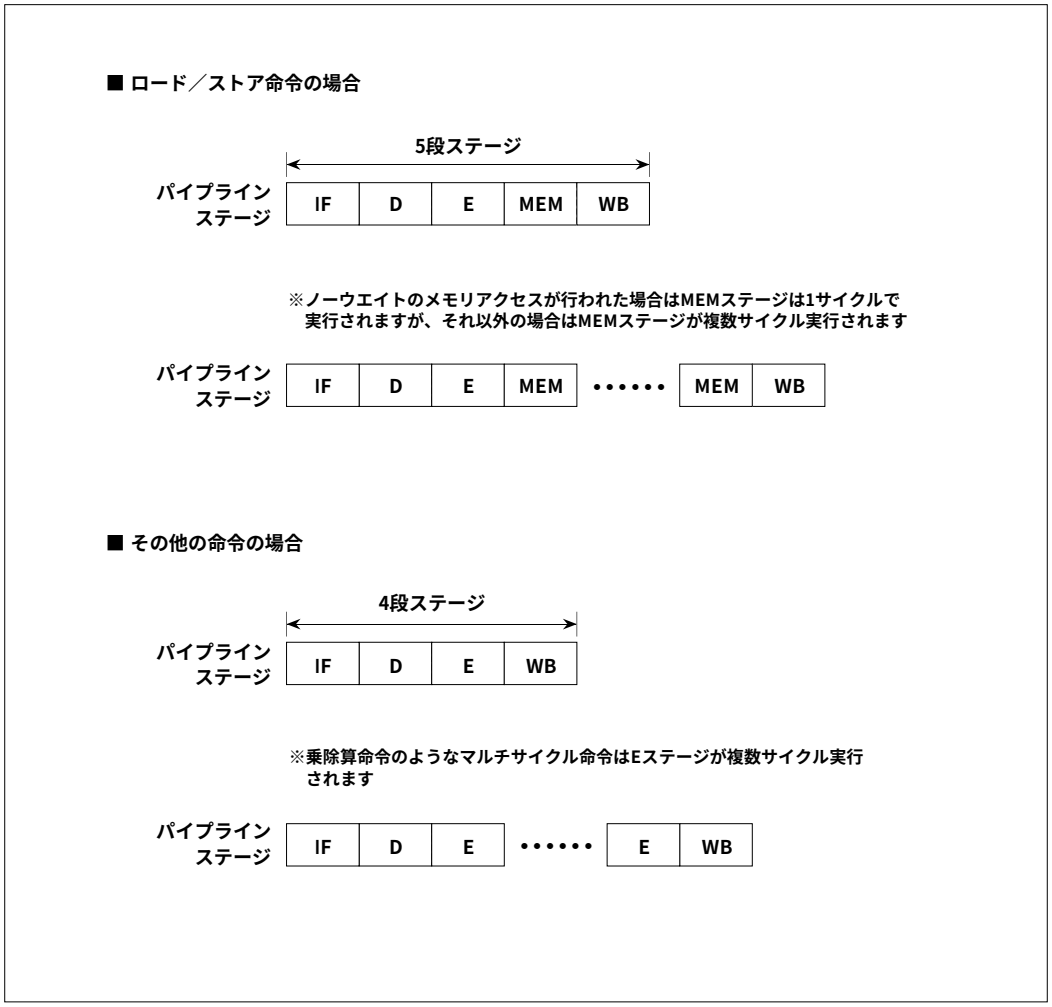
次ページにパイプラインの構成とその動作について示します。



付図3.1　パイプラインの構成とその動作

付録3.2　命令とパイプライン処理

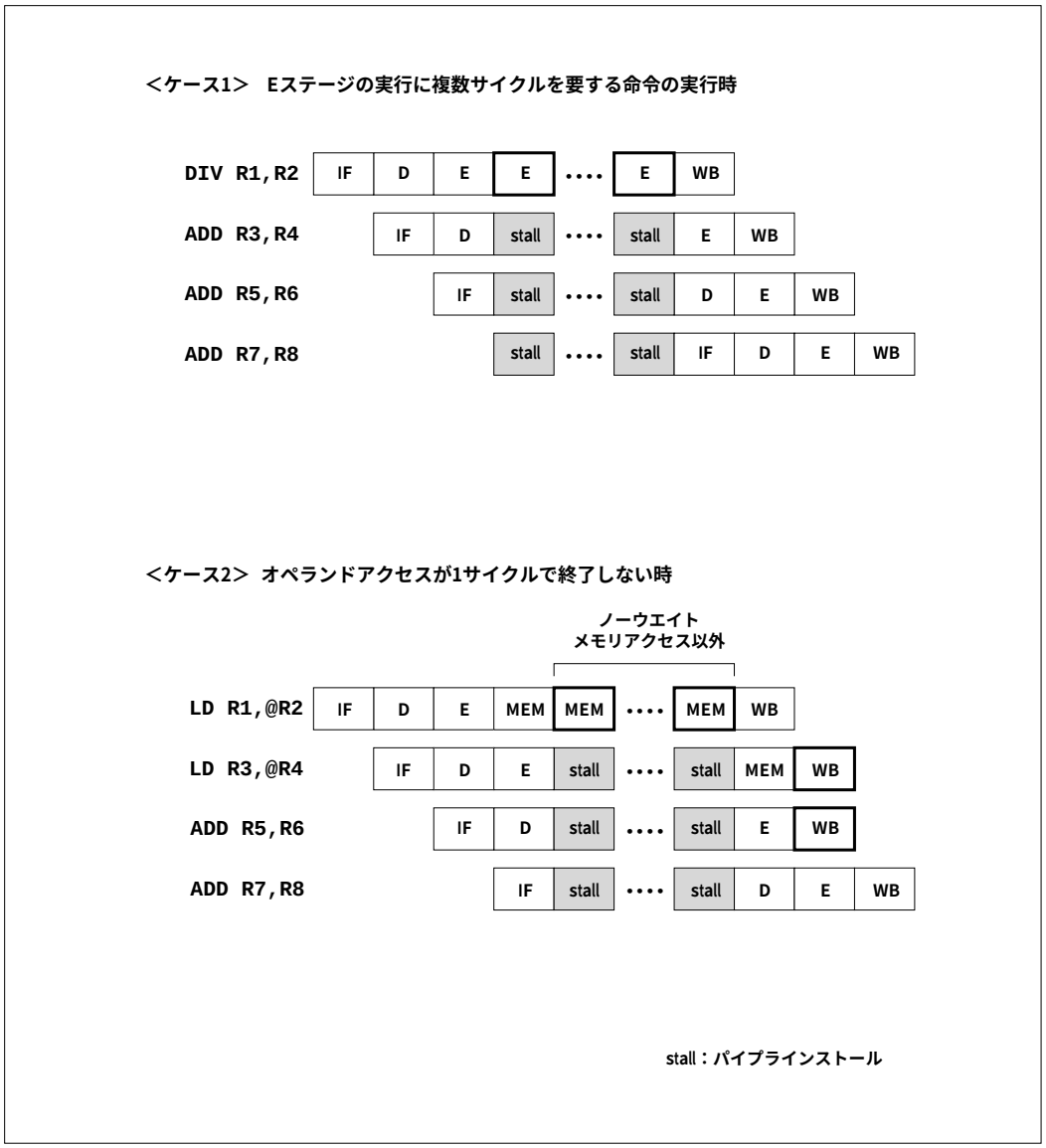
M32R CPUのパイプラインは5段のステージで構成されますが、実際にはMEMステージはロード／ストア命令でしか使用されないため、それ以外の命令では4段のパイプライン処理となります。



付図3.2　命令とパイプライン処理

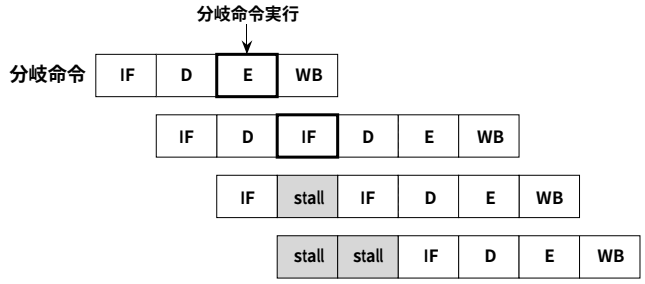
付録3.3　パイプラインの基本動作

理想的なパイプライン処理では、各ステージでの実行サイクル数は1ですが、各ステージでの処理や分岐命令の実行などによりパイプライン動作が乱れることがあります。
以下にそれらの基本的な動作をケース別に示します。

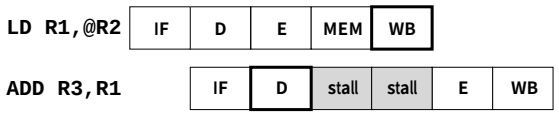


付図3.3　パイプライン動作が乱れるケース-1

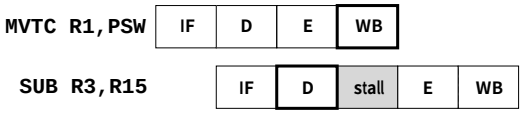
<ケース3> 分岐命令を実行した時（条件分岐命令で分岐しなかった場合を除く）



<ケース4> メモリからリードしたオペランドを後続命令が使用する時



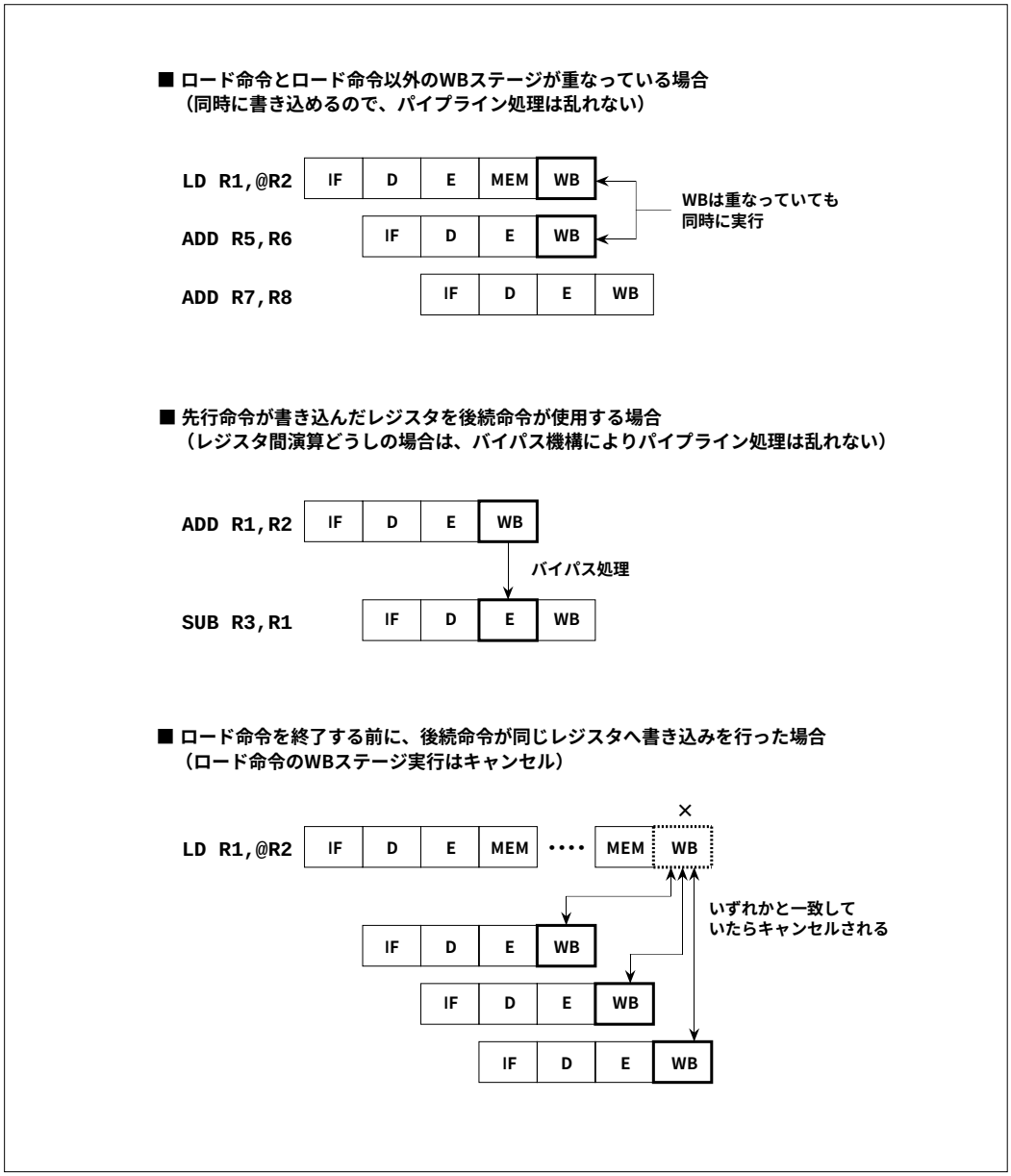
<ケース5> MVTC命令でPSWのSMビットに書き込み後、後続命令がR15を読み出す場合



stall：パイプラインストール

付図3.4 パイプライン動作が乱れるケース-2

下記の場合については、特殊なケースとして、パイプライン動作は乱れません。



付図3.5　パイプライン動作が乱れない特殊なケース

付録4　命令処理時間

M32R CPUは、通常Eステージにおける命令実行サイクル数を命令処理時間として代表しますが、パイプラインの動作によっては、それ以外のステージが処理時間に影響を与えることがあります。特に分岐命令を実行した場合の次命令においては、IF(命令フェッチ)、D(デコード)、E(実行)の各ステージの処理時間を考慮に入れることが必要です。

以下にM32R CPUの各パイプラインステージごとの命令処理時間を示します。なお、IF(命令フェッチステージ)およびMEM(メモリアクセスステージ)の処理時間は、M32Rファミリ各機種のインプリメンテーション(外部バスインタフェースの構成等)に依存するため、これらのステージの実行時間については、M32Rファミリ各機種のユーザーズマニュアルを参照ください。

付表4.1　各ステージにおける命令処理時間

	各ステージにおける実行サイクル数				
命令	IF	D	E	MEM	WB
ロード命令(LD, LDB, LDUB, LDH, LDUH, LOCK)	R(注)	1	1	R(注)	1
ストア命令(ST, STB, STH,UNLOCK)	R(注)	1	1	W(注)	–
乗算命令(MUL)	R(注)	1	3	–	1
除算／剰余命令(DIV, DIVU, REM, REMU)	R(注)	1	37	–	1
上記以外の命令(DSP機能用命令を含む)	R(注)	1	1	–	1

注. **R**, **W**で示される実行サイクル数は、M32Rファミリ各機種のユーザーズマニュアルを参照ください。

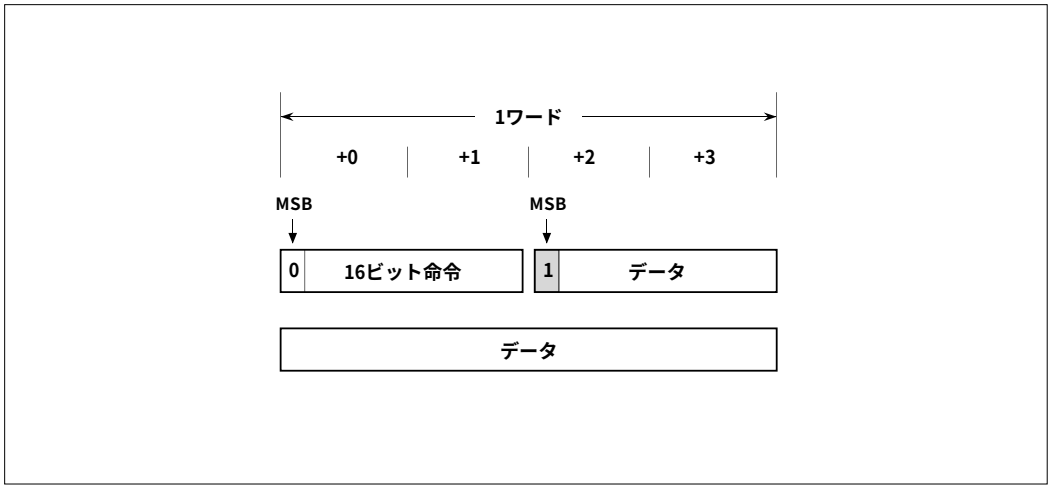
付録5　注意事項

■データ配置時の注意

プログラムのコード領域につづいてデータ領域の配置または確保をする場合は、ワードアライメント調整を行ったアドレスから配置してください。

ワードアライメント調整を行わずにデータ領域の配置または確保をした場合、1ワードの上位のハーフワードに16ビット命令が存在し、それにつづくハーフワードにMSBが"1"であるデータが配置されることがあります。このとき、M32Rファミリ上位互換のCPUでは、16ビット命令とデータを並列実行可能な命令のペアであると認識し並列実行処理を行います。

ソフトウエアの上位互換性を考慮して、上位のハーフワードに16ビット命令が存在する場合、それにつづくデータ領域はワードアライメント調整を行ったアドレスから配置または確保するように、プログラミング時にはご注意ください。



索引

索引

記号

A

B

C

D

E

J

L

M

N

O

P

R

S

T

索引

U

X

ア

イ

エ

オ

サ

シ

ス

セ

テ

ト

ハ

ヒ

フ

メ

ユ

レ

ロ

ワ

索引

＊空きページです＊

三菱シングルチップマイクロコンピュータ
ソフトウエアマニュアル
M32Rファミリ

2000年3月
発行所　三菱電機株式会社半導体営業統括部
〒100–8310　東京都千代田区丸ノ内2-2-3 / 三菱電機ビル
TEL　03-3218-9450

M32R ファミリ
ソフトウェアマニュアル

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