User's Manual

32

RH850G3KH

ユーザーズマニュアル　ソフトウェア編

ルネサスマイクロコントローラ

RH850 ファミリ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.20　2016.12

ご注意書き

1. 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・システムの設計において、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て、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。
2. 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。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。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。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   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、産業用ロボット等

   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、各種安全装置等

   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。 たとえ、意図し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損害が生じても、当社は一切その責任を負いません。 なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の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。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、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、社会的損害等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、お客様の機器・システムとしての出荷保証を行ってください。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、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の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。
9. 本資料に記載されている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・システムに使用することはできません。また、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に使用しないでください。当社製品または技術を輸出する場合は、「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。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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禁じます。

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。

CMOSデバイスの一般的注意事項

(1) 入力端子の印加波形：入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOSデバイスの入力がノイズなどに起因して，VIL（MAX.）からVIH（MIN.）までの領域にとどまるような場合は，誤動作を引き起こす恐れがあります。入力レベルが固定な場合はもちろん，VIL（MAX.）からVIH（MIN.）までの領域を通過する遷移期間中にチャタリングノイズ等が入らないようご使用ください。

(2) 未使用入力の処理：CMOSデバイスの未使用端子の入力レベルは固定してください。未使用端子入力については，CMOSデバイスの入力に何も接続しない状態で動作させるのではなく，プルアップかプルダウンによって入力レベルを固定してください。また，未使用の入出力端子が出力となる可能性（タイミングは規定しません）を考慮すると，個別に抵抗を介してVDDまたはGNDに接続することが有効です。資料中に「未使用端子の処理」について記載のある製品については，その内容を守ってください。

(3) 静電気対策：MOSデバイス取り扱いの際は静電気防止を心がけてください。MOSデバイスは強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には，当社が出荷梱包に使用している導電性のトレーやマガジン・ケース，または導電性の緩衝材，金属ケースなどを利用し，組み立て工程にはアースを施してください。プラスチック板上に放置したり，端子を触ったりしないでください。また，MOSデバイスを実装したボードについても同様の扱いをしてください。

(4) 初期化以前の状態　電源投入時，MOSデバイスの初期状態は不定です。電源投入時の端子の出力状態や入出力設定，レジスタ内容などは保証しておりません。ただし，リセット動作やモード設定で定義している項目については，これらの動作ののちに保証の対象となります。リセット機能を持つデバイスの電源投入後は，まずリセット動作を実行してください。

(5) 電源投入切断順序　内部動作および外部インタフェースで異なる電源を使用するデバイスの場合，原則として内部電源を投入した後に外部電源を投入してください。切断の際には，原則として外部電源を切断した後に内部電源を切断してください。逆の電源投入切断順により，内部素子に過電圧が印加され，誤動作を引き起こしたり，異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源投入切断シーケンス」についての記載のある製品については，その内容を守ってください。

(6) 電源OFF時における入力信号　当該デバイスの電源がOFF状態の時に，入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により，誤動作を引き起こしたり，異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源OFF時における入力信号」についての記載のある製品については，その内容を守ってください。

このマニュアルの使い方

目的と対象者　このマニュアルは、RH850G3KH ソフトウェア編を理解し、それを用いた応用システムを設計するユーザを対象とします。

凡 例　データ表記の重み：左が上位桁、右が下位桁

アクティブ・ロウの表記：$\overline{xxx}$（端子、信号名称に上線）

メモリ・マップのアドレス：上部－上位、下部－下位

注 ：本文中に付けた注の説明

注意：気を付けて読んでいただきたい内容

備考：本文の補足説明

数の表記：　2 進数 … xxxx または $xxxx_B$

10 進数 … xxxx

16 進数 … $xxxx_H$

目　次

第 1 章　概要

1.1　RH850G3KH の特長

RH850G3KH は、32 ビット RISC マイコン V850 シリーズに対し、命令セット・レベルでの上位互換性を特長とする CPU です。

RH850G3KH の特長を**表 1.1** に示します。

表 1.1　RH850G3KH の特長

項目	特長
CPU	• 組み込み制御用高性能 32 ビット・アーキテクチャ • 32 ビット内部データ・バス • 32 本の 32 ビット汎用レジスタ • RISC タイプ命令セット（V850、V850E1、V850E2 と上位互換性あり） ロング／ショート形式を持つロード／ストア命令 3 オペランド命令 C 言語に基づく命令セット • CPU 動作モード ユーザ・モード、スーパバイザ・モード • アドレス空間：データ／命令ともに 4G バイト･リニア
コプロセッサ	• 浮動小数点演算コプロセッサ（FPU）搭載 単精度（32 ビット）をサポート IEEE754 に準拠したデータ・タイプおよび例外をサポート 丸めモード：近傍、0 方向、＋∞方向、－∞方向 非正規化数の扱い：0 への切り捨て、または IEEE754 準拠のための例外通知
例外／割り込み	• スケーラブルな割り込みチャネル数 • チャネルごとに設定可能な 16 レベルの割り込み優先度 • 性能要求／メモリ消費量によって選択可能なベクタ選択方式 直接分岐方式の例外ベクタ（直接ベクタ方式） アドレス・テーブル参照型の間接分岐方式の例外ベクタ（テーブル参照方式） • 専用命令（PUSHSP, POPSP）による割り込み時のコンテキスト高速退避／復帰処理の支援
メモリ管理	• メモリ保護機能搭載（MPU）
キャッシュ	• キャッシュ非搭載

第 2 章　プロセッサ・モデル

本 CPU は、基本的な演算機能、レジスタ、例外管理機能などを備えたプロセッサ･モデルとなっています。

本章では、本 CPU のプロセッサ･モデルのうち、特徴的な機能を説明します。

2.1　CPU 動作モード

CPU の動作状態として、スーパバイザ・モード（SV）とユーザ・モード（UM）の 2 つの状態を有します。スーパバイザ・モード／ユーザ・モードは、PSW レジスタの UM ビットで示します。

- スーパバイザ・モード（PSW.UM = 0）　：すべてのハードウェア機能の管理・運用が可能なモード。
- ユーザ・モード（PSW.UM = 1）　：利用可能なハードウェア機能が制限されるモード。

2.1.1　CPU 動作モードの意味付け

(1) スーパバイザ・モード（SV）

すべてのハードウェア機能の管理・運用が可能なモードです。リセット解除後は、常にスーパバイザ・モードで起動します。

(2) ユーザ・モード（UM）

ユーザ・モードは、スーパバイザと対になる動作モードです。ユーザ・モードでは、スーパバイザによってアクセスが許可されたアドレス空間と、ユーザ資源として定義されたシステム・レジスタが利用できます。スーパバイザ実行権の命令は実行できず、例外となります。

ユーザ・モード（PSW.UM = 1）での制限

- SV 特権命令のオペレーティング制限による特権命令違反（→ PIE 例外）

特権命令のオペレーティング制限は、**「2.1.3　CPU 動作モードと特権」**を参照してください。

2.1.2　CPU 動作モードの遷移

CPU 動作モードは、3 つの事象にともない遷移します。

(a)　例外の受付による遷移

例外の受付によって、それぞれの例外に定められたモードに遷移します。

(b)　復帰命令による遷移

復帰命令の実行によって、EIPSW、FEPSW に退避されている対応ビットの値に応じて、PSW の値を復帰します。

(c)　システム・レジスタ操作命令による遷移

LDSR 命令によって、PSW の動作モード・ビットを直接書き換える／書き換えられることで遷移します。

注　意

1. **スーパバイザ・モードでは LDSR 命令を使って PSW.UM ビットを直接変更することができますが、システム・レジスタに関するハザードはハードウェア仕様で定義されます。本ビットの変更は PSW レジスタに関するハザードを回避するために復帰命令の使用を推奨します。**
2. **ユーザ・モードでは、PSW の上位 31 ～ 5 ビットに書き込み不可なため、遷移することができません。スーパバイザ・モードでは遷移することが可能ですが、システム・レジスタ・アクセスに関するハザードはハードウェア仕様で定義されます。本ビットの変更は PSW レジスタに関するハザードを回避するために復帰命令による遷移を推奨します。**

2.1.3　CPU 動作モードと特権

本 CPU では、CPU 動作モードと特定の資源に対する使用権の設定にしたがって利用可能な機能を制限することができます。特定の命令（特定のシステム・レジスタを更新する命令を含む）は、定義された動作モードでのみ実行可能です。このような特定の操作を実行するのに必要な権限を、以下では特権と呼びます。特権のない動作モードでは、命令の実行は行えず、例外が発生します。

本 CPU で定義される特権（使用権）は、次の 2 種類です。

- スーパバイザ（SV）特権：重要なシステム資源操作、致命的エラー処理、ユーザ・モードのプログラムの実行管理に必要な特権
- コプロセッサ使用権　　：コプロセッサの使用に必要な権限

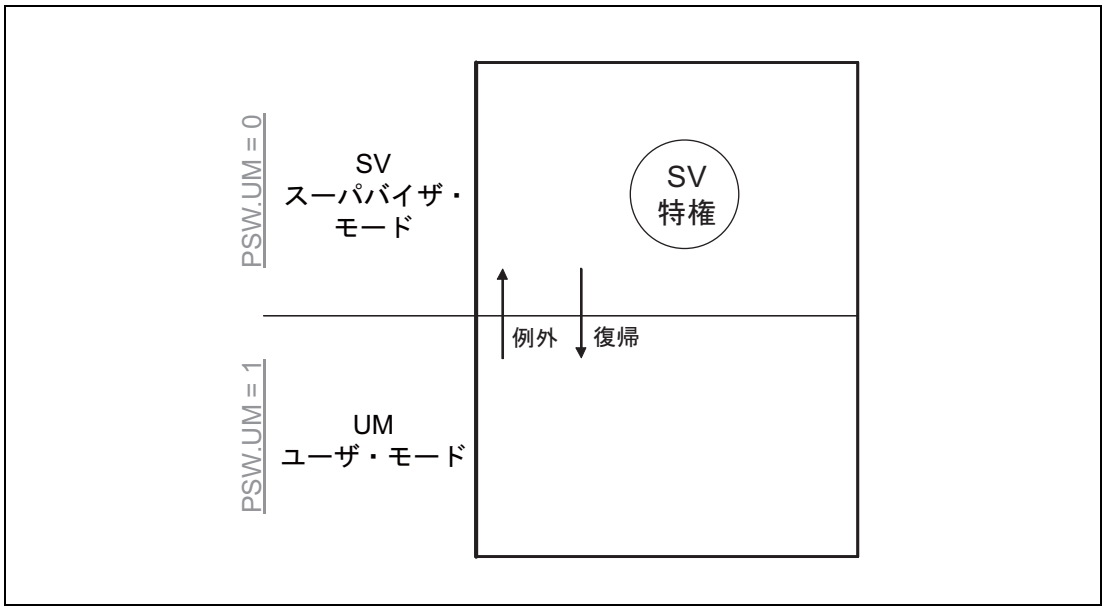


図 2.1　CPU 動作モードと特権

(1) スーパバイザ特権（SV 特権）

重要なシステム資源に対する操作、致命的エラー処理、ユーザ・モードのプログラムの実行管理に必要な特権を、スーパバイザ特権（SV 特権）と呼びます。スーパバイザ・モードは SV 特権を保持します。一般に、重要なシステム資源に対する操作を行うための命令の実行は SV 特権が必要で、これらの命令を SV 特権命令と呼ぶことがあります。

(2) コプロセッサ使用権

コプロセッサは、CPU 動作モードによらず、独立して利用可否を設定することができます。

PSW レジスタ上の CU2-0 ビットによって、各プログラムにコプロセッサの使用を許可／禁止をスーパバイザが決定します。CU ビットがセット（1）されていない場合、対応するコプロセッサ命令の実行、またはシステム・レジスタをアクセスした場合は、コプロセッサ使用不可例外が発生します。

コプロセッサ非搭載の場合には、該当する CU ビットをセット（1）することはできません。CU2-0 ビットによる設定は、CPU 動作モードに関わりなく有効であり、スーパバイザがコプロセッサのシステム・レジスタをアクセスする場合、自身で CU2-0 ビットを設定して使用を許可する必要があります。

(3) 特権違反時の動作

特権を所持していない場合に特権命令を実行した場合、PIE 例外または UCPOP 例外が発生します。

動作モードと使用権の状態によって、実行可能な命令の関係を**表 2.1** に示します。

表 2.1　特権違反時の動作

	PSW				実行の可否
	UM	CU2	CU1	CU0	
SV 特権命令	0	—	—	—	実行可
	1	—	—	—	実行不可／ PIE 例外
コプロセッサ命令 1[注 1]（PSW.CU0 ビット）	—	—	—	1	実行可
	—	—	—	0	実行不可／ UCPOP 例外
コプロセッサ命令 2[注 1]（PSW.CU1 ビット）	—	—	1	—	実行可
	—	—	0	—	実行不可／ UCPOP 例外
コプロセッサ命令 3[注 1]（PSW.CU2 ビット）	—	1	—	—	実行可
	—	0	—	—	実行不可／ UCPOP 例外
上記以外の命令（ユーザ命令）	—	—	—	—	実行可

注 1.　コプロセッサのシステム・レジスタに対する LDSR ／ STSR 命令を含みます。

備考　— : 0 または 1

注　意

アクセス権限が CUn かつ SV と定義されたレジスタに、CUn = 0 かつ UM = 0 の状態でアクセスした場合、UCPOP 例外が発生します。

2.2　命令の実行

本 CPU における命令の実行フローを次に示します。

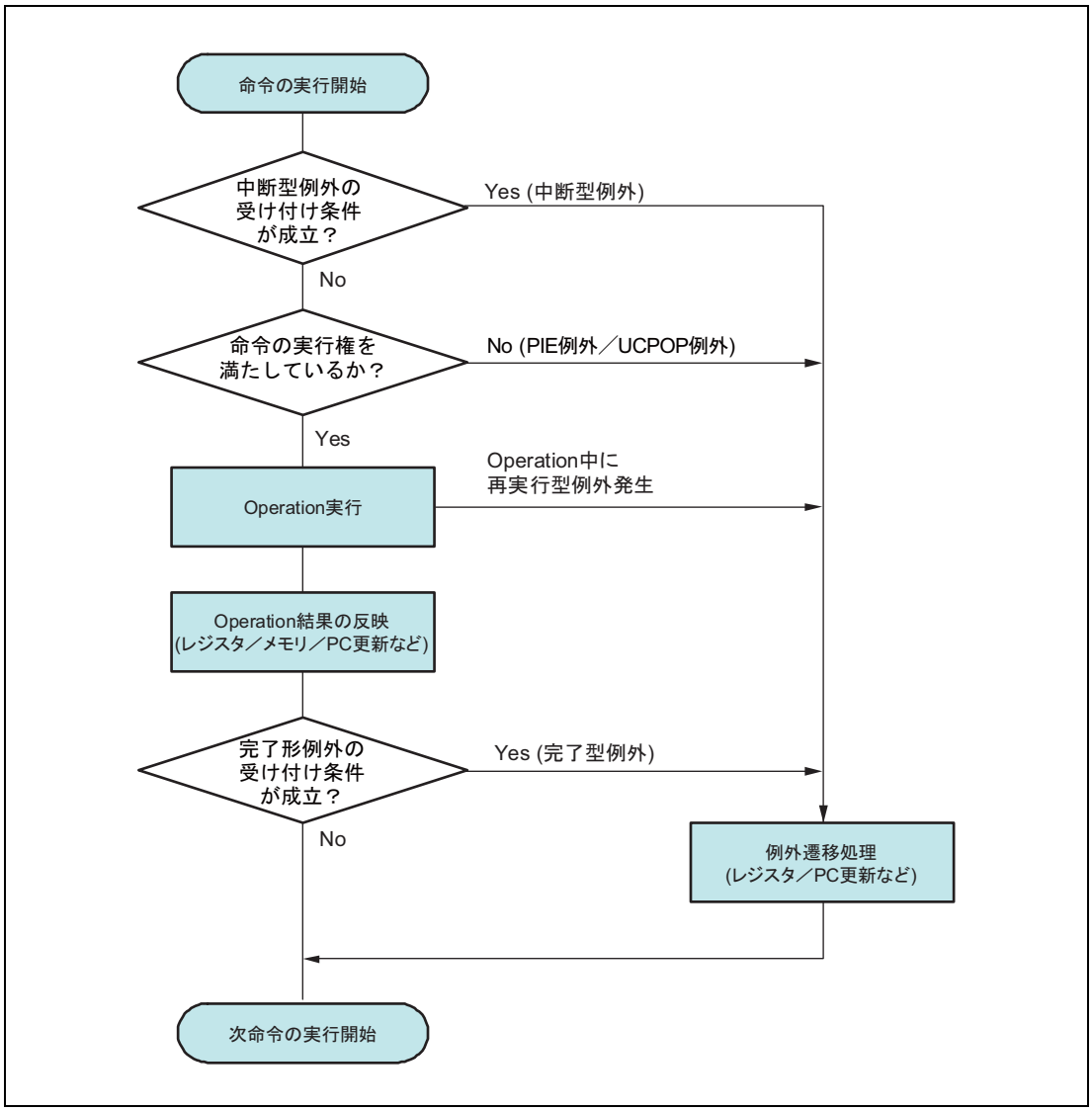


図 2.2　命令の実行フロー

命令の実行にあたっては、中断型例外が受け付け可能である場合、または命令の実行権を満たしていない場合に、命令のオペレーションを実行する前に、例外を受け付けます。また、オペレーション実行中に再実行型の例外が起きた場合は、オペレーションの実行中に例外を受け付けます。これらの場合には、原則的に命令オペレーションの結果は、レジスタやメモリ等に反映せず、CPU の状態は命令実行前の状態を保ちます。[注 1]

また、ソフトウェア例外などの完了型例外においては、命令オペレーションの結果を反映後、例外を受け付けます。

注 1. 次の命令は、中間結果をメモリに反映させる場合があります。
・PREPARE, DISPOSE, PUSHSP, POPSP

2.3　例外／割り込み

例外／割り込みとは、特定の要因によって実行中のプログラムから別のプログラムへの強制的な分岐動作を発生する例外的事象です。周辺装置などからの割り込みや、プログラムの異常動作など様々な要因によって、例外／割り込みが発生します。

詳細は、**「第 4 章　例外／割り込み」**を参照してください。

2.3.1　例外の実行形態

本 CPU の例外は、例外を通知する目的に合わせて、次の 3 つの実行形態を取ります。

- 中断型例外
- 再実行型例外
- 完了型例外

(1)　中断型例外

命令のオペレーションを実行する前に、その命令を中断して受け付けられる例外です。割り込みや、インプレサイス（不正確な）例外などが中断型例外にあたります。

この形態の例外は、割り込みや、ハードウェア・エラーなどの現在処理中のプログラムとは無関係の異なるプログラムを起動する場合と、命令動作に起因して発生しますが命令実行の過程では例外を起こせず、やむをえず遅延し、後続の命令の実行中に例外が発生するインプレサイス例外があります。

(2)　再実行型例外

命令のオペレーションの実行中に発生し、その命令の実行を完了せずに受け付ける例外で、命令実行時に命令を完了せず、また後続の命令を実行することもなく、正確に例外を受け付けるため、プレサイス（正確な）例外とも呼ばれます。

中断型例外であるインプレサイス例外とは異なり、命令実行の過程で例外が生じ、命令実行を取りやめるため、例外処理後に再び、例外を引き起こした命令を実行可能です。このため、メモリ管理機能などでは、例外処理によって適切な設定を施したあと、再度同じ命令を再実行することで、プログラムの論理的な動作の一貫性を保ったまま、複雑なメモリ管理を実現することができます。

(3)　完了型例外

命令のオペレーションの実行の結果として発生し、その命令の実行を完了後に受け付ける例外です。ソフトウェア例外などが完了型例外にあたります。

命令の正常なオペレーションとして例外の発生が定義されているため、再実行型とは異なり、例外を引き起こした命令自身は正常に完了し、再実行することはありません。主に、管理プログラムの呼び出しなどのコール・ゲート目的での利用を想定しています。

2.3.2　例外レベル

本 CPU では、例外処理中により緊急度の高い例外が発生した場合に、例外処理中でも例外が発生します。中断される例外処理がまだメモリへのコンテキスト退避を行っていない場合でも、例外を受け付け、再び中断された例外処理に復帰できるように、例外要因を次の 2 つの例外レベルに階層化して管理しています。

- EI レベル例外
- FE レベル例外

通常のユーザ処理、割り込み処理、OS 処理などは基本的に EI レベル例外を利用して構成します。システム上緊急度の高い割り込み処理や、OS 処理中などにも発生しうるメモリ管理機能の例外処理などは EI レベル例外処理中のいかなる場合にでも受け付けの行える FE レベル例外を利用します。

2.4　コプロセッサ

本 CPU は、単精度 FPU 拡張機能を搭載しています。

2.4.1　コプロセッサ使用権

コプロセッサ命令と定義されたオペコードの実行には対応するコプロセッサ命令ごとの使用許可が必要です。コプロセッサ使用権は PSW.CU2-0 ビットで表現され、対応するコプロセッサ使用権がクリア（0）されている状態で実行しようとすると、コプロセッサ使用不可例外（UCPOP）が発生します。

2.4.2　コプロセッサ使用権とコプロセッサの対応

本 CPU では、CPU 動作中にプログラムごとにコプロセッサの利用可否を制御するコプロセッサ使用権を定義しています。コプロセッサ使用権は CU0-CU2 までの 3 つあり、コプロセッサとの対応関係は次の表で示されます。

表 2.2　コプロセッサ使用権とコプロセッサの対応

コプロセッサ使用権	コプロセッサ機能	例外要因コード
CU0	単精度 FPU 拡張機能	80_H
CU1	予約	81_H
CU2	予約	82_H

2.4.3　コプロセッサ使用不可例外

コプロセッサ使用不可例外は、コプロセッサ使用権が与えられていない状態（PSW.CUn = 0）で、対応するコプロセッサ命令を実行しようとした場合、または対応するコプロセッサのシステム・レジスタにアクセスしようとした場合に発生します。

2.4.4　システム・レジスタ

コプロセッサ機能によっては、その機能の一部としてシステム・レジスタが定義されるものがあります。コプロセッサ機能のシステム・レジスタへのアクセスにはコプロセッサ使用権が必要です。一部のシステム・レジスタはコプロセッサ使用権に加えてスーパバイザ特権（SV 権限）が必要な場合があります。

システム・レジスタのアクセスに必要な権限の詳細は**「2.5　レジスタ」**を参照してください。

2.5　レジスタ

本 CPU では、プログラム・レジスタ（汎用レジスタやプログラム･カウンタ PC）と、状態制御や、例外情報保持のためのシステム・レジスタが定義されます。

2.5.1　プログラム・レジスタ

プログラム・レジスタには、汎用レジスタ（r0-r31）とプログラム・カウンタ（PC）があります。

表 2.3　プログラム・レジスター覧

分類	特権	名称
プログラム・カウンタ	UM	PC
汎用レジスタ	UM	r0-r31

備考　UM：ユーザ・レジスタ。特権はないので、常にアクセス可能。

2.5.2　システム・レジスタ

プログラム･レジスタについては、**「3.1　プログラム・レジスタ」**を参照してください。

グループ番号 0-3　：基本機能にかかわるレジスタ

グループ番号 4-7　：メモリ管理機能にかかわるレジスタ

グループ番号 12-15 ：CPU ハードウェア仕様で定義されるレジスタ

グループ番号 16-　：将来の拡張のため予約

システム・レジスタの詳細は、**「第 3 章　レジスタ・セット」**の各節を参照してください。

2.5.3　レジスタの更新

レジスタの更新方法は、いくつかの方法が存在します。通常、命令オペレーションによって更新する際に特別な制限はありませんが、以下の命令による更新の場合、動作モードによる制限が課せられます。

- LDSR
- STSR

(1) LDSR、STSR

LDSR、STSR 命令は、システム・レジスタすべてにアクセスが可能です。ただし、適切な権限を所持していない状態でアクセスを行った場合、PIE 例外、あるいは UCPOP 例外が発生する場合があります。レジスタごとのアクセス権限は、**「第 3 章　レジスタ・セット」**の各システム・レジスター覧を参照してください。また、特権違反時の挙動については、**「2.1.3　CPU 動作モードと特権」**を参照してください。

LDSR/STSR 命令の実行フローを**図 2.3** に示します。

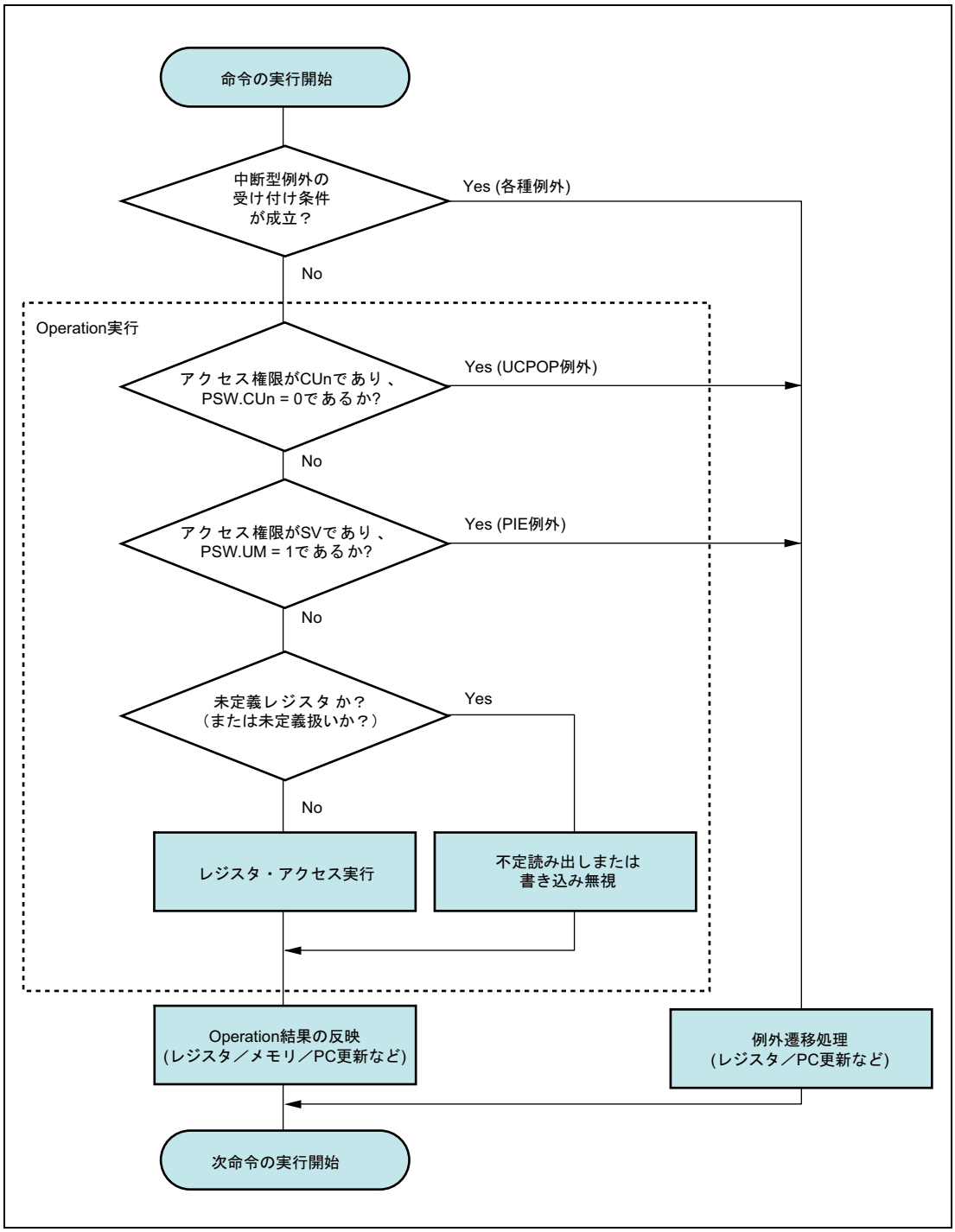


図 2.3　　LDSR/STSR 命令の実行フロー

2.5.4　未定義レジスタへの操作

レジスタの割り当てられていないシステム・レジスタ番号へのアクセスや、アクセス不可能なレジスタへのアクセスを行った場合、次のような結果が生じます。

- 未定義レジスタは、すべて SV 権限として扱われます。ユーザ・モード（PSW.UM = 1）で LDSR/STSR 命令を使用してアクセスした場合、PIE 例外が発生します。
- 読み出し操作の場合、読み出し結果は不定です。読み出し値をプログラムで利用した場合、意図しない動作となる場合があります。
- 書き込み操作の場合、書き込み操作は無視されます。

2.6　データ・タイプ

2.6.1　データ形式

本 CPU では、データをリトル・エンディアン形式で取り扱います。つまり、ハーフワード、ワードではバイト 0 が常に最下位（最右端）バイトとなります。

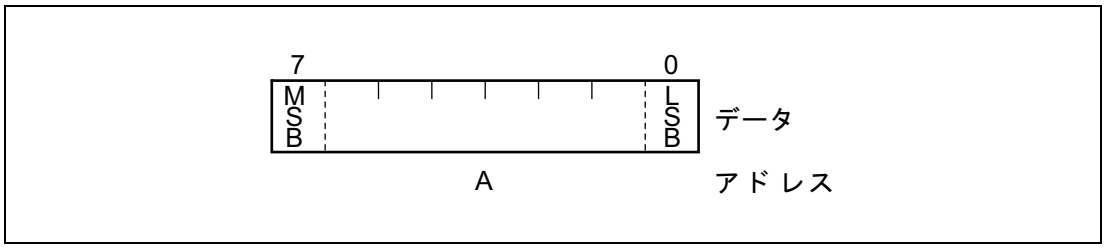
また、サポートしているデータ形式は次のとおりです。

- バイト（8 ビット長データ）
- ハーフワード（16 ビット長データ）
- ワード（32 ビット長データ）
- ダブルワード（64 ビット長データ）
- ビット（1 ビット長データ）

各データ形式のメモリ上の配置は次のようになります。

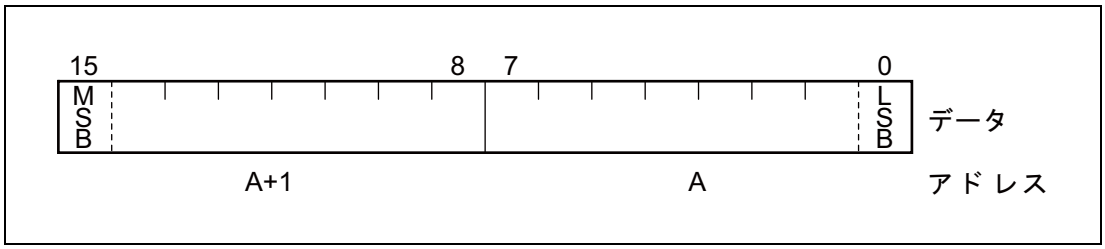
(1) バイト

バイトは、任意のバイト境界から始まる連続した 8 ビットに配置されます。各ビットには 0 から 7 までの番号が付けられており、LSB（Least significant bit）はビット 0、MSB（Most significant bit）はビット 7 に対応します。バイトは、そのアドレス「A」で指定されます。



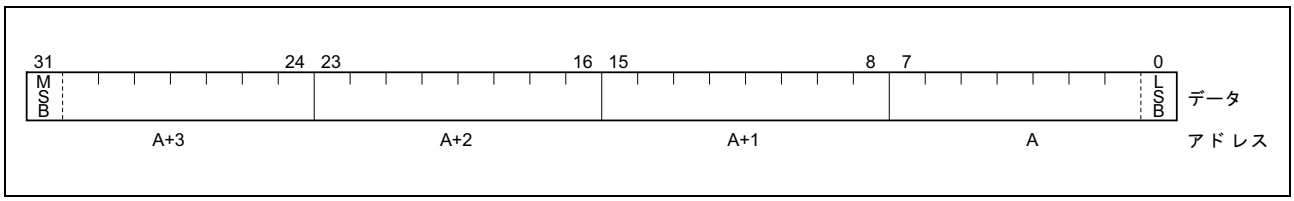
(2) ハーフワード

ハーフワードは任意のバイト境界から始まる連続した 2 バイト（16 ビット）に配置されます。各ビットには、0 から 15 までの番号が付けられており、LSB はビット 0、MSB はビット 15 に対応します。ハーフワードはそのアドレス「A」で指定され、2 つのアドレス「A」、「A ＋ 1」のバイト・データを占めます。



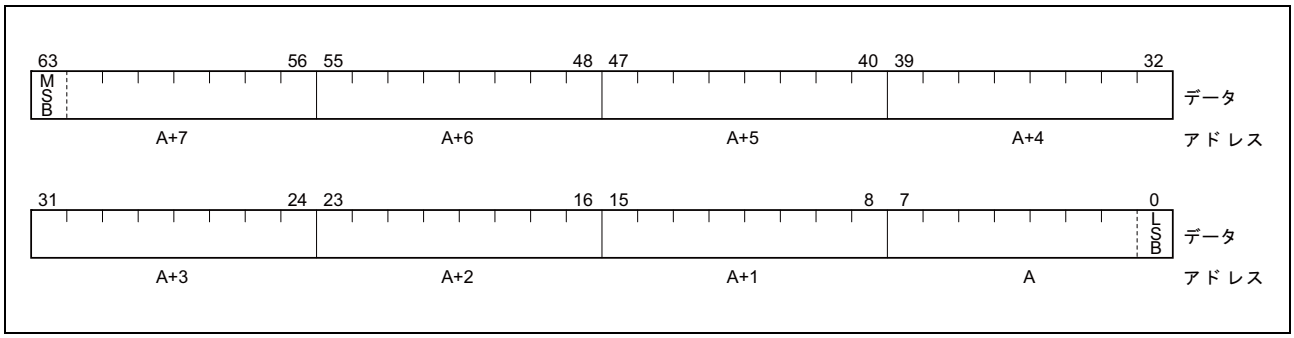
(3) ワード

ワードは任意のバイト境界から始まる連続した 4 バイト（32 ビット）に配置されます。各ビットには 0 から 31 までの番号が付けられており、LSB はビット 0、MSB はビット 31 に対応します。ワードはそのアドレス「A」で指定され、4 つのアドレス「A」、「A ＋ 1」、…「A ＋ 3」のバイト・データを占めます。



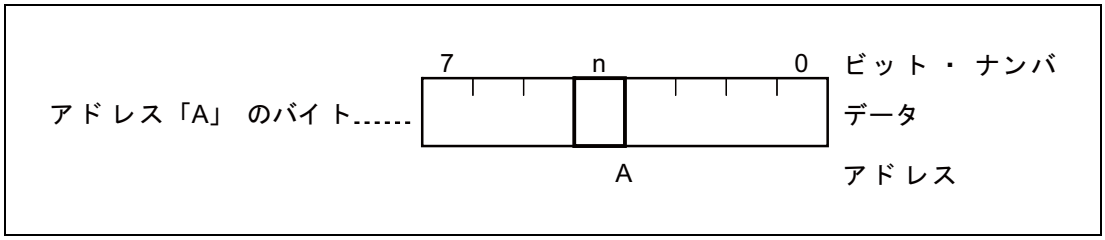
(4) ダブルワード

ダブルワードは任意のバイト境界から始まる連続した 8 バイト（64 ビット）に配置されます。各ビットには 0 から 63 までの番号が付けられており、LSB はビット 0、MSB はビット 63 に対応します。ダブルワードはそのアドレス「A」で指定され、8 つのアドレス「A」、「A ＋ 1」、…「A ＋ 7」のバイト・データを占めます。



(5) ビット

ビットは、任意のバイト境界から始まる 8 ビット・データの n ビット目の 1 ビットに配置されます。ビットはそのバイトのアドレス「A」と、ビット・ナンバ「n」で指定されます（n = 0 ～ 7）。



2.6.2　データ表現

(1) 整数

整数は2の補数による2進数で表現し、64ビット、32ビット、16ビット、8ビットの3通りの長さを持っています。整数の位取りは、その長さにかかわらずビット0を最下位ビットとし、ビット番号が増えるにしたがって位取りを高くします。2の補数表現であるため、最上位ビットを符号ビットとして使用します。

各データ長の整数の範囲は次のとおりです。

- ダブルワード（64ビット）：–9223372036854775808 ～ +9223372036854775807
- ワード（32ビット）　　：–2147483648 ～ +2147483647
- ハーフワード（16ビット）：–32768 ～ +32767
- バイト（8ビット）　　：–128 ～ +127

(2) 符号なし整数

「整数」が、正負両方の値を取るデータであるのに対して、「符号なし整数」は、負でない整数を意味します。整数と同様に、符号なし整数も2進数で表現し、64ビット、32ビット、16ビット、8ビットの4通りの長さを持っています。符号なし整数の位取りは、整数と同様に、その長さにかかわらずビット0を最下位ビットとし、ビット番号が増えるにしたがって位取りを高くします。ただし符号ビットは存在しません。

各データ長の符号なし整数の範囲は次のとおりです。

- ダブルワード（64ビット）：0 ～ 18446744073709551615
- ワード（32ビット）　　：0 ～ 4294967295
- ハーフワード（16ビット）：0 ～ 65535
- バイト（8ビット）　　：0 ～ 255

(3) ビット

ビット・データとして、クリア（0）またはセット（1）の2つの値をとる1ビットのデータを扱うことができます。ビットに関する操作は、メモリ空間の1バイト・データだけを対象とし、次の4種類の操作ができます。

- セット
- クリア
- 反転
- テスト

2.6.3　データ・アライメント

本 CPU は、アドレス計算結果に対して、同時に 2 種類のデータ・アライメントのチェックを行います。

タイプ 1 : 32 ビットアライメントまでのアライメント・チェック

処理対象のデータがハーフワード形式の場合は、16 ビットアライメント（アドレスの最下位ビットが 0）以外のアドレスへのアクセスを、処理対象のデータがワード形式またはダブルワード形式の場合は、32 ビットアライメント（アドレスの下位 2 ビットが 0）以外のアドレスへのアクセスを、アライメントがとれていない（アライメント違反）と判定します。

アライメント違反時は、ミスアライン例外（MAE）を発生させることができます。[注 1]
なお、PREPARE 命令、DISPOSE 命令、PUSHSP 命令、POPSP 命令は、アドレス下位 2 ビットを “00” にマスクするため、常にアライメントがとれたアクセスを行います。

タイプ 2 : 64 ビットアライメントまでのアライメント・チェック

処理対象のデータがハーフワード形式および、ワード形式の場合はタイプ 1 と同じ判定をします。処理対象のデータがダブルワード形式の場合は、64 ビットアライメント（アドレスの下位 3 ビットが 0）以外のアドレスへのアクセスを、アライメントがとれていない（アライメント違反）と判定します。

メモリ保護違反を引き起こした命令がアライメント違反のアクセス （ミスアライン・アクセス）を行う場合は、FEIC.MS ビット[注 2] がセット (1) されます。
なお、PREPARE 命令、DISPOSE 命令、PUSHSP 命令、POPSP 命令は、アドレス下位 2 ビットを “00” にマスクするため、常にアライメントがとれたアクセスを行います。

注 1. MCTL.MA ビットの値によります。

注 2. FEIC.MS ビットの詳細は、**「表 5.1　メモリ保護違反時の例外要因コード」**を参照して下さい。

タイプ 1 のアライメント・チェックとタイプ 2 のアライメント・チェックにより、アライメント違反と判定される命令とアドレスの組み合わせと、アライメント違反時の動作を、**表 2.4** と **表 2.5** に示します。

表 2.4　アライメント違反条件とアライメント違反時の動作（MCTL.MA = 0）

データ形式	命令	アドレスの下位 3 ビット（上段：アクセスの可否、下段：アライメント・チェック後の動作）							
		000	001	010	011	100	101	110	111
ハーフワード（16 ビット）	LD.H, LD.HU, SLD.H, SLD.HU, SST.H, ST.H	アクセス可	アクセス不可	アクセス可	アクセス不可	アクセス可	アクセス不可	アクセス可	アクセス不可
		—	MAE 発生 FEIC.MS = 1 注 1	—	MAE 発生 FEIC.MS = 1 注 1	—	MAE 発生 FEIC.MS = 1 注 1	—	MAE 発生 FEIC.MS = 1 注 1
ワード（32 ビット）	LD.W, SLD.W, SST.W, ST.W LDL.W, STC.W, CAXI	アクセス可	アクセス不可	アクセス不可	アクセス不可	アクセス可	アクセス不可	アクセス不可	アクセス不可
		—	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	—	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1
ダブルワード（64 ビット）	LD.DW, ST.DW	アクセス可	アクセス不可	アクセス不可	アクセス不可	アクセス可	アクセス不可	アクセス不可	アクセス不可
		—	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1

注 1.　メモリ保護違反の場合のみ。
FEIC.MS ビットの詳細は、**「表 5.1　メモリ保護違反時の例外要因コード」**を参照してください。

表 2.5　アライメント違反条件とアライメント違反時の動作（MCTL.MA = 1）

データ形式	命令	アドレスの下位 3 ビット（上段：アクセスの可否、下段：アライメント・チェック後の動作）							
		000	001	010	011	100	101	110	111
ハーフワード（16 ビット）	LD.H, LD.HU, SLD.H, SLD.HU, SST.H, ST.H	アクセス可	アクセス可	アクセス可	アクセス可	アクセス可	アクセス可	アクセス可	アクセス可
		—	FEIC.MS = 1 注 1	—	FEIC.MS = 1 注 1	—	FEIC.MS = 1 注 1	—	FEIC.MS = 1 注 1
ワード（32 ビット）	LD.W, SLD.W, SST.W, ST.W	アクセス可	アクセス可	アクセス可	アクセス可	アクセス可	アクセス可	アクセス可	アクセス可
		—	FEIC.MS = 1 注 1	FEIC.MS = 1 注 1	FEIC.MS = 1 注 1	—	FEIC.MS = 1 注 1	FEIC.MS = 1 注 1	FEIC.MS = 1 注 1
	LDL.W, STC.W, CAXI	アクセス可	アクセス不可	アクセス不可	アクセス不可	アクセス可	アクセス不可	アクセス不可	アクセス不可
		—	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	—	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1
ダブルワード（64 ビット）	LD.DW, ST.DW	アクセス可	アクセス不可	アクセス不可	アクセス不可	アクセス可	アクセス不可	アクセス不可	アクセス不可
		—	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1	MAE 発生 FEIC.MS = 1 注 1

注 1.　メモリ保護違反の場合のみ。
FEIC.MS ビットの詳細は、**「表 5.1　メモリ保護違反時の例外要因コード」**を参照してください。

2.7　アドレス空間

本 CPU は、4 G バイトのリニアなアドレス空間をサポートしています。このアドレス空間にはメモリと I/O の両方がマッピング可能です（メモリ・マップド I/O 方式）。CPU からメモリ、I/O に対して 32 ビットのアドレスが出力され、アドレス番地は最大「$2^{32}-1$」となります。

各アドレスに配置されるバイト・データは、ビット 0 を LSB、ビット 7 を MSB と定義されています。また、複数バイト構成のデータでは特に注意しないかぎり、下位側アドレスのバイト・データが LSB、上位側アドレスのバイト・データが MSB を持つように定義されています（リトル・エンディアン形式）。

このマニュアルでは、複数バイト構成のデータを表現する場合、次のように右側を下位側アドレス、左側を上位側アドレスとして表現します。

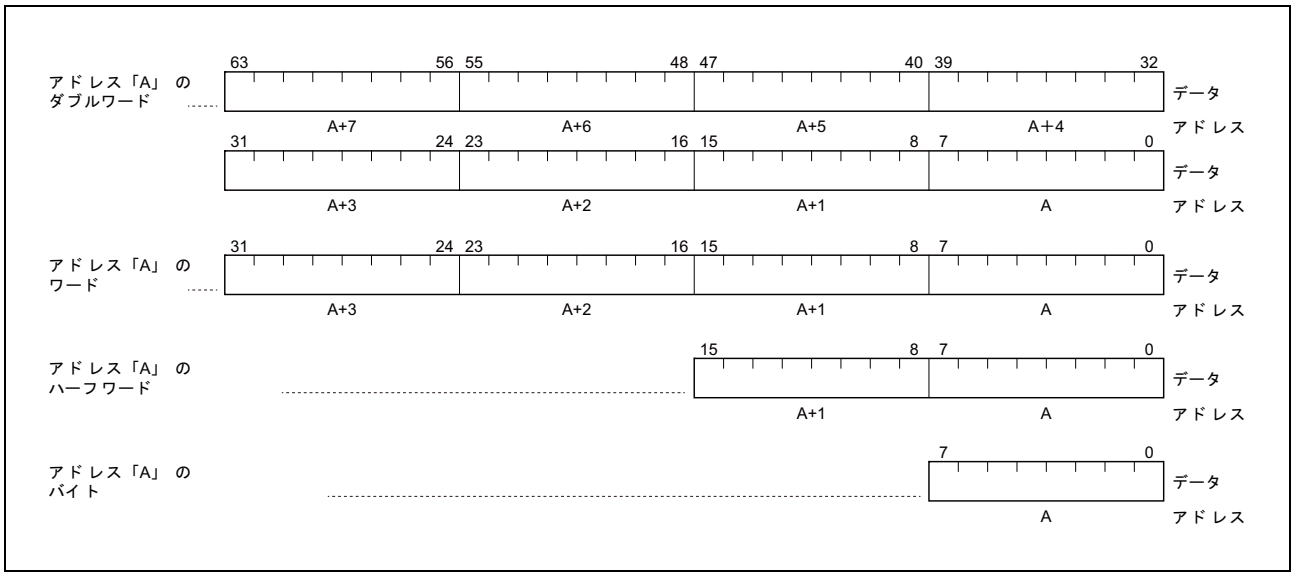


図 2.4　アドレス空間のバイト構成

2.7.1　メモリ・マップ

本 CPU は、32 ビット・アーキテクチャであり、最大 4 G バイトのリニア・アドレス空間をサポートします。命令アドレッシング（命令アクセス）、およびオペランド・アドレッシング（データ・アクセス）において、この 4 G バイトのアドレス空間内の全範囲を指定可能です。

メモリ・マップを**図 2.5** に示します。

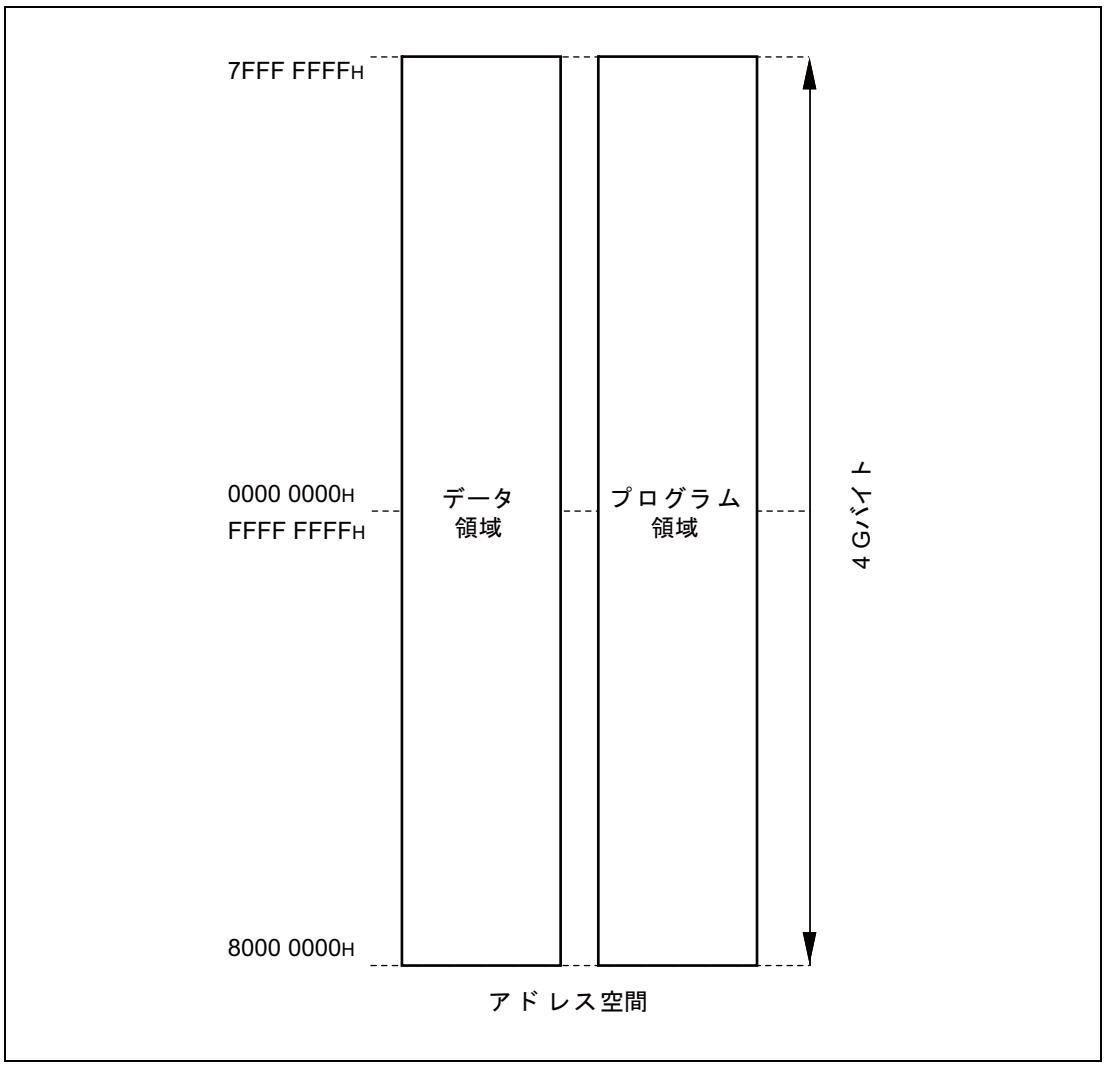


図 2.5　メモリ・マップ（アドレス空間）

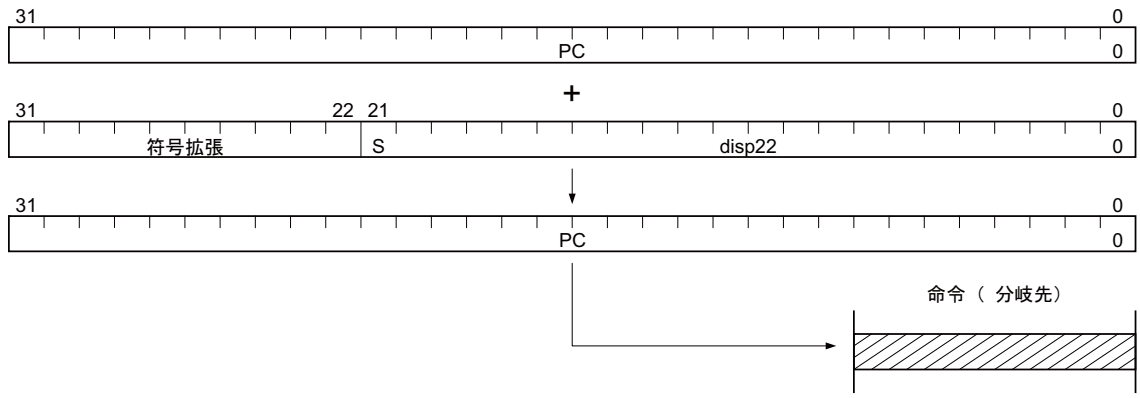
2.7.2　命令アドレッシング

命令アドレスは、プログラム・カウンタ（PC）の内容によって決定され、実行した命令のバイト数に応じて自動的にインクリメントされます。また、分岐命令を実行する際には、次に示すアドレッシングにより、分岐先アドレスを PC にセットします。

(1) レラティブ・アドレッシング（PC 相対）

プログラム・カウンタ（PC）に、命令コードの符号付き N ビット・データ（ディスプレースメント：disp N）を加算します。このとき、ディスプレースメントは、2 の補数データとして扱われ、それぞれ最上位ビットが符号ビット（S）となります。ディスプレースメントが 32 ビット未満の場合、上位ビットを符号拡張します（N は命令ごとに異なります）。

JARL 命令、JR 命令、Bcond 命令が、このアドレッシングの対象となります。



備考　22 ビット・ディスプレースメントの場合です。

図 2.6　　レラティブ・アドレッシング

(2) レジスタ・アドレッシング（レジスタ間接）

命令によって指定される汎用レジスタ（reg1）またはシステム・レジスタ（regID）の内容をプログラム・カウンタ（PC）に転送します。

JMP 命令、CTRET 命令、EIRET 命令、FERET 命令、DISPOSE 命令が、このアドレッシングの対象となります。

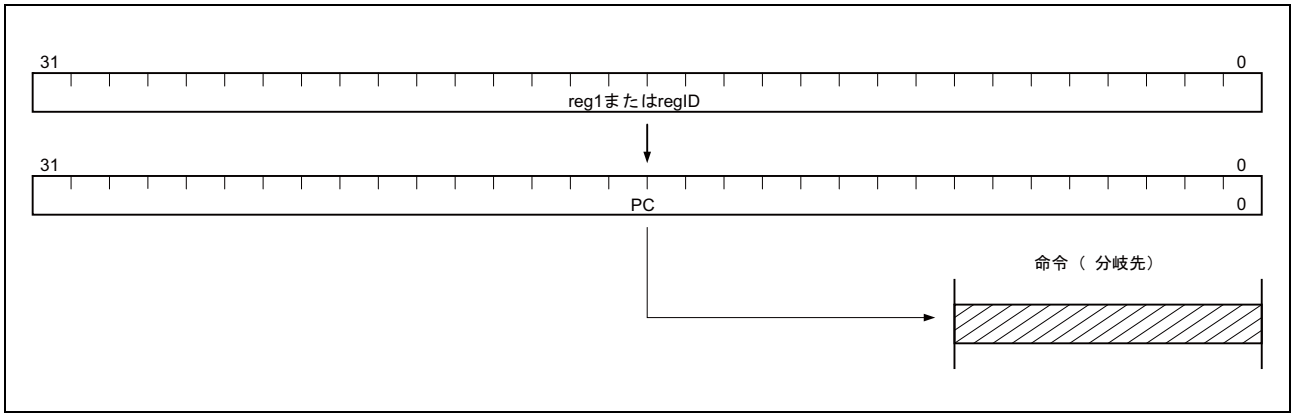


図 2.7　　レジスタ・アドレッシング

(3) ベースト・アドレッシング

命令によって指定される汎用レジスタ（reg1）に、N ビット・ディスプレースメント（dispN）を加算した内容をプログラム・カウンタ（PC）に転送します。このとき、ディスプレースメントは、2 の補数データとして扱われ、それぞれ最上位ビットが符号ビット（S）となります。ディスプレースメントが 32 ビット未満の場合、上位ビットを符号拡張します（N は命令ごとに異なります）。

JMP 命令が、このアドレッシングの対象となります。

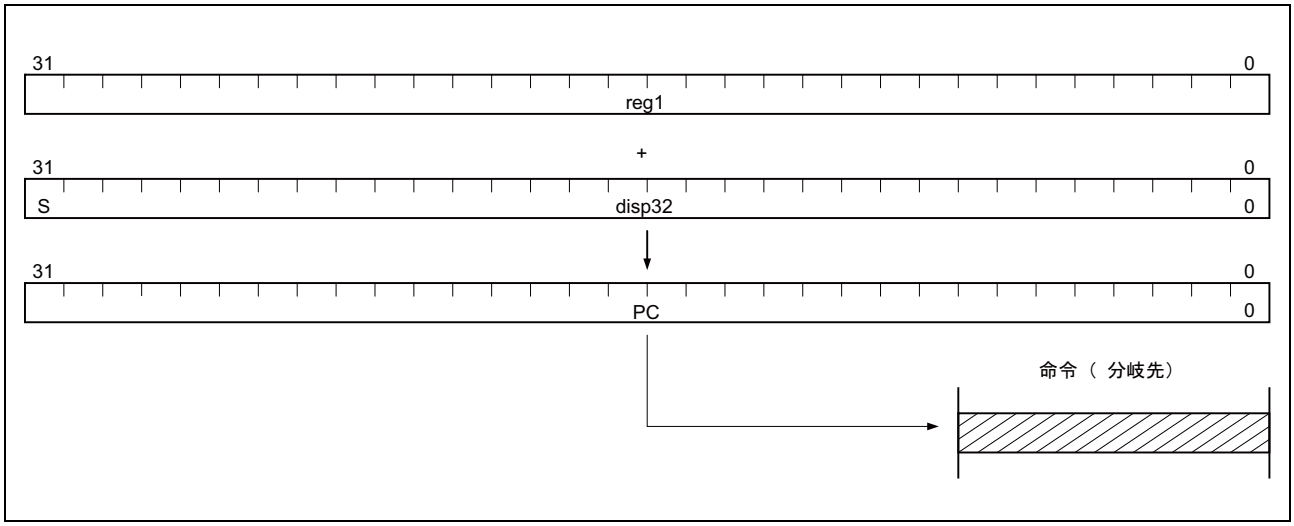


図 2.8　　ベースト・アドレッシング

(4) その他のアドレッシング

命令によって指定される値をプログラム・カウンタ（PC）に転送します。値の指定方法は、それぞれの命令のオペレーション、または説明に記載されています。

CALLT 命令、SYSCALL 命令、TRAP 命令、FETRAP 命令、RIE 命令、および例外発生時の分岐が、このアドレッシングの対象となります。

2.7.3　データ・アドレッシング

命令を実行する際に対象となるレジスタやメモリなどをアクセスするために、次に示す方法があります。

(1) レジスタ・アドレッシング

汎用レジスタ指定フィールドにより指定される汎用レジスタ、またはシステム・レジスタをオペランドとしてアクセスするアドレッシングです。

オペランドに、reg1、reg2、reg3 または regID を含む命令が、このアドレッシングの対象となります。

(2) イミディエイト・アドレッシング

命令コード中に、操作対象となる任意の長さのデータを持つアドレッシングです。

オペランドに、imm5、imm16、vector、または cccc を含む命令が、このアドレッシングの対象となります。

備　考

vector：　例外・ベクタ（00_H ～ $1F_H$）を指定するイミディエイトであり、TRAP 命令、FETRAP 命令、SYSCALL 命令で使用されるオペランドです。データ幅は、各命令で異なります。

cccc：　条件コード指定用の 4 ビット・データであり、CMOV 命令、SASF 命令、SETF 命令で使用されるオペランドです。0 の 1 ビットを上位に付加し、5 ビット・イミディエイト・データとしてオペコード中に割り当てられます。

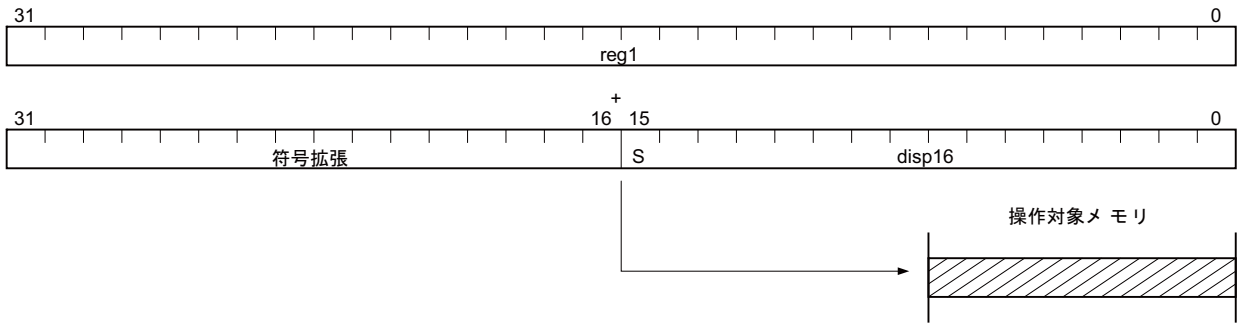
(3) ベースト・アドレッシング

ベースト・アドレッシングには、次に示す 2 種類があります。

(a) タイプ 1

命令コード中のアドレッシング指定フィールドで指定される汎用レジスタ（reg1）の内容とワード長まで符号拡張した N ビット・ディスプレースメント（dispN）の和をオペランド・アドレスとして、操作対象となるメモリへのアクセスを行うアドレッシングです。このとき、ディスプレースメントは、2 の補数データとして扱われ、それぞれ最上位ビットが符号ビット（S）となります。ディスプレースメントが 32 ビット未満の場合、上位ビットを符号拡張します（N は命令ごとに異なります）。

LD 命令、ST 命令、CAXI 命令が、このアドレッシングの対象となります。



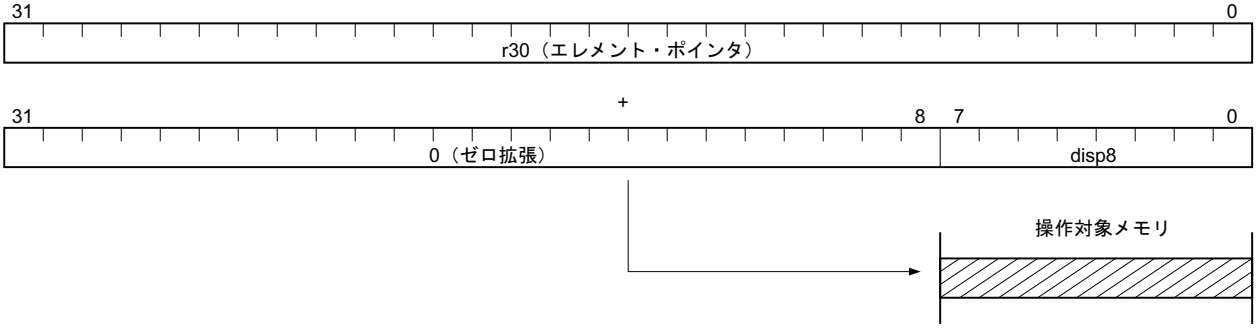
備考　16 ビット・ディスプレースメントの場合です。

図 2.9　　ベースト・アドレッシング（タイプ 1）

(b) タイプ 2

エレメント・ポインタ（r30）の内容とワード長までゼロ拡張した N ビット・ディスプレースメント・データ（dispN）の和をオペランド・アドレスとして、操作対象となるメモリへのアクセスを行うアドレッシングです。ディスプレースメントが 32 ビット未満の場合、上位ビットをゼロ拡張します（N は命令ごとに異なります）。

SLD 命令と SST 命令が、このアドレッシングの対象となります。



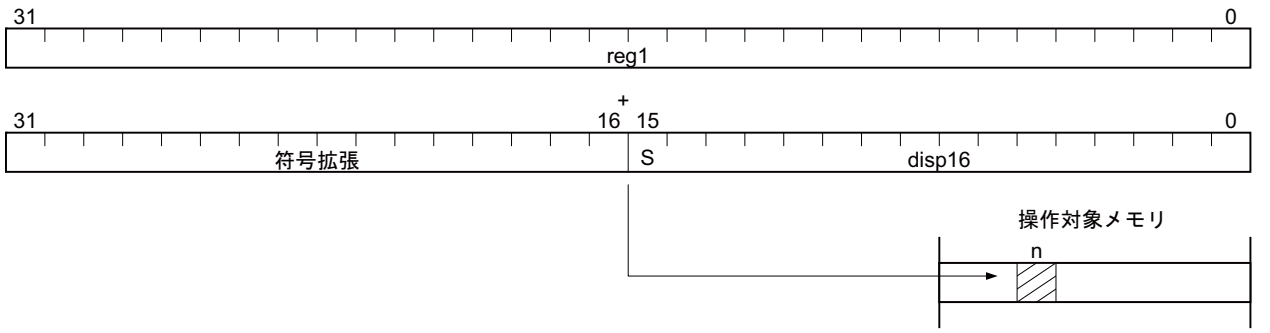
備考　8 ビット・ディスプレースメントの場合です。

図 2.10　　ベースト・アドレッシング（タイプ 2）

(4) ビット・アドレッシング

汎用レジスタ（reg1）の内容とワード長まで符号拡張したNビット・ディスプレースメント（dispN）の和をオペランド・アドレスとして、操作対象となるメモリ空間の1バイト中の1ビット（3ビット・データ「bit #3」で指定）をアクセスするアドレッシングです。このとき、ディスプレースメントは、2の補数データとして扱われ、それぞれ最上位ビットが符号ビット（S）となります。ディスプレースメントが32ビット未満の場合、上位ビットを符号拡張します（Nは命令ごとに異なります）。

CLR1命令、SET1命令、NOT1命令、TST1命令が、このアドレッシングの対象となります。



備考　n: 3ビット・データ（bit#3）で推定されるビット位置（n = 0 ～ 7）
16ビット・ディスプレースメントの場合です。

図 2.11　　ビット・アドレッシング

(5) ポスト・インデックス・インクリメント／デクリメント・アドレッシング

汎用レジスタ（reg1）の内容をオペランド・アドレスとして、操作対象となるメモリへアクセスを行ったあと、汎用レジスタ（reg1）を更新します。汎用レジスタ（reg1）の更新はインクリメントまたはデクリメントで、タイプ 1 ～ 3 の 3 種類があります。

汎用レジスタ（reg1）の値のインクリメントの結果、正の最大値 0xFFFFFFFF を越えた場合は 0x00000000 にラップ・アラウンドし、デクリメントの結果、正の最小値 0x00000000 を越えた場合は 0xFFFFFFFF にラップ・アラウンドします。

(a) タイプ 1

汎用レジスタ（reg1）の内容と、アクセスしたデータ・タイプにしたがう定数（アクセス・データ・サイズ）を加算した結果で汎用レジスタ（reg1）を更新します。アクセスしたデータ・タイプがバイトの場合は 1 を、ハーフワードの場合は 2 を、ワードの場合は 4 を、ダブルワードの場合は 8 を、それぞれ加算します。

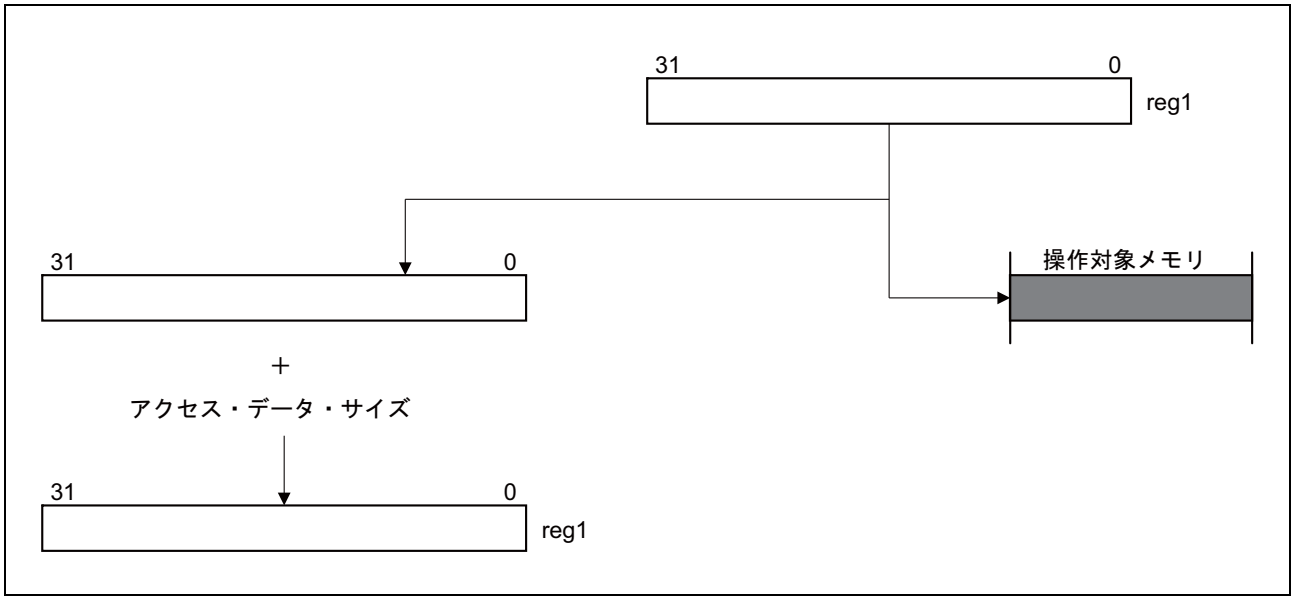


図 2.12　ポスト・インデックス・インクリメント／デクリメント・アドレッシング（タイプ 1）

(b)　タイプ2

汎用レジスタ（reg1）の内容と、アクセスしたデータ・サイズにしたがう定数を減算した結果で汎用レジスタ（reg1）を更新します。アクセスしたデータ・サイズがバイトの場合は1を、ハーフワードの場合は2を、ワードの場合は4を、ダブルワードの場合は8を、それぞれ減算します。

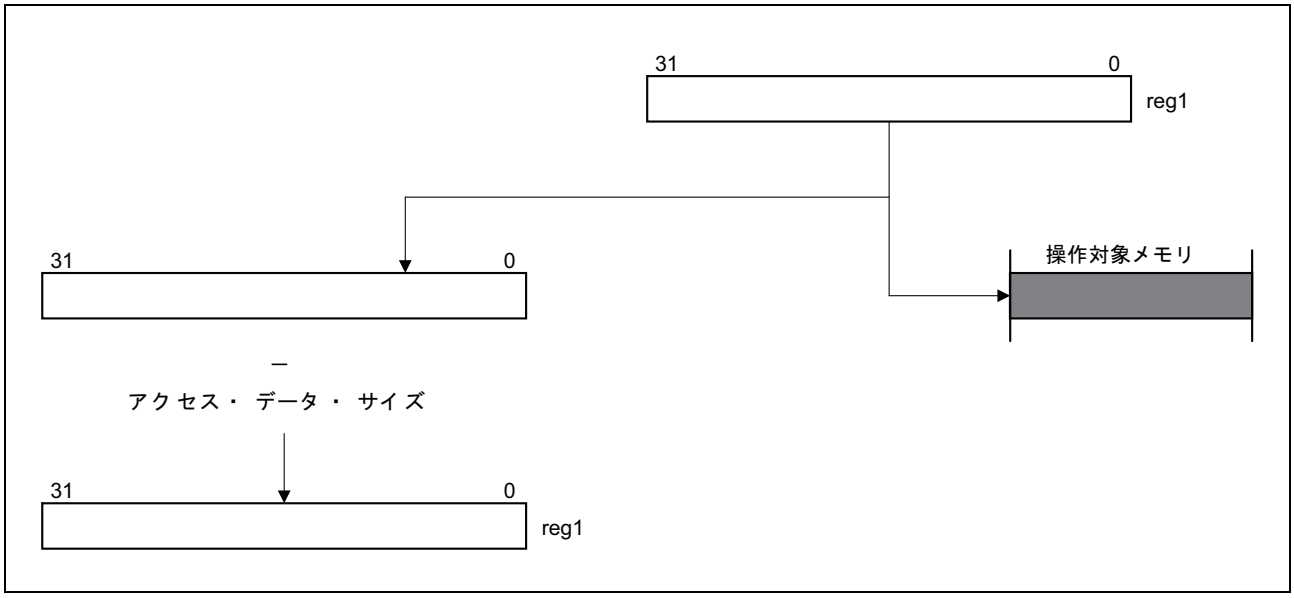


図 2.13　ポスト・インデックス・インクリメント／デクリメント・アドレッシング（タイプ2）

(c) タイプ3

汎用レジスタ（reg1）の内容と、汎用レジスタ（reg2）の内容を加算した結果で汎用レジスタ（reg1）を更新します。汎用レジスタ（reg2）の最上位ビットが1の場合は、負の値を示すので、ポスト・デクリメントとして動作し、汎用レジスタ（reg2）の最上位ビットが0の場合は、正の値を示すので、ポスト・インクリメントとして動作します。汎用レジスタ（reg2）の値は変化しません。

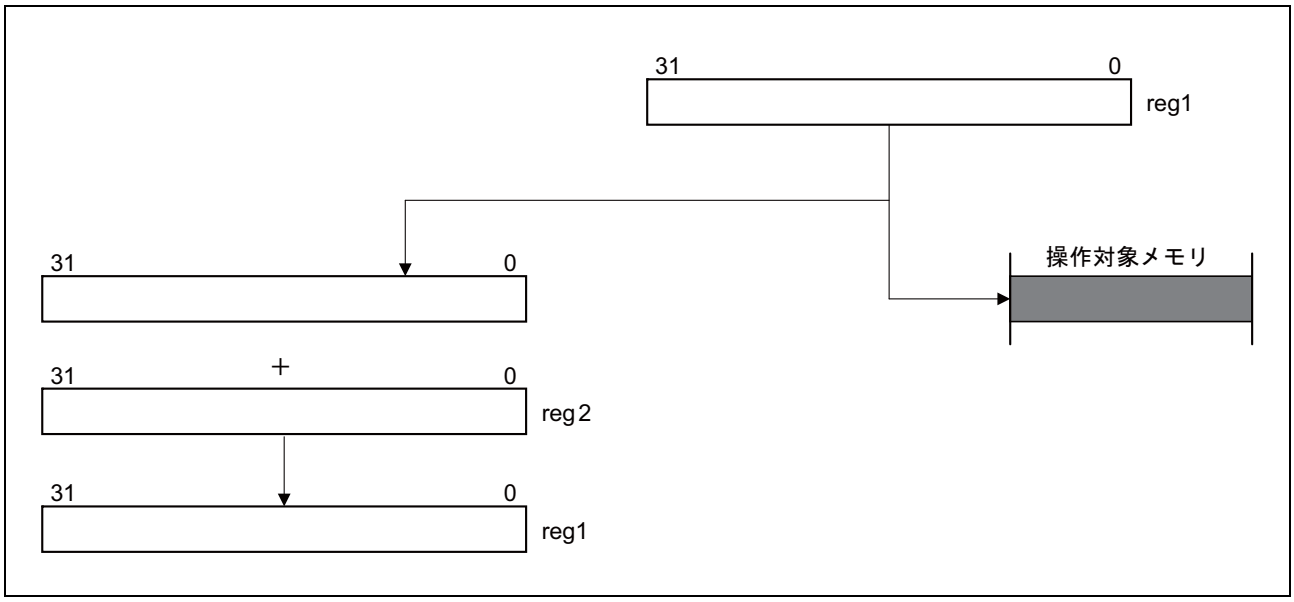


図 2.14　ポスト・インデックス・インクリメント／デクリメント・アドレッシング（タイプ3）

(6) その他のアドレッシング

命令によって指定される値をオペランド・アドレスとして、操作対象となるメモリへのアクセスを行うアドレッシングです。値の指定方法は、それぞれの命令のオペレーション、または説明に記載されています。

SWITCH命令、CALLT命令、SYSCALL命令、PREPARE命令、DISPOSE命令、PUSHSP命令、POPSP命令が、このアドレッシングの対象となります。

2.8　CPU 番号の取得

本 CPU では、マルチプロセッサ構成において CPU を識別する方法を提供しています。

マルチプロセッサ構成の場合には、HTCFG0.PEID を参照することで、プログラム自身がいずれの CPU コアで実行されているのかを知ることが可能です。HTCFG0.PEID はマルチプロセッサ・システム内で、ユニークな数値が割り振られます。

2.9　システム・プロテクション識別子

本 CPU は、メモリ資源や周辺装置などを、システム・プロテクション・グループと呼ばれるグループによって管理し、実行中のプログラムに対していずれのグループに属するかを指定することで、各マシンごとに操作可能なメモリ資源や周辺装置を割り当てることが可能です。

実行中のプログラムは、それぞれ MCFG0.SPID で示されるグループに属しており、メモリ資源や周辺装置への操作は、この SPID を基準に、操作可能かどうかが決定されます。MCFG0.SPID は、スーパバイザによって任意の値が設定されます。

注　意

MCFG0.SPID の値にしたがって、どのようにメモリ資源や周辺装置資源への操作の割り当てられるかについては、ハードウェア仕様によって定められます。

第 3 章　レジスタ・セット

本章では本 CPU に搭載しているプログラム・レジスタとシステム・レジスタについて説明します。

3.1　プログラム・レジスタ

プログラム・レジスタには、汎用レジスタ（r0 ～ r31）とプログラム・カウンタ（PC）があります。汎用レジスタの r0 は常に 0 を保持していますが、r1 ～ r31 のリセット後の値は不定です。

表 3.1　プログラム・レジスター覧

プログラム・レジスタ	名称	機能	説明
汎用レジスタ	r0	ゼロ・レジスタ	常に 0 を保持
	r1	アセンブラ予約レジスタ	アドレス生成用のワーキング・レジスタとして使用
	r2	アドレス／データ変数用レジスタ （使用するリアルタイム OS がこのレジスタを使用していない場合）	
	r3	スタック・ポインタ（SP）	関数コール時のスタック・フレーム生成時に使用
	r4	グローバル・ポインタ（GP）	データ領域のグローバル変数をアクセスするときに使用
	r5	テキスト・ポインタ（TP）	テキスト領域（プログラム・コードを配置する領域）の先頭を示すレジスタとして使用
	r6 ～ r29	アドレス／データ変数用レジスタ	
	r30	エレメント・ポインタ（EP）	メモリをアクセスするときのアドレス生成用ベース・ポインタとして使用
	r31	リンク・ポインタ（LP）	コンパイラが関数コールをするときに使用
プログラム・カウンタ	PC	プログラム実行中の命令アドレスを保持	

備考　アセンブラや C コンパイラで使用される r1、r3 ～ r5、r31 の詳細な説明は、それぞれのソフトウェア開発環境のマニュアルを参照してください。

3.1.1　汎用レジスタ

汎用レジスタとして、r0 ～ r31 の 32 本が用意されています。これらのレジスタは、すべてデータ変数用またはアドレス変数用として利用できます。

汎用レジスタのうち、r0 ～ r5、r30、r31 は、ソフトウェア開発環境において特殊な用途に用いられることを想定しているため、使用する際には次のような注意が必要です。

(1) r0, r3, r30

命令により暗黙的に使用されます。

r0 は常に 0 を保持しているレジスタであり、0 を使用する演算やベース・アドレスが 0 のアドレッシングで使用されます。

r3 は PREPARE 命令、DISPOSE 命令、PUSHSP 命令、POPSP 命令により、暗黙的に使用されます。

r30 は SLD 命令と SST 命令により、メモリをアクセスするときのベース・ポインタとして使用されます。

(2) r1, r4, r5, r31

アセンブラと C コンパイラにより暗黙的に使用されます。

これらのレジスタを使用する際には、レジスタの内容を破壊しないように退避してから使用し、使用後に元に戻す必要があります。

(3) r2

リアルタイム OS が使用する場合があります。使用するリアルタイム OS が r2 を使用していない場合は、アドレス変数用またはデータ変数用レジスタとして利用できます。

3.1.2　PC — プログラム・カウンタ

プログラム実行中の命令アドレスを保持しています。

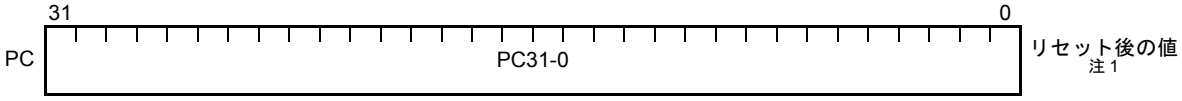


表 3.2　PC レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 1	PC31-1	実行中の命令アドレスです。	R/W	注 1
0	PC0	常に 0 を示します。奇数番地への分岐はできません。	R/W	0

注 1.　詳細は、製品のハードウェアマニュアルを参照してください。

3.2　基本システム・レジスタ

基本システム・レジスタは、CPU の状態制御、例外情報保持などを行います。

基本システム・レジスタへのリード／ライトは、LDSR 命令、STSR 命令により、レジスタ番号と選択識別子からなる、システム・レジスタ番号を指定することで行います。

表 3.3　基本システム・レジスター覧

レジスタ番号（regID, selID）	名称	機能	アクセス権限
SR0, 0	EIPC	EI レベル例外受け付け時の状態退避レジスタ	SV
SR1, 0	EIPSW	EI レベル例外受け付け時の状態退避レジスタ	SV
SR2, 0	FEPC	FE レベル例外受け付け時の状態退避レジスタ	SV
SR3, 0	FEPSW	FE レベル例外受け付け時の状態退避レジスタ	SV
SR5, 0	PSW	プログラム・ステータス・ワード	注 1
SR6, 0	FPSR	（**「3.4　FPU 機能レジスタ」**参照）	CU0 かつ SV
SR7, 0	FPEPC	（**「3.4　FPU 機能レジスタ」**参照）	CU0 かつ SV
SR8, 0	FPST	（**「3.4　FPU 機能レジスタ」**参照）	CU0
SR9, 0	FPCC	（**「3.4　FPU 機能レジスタ」**参照）	CU0
SR10, 0	FPCFG	（**「3.4　FPU 機能レジスタ」**参照）	CU0
SR11, 0	FPEC	（**「3.4　FPU 機能レジスタ」**参照）	CU0 かつ SV
SR13, 0	EIIC	EI レベル例外要因	SV
SR14, 0	FEIC	FE レベル例外要因	SV
SR16, 0	CTPC	CALLT 実行時の状態退避レジスタ	UM
SR17, 0	CTPSW	CALLT 実行時の状態退避レジスタ	UM
SR20, 0	CTBP	CALLT ベース・ポインタ	UM
SR28, 0	EIWR	EI レベル例外用作業レジスタ	SV
SR29, 0	FEWR	FE レベル例外用作業レジスタ	SV
SR31, 0	(BSEL)	（V850E2 シリーズ後方互換のため予約）注 2	SV
SR0, 1	MCFG0	マシン・コンフィグレーション	SV
SR2, 1	RBASE	リセット・ベクタ・ベースアドレス	SV
SR3, 1	EBASE	例外ハンドラ・ベクタ・アドレス	SV
SR4, 1	INTBP	割り込みハンドラ・アドレス・テーブルのベース・アドレス	SV
SR5, 1	MCTL	CPU の制御	SV
SR6, 1	PID	プロセッサ識別子	SV
SR11, 1	SCCFG	SYSCALL の動作設定	SV
SR12, 1	SCBP	SYSCALL ベース・ポインタ	SV
SR0, 2	HTCFG0	スレッド・コンフィグレーション	SV
SR6, 2	MEA	メモリ・エラー・アドレス	SV
SR7, 2	ASID	アドレス空間識別子	SV
SR8, 2	MEI	メモリ・エラー情報	SV

注 1.　ビットによってアクセス権限が異なります。詳細は、**「3.2　基本システム・レジスタ」**の**「(5)　PSW — プログラム・ステータス・ワード」**を参照してください。

注 2.　V850E2 シリーズの後方互換のため予約しています。読み込みは常に 0 を読み出し、書き込みは無視します。

(1) EIPC — EI レベル例外受け付け時の状態退避レジスタ

EI レベル例外を受け付けた場合、EI レベル例外が発生したときに実行していた命令、あるいはその次の命令のアドレスが退避されます（「4.1.3　例外の実行形態」を参照してください）。

EI レベル例外時状態退避レジスタは、1 組であるため、多重例外処理を行う場合はプログラムによってこれらのレジスタの内容を退避する必要があります。

EIPC レジスタには必ず偶数番地を設定してください。奇数番地の指定はできません。

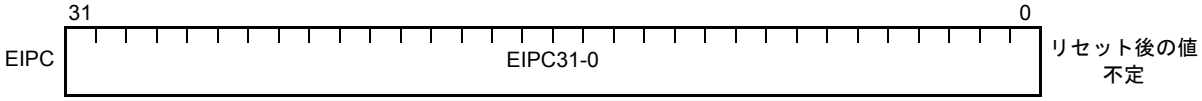


表 3.4　EIPC レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 1	EIPC31-1	EI レベル例外受け付け時の復帰 PC を示します。	R/W	不定
0	EIPC0	EI レベル例外受け付け時の復帰 PC を示します。 常に 0 を設定してください。1 を設定した場合でも、EIRET 命令の実行により PC に転送される値は、0 となります。	R/W	不定

(2) EIPSW — EI レベル例外受け付け時の状態退避レジスタ

EI レベル例外を受け付けた場合、そのときの PSW の内容が退避されます。

EI レベル例外時状態退避レジスタは、1 組であるため、多重例外処理を行う場合はプログラムによってこれらのレジスタの内容を退避する必要があります。

注　意

ビット 11 ～ 9 はデバッグ機能にかかわるため、通常は変更できません。

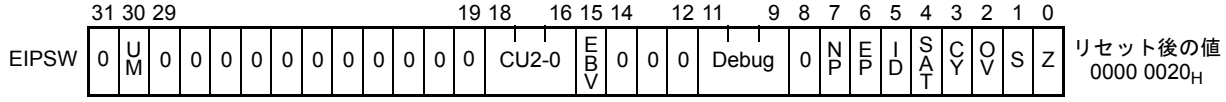


表 3.5　EIPSW レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31	—	（将来のための予約です。必ず 0 を設定してください）	R	0
30	UM	EI レベル例外受け付け時の PSW.UM ビットを退避します。	R/W	0
29 ～ 19	—	（将来のための予約です。必ず 0 を設定してください）	R	0
18 ～ 16	CU2-0	EI レベル例外受け付け時の PSW.CU2-0 フィールドを退避します。（CU2-1 ビットは将来のための予約です。必ず 0 を設定してください）	R/W	0
15	EBV	EI レベル例外受け付け時の PSW.EBV ビットを退避します。	R/W	0
14 ～ 12	—	（将来のための予約です。必ず 0 を設定してください）	R	0
11 ～ 9	Debug	EI レベル例外受け付け時の PSW.Debug フィールドを退避します。	R/W	0
8	—	（将来のための予約です。必ず 0 を設定してください）	R	0
7	NP	EI レベル例外受け付け時の PSW.NP ビットを退避します。	R/W	0
6	EP	EI レベル例外受け付け時の PSW.EP ビットを退避します。	R/W	0
5	ID	EI レベル例外受け付け時の PSW.ID ビットを退避します。	R/W	1
4	SAT	EI レベル例外受け付け時の PSW.SAT ビットを退避します。	R/W	0
3	CY	EI レベル例外受け付け時の PSW.CY ビットを退避します。	R/W	0
2	OV	EI レベル例外受け付け時の PSW.OV ビットを退避します。	R/W	0
1	S	EI レベル例外受け付け時の PSW.S ビットを退避します。	R/W	0
0	Z	EI レベル例外受け付け時の PSW.Z ビットを退避します。	R/W	0

(3) FEPC — FE レベル例外受け付け時の状態退避レジスタ

FE レベル例外を受け付けた場合、FE レベル例外が発生したときに実行していた命令、あるいはその次の命令のアドレスが退避されます（「4.1.3　例外の実行形態」を参照してください）。

FE レベル例外時状態退避レジスタは、1 組であるため、多重例外処理を行う場合はプログラムによってこれらのレジスタの内容を退避する必要があります。

FEPC レジスタには必ず偶数番地を設定してください。奇数番地の指定はできません。

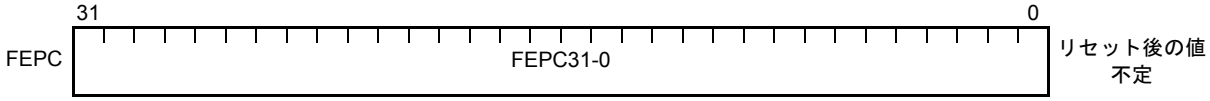


表 3.6　FEPC レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 1	FEPC31-1	FE レベル例外受け付け時の復帰 PC を示します。	R/W	不定
0	FEPC0	FE レベル例外受け付け時の復帰 PC を示します。 常に 0 を設定してください。1 を設定した場合でも、FERET 命令の実行により PC に転送される値は、0 となります。	R/W	不定

(4) FEPSW — FE レベル例外受け付け時の状態退避レジスタ

FE レベル例外を受け付けた場合、そのときの PSW の内容が退避されます。

FE レベル例外時状態退避レジスタは、1 組であるため、多重例外処理を行う場合はプログラムによってこれらのレジスタの内容を退避する必要があります。

注　意

ビット 11 ～ 9 はデバッグ機能にかかわるため、通常は変更できません。

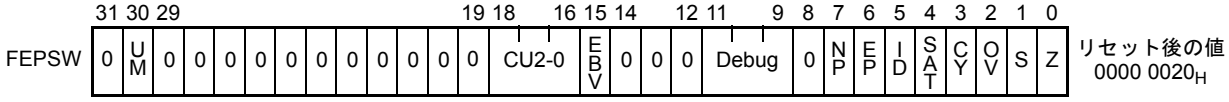


表 3.7　FEPSW レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31	—	（将来のための予約です。必ず 0 を設定してください）	R	0
30	UM	FE レベル例外受け付け時の PSW.UM ビットを退避します。	R/W	0
29 ～ 19	—	（将来のための予約です。必ず 0 を設定してください）	R	0
18 ～ 16	CU2-0	FE レベル例外受け付け時の PSW.CU2-0 フィールドを退避します。（CU2-1 ビットは将来のための予約です。必ず 0 を設定してください）	R/W	0
15	EBV	FE レベル例外受け付け時の PSW.EBV ビットを退避します。	R/W	0
14 ～ 12	—	（将来のための予約です。必ず 0 を設定してください）	R	0
11 ～ 9	Debug	FE レベル例外受け付け時の PSW.Debug フィールドを退避します。	R/W	0
8	—	（将来のための予約です。必ず 0 を設定してください）	R	0
7	NP	FE レベル例外受け付け時の PSW.NP ビットを退避します。	R/W	0
6	EP	FE レベル例外受け付け時の PSW.EP ビットを退避します。	R/W	0
5	ID	FE レベル例外受け付け時の PSW.ID ビットを退避します。	R/W	1
4	SAT	FE レベル例外受け付け時の PSW.SAT ビットを退避します。	R/W	0
3	CY	FE レベル例外受け付け時の PSW.CY ビットを退避します。	R/W	0
2	OV	FE レベル例外受け付け時の PSW.OV ビットを退避します。	R/W	0
1	S	FE レベル例外受け付け時の PSW.S ビットを退避します。	R/W	0
0	Z	FE レベル例外受け付け時の PSW.Z ビットを退避します。	R/W	0

(5) PSW — プログラム・ステータス・ワード

PSW（プログラム・ステータス・ワード）は、プログラムの状態（命令実行の結果）を示すフラグや CPU の動作状態を示すビットの集合です（フラグとは条件命令（Bcond や CMOV など）によって参照される PSW 上のビットを示します）。

注　意

1. **LDSR 命令を使用して、このレジスタのビット 7 ～ 0 の内容を変更した場合は、LDSR 命令の直後の命令から変更内容が有効となります。**
2. **PSW レジスタは、ビットごとにアクセス権限が違います。すべてのビットは読み出しは常に行えますが、書き込みは特定の条件でのみ書き込み可能なものが存在します。各ビットのアクセス権限は、表 3.8 を参照してください。**

表 3.8　PSW レジスタ・ビットのアクセス権限

ビット		読み出し時アクセス権限	書き込み時アクセス権限
30	UM	UM	SV 注 1
18 ～ 16	CU2-0		SV 注 1
15	EBV		SV 注 1
11 ～ 9	Debug		特殊 注 1
7	NP		SV 注 1
6	EP		SV 注 1
5	ID		SV 注 1
4	SAT		UM
3	CY		UM
2	OV		UM
1	S		UM
0	Z		UM

注 1.　PSW レジスタ全体はアクセス権限が UM であるため、PSW.UM = 1 のとき LDSR で書き込みを行っても PIE 例外は起きません。書き込みが無視されます。

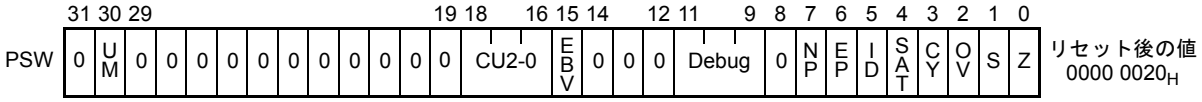


表 3.9　PSW レジスタの内容（1/2）

ビット	名称	意味	R/W	リセット後の値
31	—	（将来のための予約です。必ず 0 を設定してください）	R	0
30	UM	CPU がユーザモード（UM モード）にあることを示します。 0：スーパバイザ・モード 1：ユーザ・モード	R/W	0
29 ～ 19	—	（将来のための予約です。必ず 0 を設定してください）	R	0
18 ～ 16	CU2-CU0	コプロセッサ使用権を示します。コプロセッサに対応するビットが 0 のとき、そのコプロセッサ命令の実行、コプロセッサ・リソース（システム･レジスタ）へのアクセスで、コプロセッサ使用不可例外を発生します。 CU2 ビット 18：（将来のための予約です。必ず 0 を設定してください） CU1 ビット 17：（将来のための予約です。必ず 0 を設定してください） CU0 ビット 16：FPU	R/W	000
15	EBV	リセット・ベクタ、例外ベクタの動作を示します。詳細は、**「(17) RBASE — リセット・ベクタ・ベースアドレス」**、**「(18)　EBASE — 例外ハンドラ・ベクタ・アドレス」**を参照してください。	R/W	0
14 ～ 12	—	（将来のための予約です。必ず 0 を設定してください）	R	0
11 ～ 9	Debug	開発ツール向けのデバッグ機能で使用します。通常は、0 を設定してください。	—	0
8	—	（将来のための予約です。必ず 0 を設定してください）	R	0
7	NP	FE レベル例外の受け付けを禁止します。FE レベル例外が受け付けられるとセット（1）され、EI レベル例外、FE レベル例外の受け付けを禁止します。NP ビットによって、受け付けが禁止される例外は、「表 4.1　例外要因一覧」を参照してください。 0：FE レベル例外の受け付けを許可する 1：FE レベル例外の受け付けを禁止する	R/W	0
6	EP	割り込みコントローラで制御される割り込み以外の例外処理中であることを示します。該当する例外の発生でセット（1）されます。なお、このビットはセット（1）されても例外要求の受け付けには影響しません。 0：割り込み以外の例外処理中ではない 1：割り込み以外の例外処理中である	R/W	0
5	ID	EI レベル例外の受け付けを禁止します。EI レベル例外、FE レベル例外が受け付けられるとセット（1）され、EI レベル例外の受け付けを禁止します。ID ビットによって、受け付けが禁止される例外は、「表 4.1　例外要因一覧」を参照してください。また、通常のプログラムや、割り込み処理中にクリティカル・セクションとして、EI レベル例外の受け付けを禁止する場合にも使用されます。DI 命令の実行によってセット（1）し、EI 命令の実行によってクリア（0）します。 EI 命令、DI 命令による ID ビットの変更は、次の命令から有効となります。 0：EI レベル例外の受け付けを許可する 1：EI レベル例外の受け付けを禁止する	R/W	1
4	SAT 注 1	飽和演算命令の演算結果がオーバフローし、演算結果が飽和していることを示します。累積フラグのため、飽和演算命令で演算結果が飽和するとセット（1）され、以降の命令の演算結果が飽和しなくてもクリア（0）されません。クリア（0）する場合は、LDSR 命令により行います。なお、算術演算命令の実行では、セット（1）もクリア（0）も行いません。 0：飽和していない 1：飽和している	R/W	0

表 3.9　PSW レジスタの内容 (2/2)

ビット	名称	意味	R/W	リセット後の値
3	CY	演算結果にキャリー、またはボローがあったかどうかを示します。 0：キャリー、およびボローが発生していない 1：キャリー、またはボローが発生した	R/W	0
2	OV 注 1	演算中にオーバフローが発生したかどうかを示します。 0：オーバフローが発生していない 1：オーバフローが発生した	R/W	0
1	S 注 1	演算の結果が負かどうかを示します。 0：演算の結果は、正または 0 であった 1：演算の結果は負であった	R/W	0
0	Z	演算の結果が 0 かどうかを示します。 0：演算の結果は 0 でなかった 1：演算の結果は 0 であった	R/W	0

注 1.　飽和演算時の OV フラグと S フラグの内容で飽和処理した演算結果が決まります。また、飽和演算時に OV フラグがセット（1）された場合だけ、SAT フラグはセット（1）されます。

演算結果の状態	フラグの状態			飽和処理をした演算結果
	SAT	OV	S	
正の最大値を越えた	1	1	0	7FFF FFFF$_H$
負の最大値を越えた	1	1	1	8000 0000$_H$
正（最大値を越えない）	演算前の値を	0	0	演算結果そのもの
負（最大値を越えない）	保持		1	

(6) EIIC — EI レベル例外要因

EIIC レジスタは、EI レベルの例外が発生した場合に、その要因を保持するレジスタです。EIIC レジスタが保持する値は、例外要因ごとにコード化された例外要因コードです（「表 4.1　例外要因一覧」参照）。

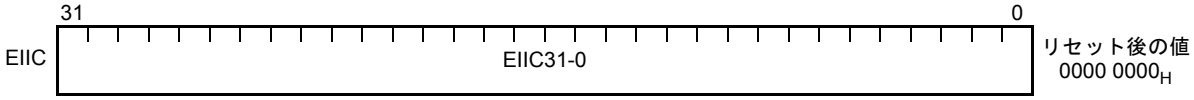


表 3.10　EIIC レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 0	EIIC31-0	EI レベル例外受け付け時に、例外要因コードが格納されます。 EIIC15 ～ 0 は、**表 4.1** で示す例外要因コードが格納されます。 EIIC31 ～ 16 は、例外ごとに定義された詳細な例外要因コードが格納されます。 例外にかかわる機能で特に定義がない場合は、0 が設定されます。	R/W	0

(7) FEIC — FE レベル例外要因

FEIC レジスタは、FE レベルの例外が発生した場合に、その要因を保持するレジスタです。FEIC レジスタが保持する値は、例外要因ごとにコード化された例外要因コードです（「表 4.1　例外要因一覧」参照）。

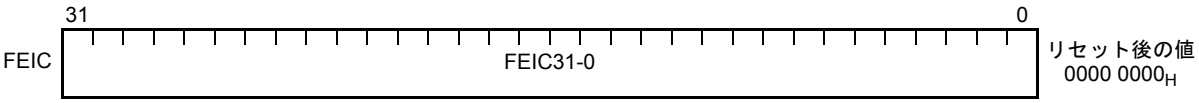


表 3.11　FEIC レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 0	FEIC31-0	FE レベル例外受け付け時に、例外要因コードが格納されます。 FEIC15 ～ 0 は、**表 4.1** で示す例外要因コードが格納されます。 FEIC31 ～ 16 は、例外ごとに定義された詳細な例外要因コードが格納されます。例外にかかわる機能で特に定義がない場合は、0 が設定されます。	R/W	0

(8) CTPC — CALLT 実行時の状態退避レジスタ

CALLT 命令が実行されると、CALLT 命令の次の命令のアドレスが CTPC に退避されます。

CTPC レジスタには必ず偶数番地を設定してください。奇数番地の指定はできません。

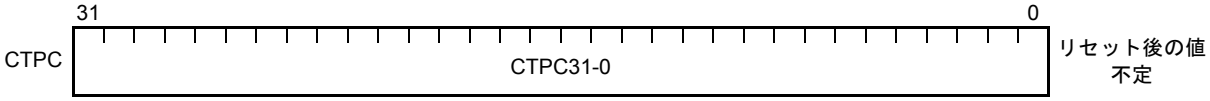


表 3.12　CTPC レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 1	CTPC31-1	CALLT 命令の次の命令の PC を示します。	R/W	不定
0	CTPC0	CALLT 命令の次の命令の PC を示します。 常に 0 を設定してください。1 を設定した場合でも、CTRET 命令の実行により PC に転送される値は、0 となります。	R/W	不定

(9) CTPSW — CALLT 実行時の状態退避レジスタ

CALLT 命令が実行されると、PSW（プログラム・ステータス・ワード）の一部が CTPSW に退避されます。

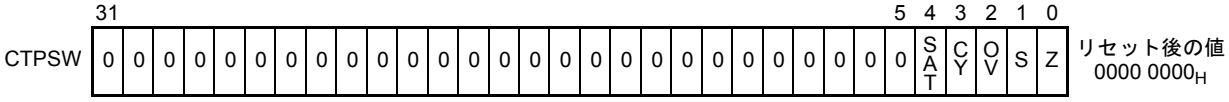


表 3.13　CTPSW レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 5	—	（将来のための予約です。必ず 0 を設定してください）	R	0
4	SAT	CALLT 命令実行時の PSW.SAT ビットを退避します。	R/W	0
3	CY	CALLT 命令実行時の PSW.CY ビットを退避します。	R/W	0
2	OV	CALLT 命令実行時の PSW.OV ビットを退避します。	R/W	0
1	S	CALLT 命令実行時の PSW.S ビットを退避します。	R/W	0
0	Z	CALLT 命令実行時の PSW.Z ビットを退避します。	R/W	0

(10) CTBP — CALLT ベース・ポインタ

CTBP レジスタは、CALLT 命令のテーブル・アドレスの指定、ターゲット・アドレスの生成に使用されます。

CTBP レジスタには必ずハーフワード・アドレスを設定してください。

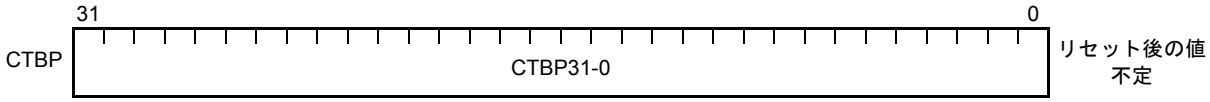


表 3.14　CTBP レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 1	CTBP31-1	CALLT 命令のベース・ポインタ・アドレスです。 CALLT 命令で利用するテーブルの先頭アドレスを示します。	R/W	不定
0	CTBP0	CALLT 命令のベース・ポインタ・アドレスです。 CALLT 命令で利用するテーブルの先頭アドレスを示します。 常に 0 を設定してください。	R	0

(11) ASID — アドレス空間識別子

アドレス空間識別子です。メモリ管理機能で提供されるアドレス空間の識別のために使用します。

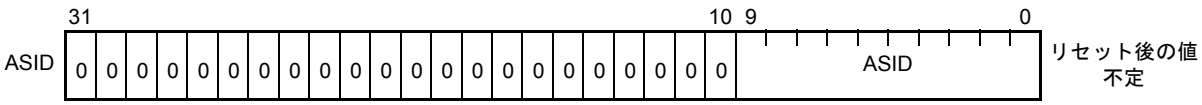


表 3.15　ASID レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 10	—	（将来のための予約です。必ず 0 を設定してください）	R	0
9 ～ 0	ASID	アドレス空間識別子です。	R/W	不定

(12) EIWR — EI レベル例外用作業レジスタ

EIWR レジスタは、EI レベルの例外が発生したときの作業用レジスタです。

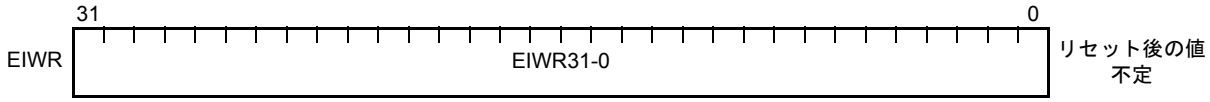


表 3.16　EIWR レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 0	EIWR31-0	EI レベル例外中に任意に利用可能な作業用レジスタです。汎用レジスタの一時退避などに利用してください。	R/W	不定

(13) FEWR — FE レベル例外用作業レジスタ

FEWR レジスタは、FE レベルの例外が発生したときの作業用レジスタです。

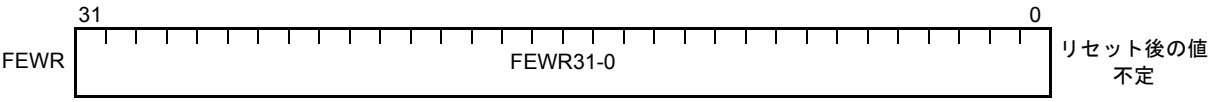


表 3.17　FEWR レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 0	FEWR31-0	FE レベル例外中に任意に利用可能な作業用レジスタです。汎用レジスタの一時退避などに利用してください。	R/W	不定

(14) HTCFG0 — スレッド・コンフィグレーション

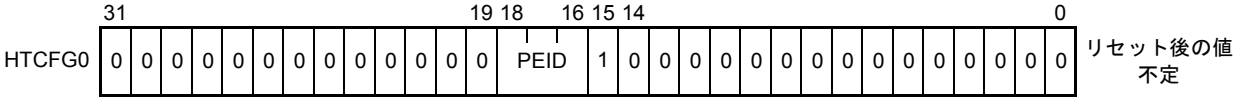


表 3.18　HTCFG0 レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 19	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
18 ～ 16	PEID	プロセッサ・エレメント番号を示します。	R	注 1
15	—	（将来のための予約です。必ず 1 を設定してください。）	R	1
14 ～ 0	—	（将来のための予約です。必ず 0 を設定してください。）	R	0

注 1.　製品仕様で定められた CPU のプロセッサ識別子が読み出せます。書き込みは行えません。詳細は、製品のハードウェアマニュアルを参照してください。

(15) MEA — メモリ・エラー・アドレス

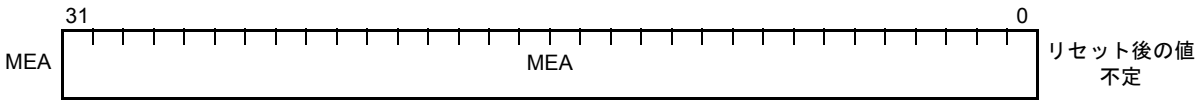


表 3.19　MEA レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 0	MEA	MAE（ミスアライン）／ MPU 違反時のアドレスを保存します。	R/W	不定

(16) MEI — メモリ・エラー情報

ミスアライン例外（MAE）、メモリ保護例外（MDP）発生時に、例外を引き起こした命令の情報を格納します。

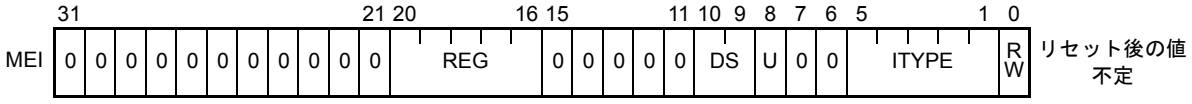


表 3.20　MEI レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 21	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
20 ～ 16	REG	例外を引き起こした命令のソース・レジスタ番号、またはデスティネーション・レジスタ番号を示します。 詳細は、**表 3.21** を参照してください。	R/W	不定
15 ～ 11	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
10、9	DS	例外を引き起こした命令のデータ・タイプを示します[注 1]。 0：バイト（8 ビット） 1：ハーフ・ワード（16 ビット） 2：ワード（32 ビット） 3：ダブル・ワード（64 ビット） 詳細は、**表 3.21** を参照してください。	R/W	不定
8	U	例外を引き起こした命令の符号拡張方式を示します。 0：Signed 1：Unsigned 詳細は、**表 3.21** を参照してください。	R/W	不定
7、6	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
5 ～ 1	ITYPE	例外を引き起こした命令を示します。 詳細は、**表 3.21** を参照してください。	R/W	不定
0	RW	例外を引き起こした命令のオペレーションが、リード（Load-memory）だったのか、ライト（Store-memory）だったのかを示します。 0：リード（Load-memory） 1：ライト（Store-memory） 詳細は、**表 3.21** を参照してください。	R/W	不定

注 1.　ハードウェアによってアクセスが分割されるような場合においても、命令の示すデータ・タイプが格納されます。

表 3.21　例外を引き起こした命令と MEI レジスタの値

命令	REG	DS	U	RW	ITYPE
SLD.B	dst	0（Byte）	0（Signed）	0（Read）	00000b
SLD.BU	dst	0（Byte）	1（Unsigned）	0（Read）	00000b
SLD.H	dst	1（Half-word）	0（Signed）	0（Read）	00000b
SLD.HU	dst	1（Half-word）	1（Unsigned）	0（Read）	00000b
SLD.W	dst	2（Word）	0（Signed）	0（Read）	00000b
SST.B	src	0（Byte）	0（Signed）	1（Write）	00000b
SST.H	src	1（Half-word）	0（Signed）	1（Write）	00000b
SST.W	src	2（Word）	0（Signed）	1（Write）	00000b
LD.B（disp16）	dst	0（Byte）	0（Signed）	0（Read）	00001b
LD.BU（disp16）	dst	0（Byte）	1（Unsigned）	0（Read）	00001b
LD.H（disp16）	dst	1（Half-word）	0（Signed）	0（Read）	00001b
LD.HU（disp16）	dst	1（Half-word）	1（Unsigned）	0（Read）	00001b
LD.W（disp16）	dst	2（Word）	0（Signed）	0（Read）	00001b
ST.B（disp16）	src	0（Byte）	0（Signed）	1（Write）	00001b
ST.H（disp16）	src	1（Half-word）	0（Signed）	1（Write）	00001b
ST.W（disp16）	src	2（Word）	0（Signed）	1（Write）	00001b
LD.B（disp23）	dst	0（Byte）	0（Signed）	0（Read）	00010b
LD.BU（disp23）	dst	0（Byte）	1（Unsigned）	0（Read）	00010b
LD.H（disp23）	dst	1（Half-word）	0（Signed）	0（Read）	00010b
LD.HU（disp23）	dst	1（Half-word）	1（Unsigned）	0（Read）	00010b
LD.W（disp23）	dst	2（Word）	0（Signed）	0（Read）	00010b
ST.B（disp23）	src	0（Byte）	0（Signed）	1（Write）	00010b
ST.H（disp23）	src	1（Half-word）	0（Signed）	1（Write）	00010b
ST.W（disp23）	src	2（Word）	0（Signed）	1（Write）	00010b
LD.DW（disp23）	dst	3（Double-word）	0（Signed）	0（Read）	00010b
ST.DW（disp23）	src	3（Double-word）	0（Signed）	1（Write）	00010b
LDL.W	dst	2（Word）	0（Signed）	0（Read）	00111b
STC.W	src	2（Word）	0（Signed）	1（Write）	00111b
CAXI	dst	2（Word）	0（Signed）	0（Read）注 1	01000b
SET1	0	0（Byte）	0（Signed）	0（Read）注 1	01001b
CLR1	0	0（Byte）	0（Signed）	0（Read）注 1	01001b
NOT1	0	0（Byte）	0（Signed）	0（Read）注 1	01001b
TST1	0	0（Byte）	0（Signed）	0（Read）	01001b
PREPARE	0	2（Word）	0（Signed）	1（Write）	01100b
DISPOSE	0	2（Word）	0（Signed）	0（Read）	01100b
PUSHSP	0	2（Word）	0（Signed）	1（Write）	01101b
POPSP	0	2（Word）	0（Signed）	0（Read）	01101b
SWITCH	0	1（Half-word）	0（Signed）	0（Read）	10000b
CALLT	0	1（Half-word）	1（Unsigned）	0（Read）	10001b
SYSCALL	0	2（Word）	0（Signed）	0（Read）	10010b
割り込み（テーブル参照）注 2	0	2（Word）	0（Signed）	0（Read）	10101b

注 1.　リード時点で例外が発生します。
注 2.　テーブル参照方式の割り込みベクタ読み込み時

備考　dst：デスティネーション・レジスタ番号、src：ソース・レジスタ番号

(17) RBASE — リセット・ベクタ・ベースアドレス

リセット時のリセット・ベクタ・アドレスを示すレジスタです。PSW.EBV ビットがクリア（0）されている場合、このベクタ・アドレスは例外ベクタ・アドレスとしても使用されます。

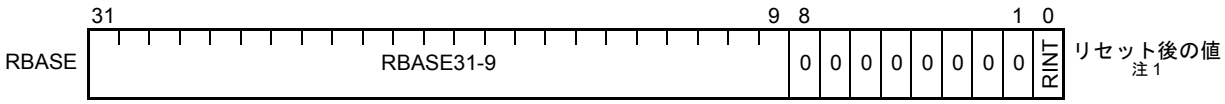


表 3.22　RBASE レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 9	RBASE31-9	リセット時のリセット・ベクタを示します。このアドレスは PSW.EBV = 0 の時、例外ベクタとしても使用されます。 RBASE8 ～ 0 は、暗黙的に 0 が利用されます。	R	注 1
8 ～ 1	—	（将来のための予約です。必ず 0 を設定してください）	R	0
0	RINT	RINT ビットがセットされている場合、割り込み処理の例外ハンドラ・アドレスの縮小を行います。詳細は、**「4.5.1 (1)　直接ベクタ方式」**を参照してください。このビットは PSW.EBV = 0 の時に有効です。	R	注 1

注 1.　リセット後の値はハードウェア仕様によって定められます。詳細は製品のハードウェアマニュアルを参照してください。

(18) EBASE — 例外ハンドラ・ベクタ・アドレス

例外ハンドラのベクタ・アドレスを示すレジスタです。PSW.EBV ビットがセット（1）されている場合に有効です。

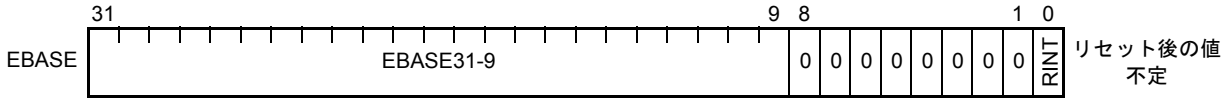


表 3.23　EBASE レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 9	EBASE31-9	例外ハンドラ・ルーチンのアドレスがこのレジスタで指定されたベース・アドレスに、各例外のオフセット・アドレスを加えたアドレスに変更されます。 EBASE8 ～ 0 は、暗黙的に 0 が利用されます。	R/W	不定
8 ～ 1	—	（将来のための予約です。必ず 0 を設定してください）	R	0
0	RINT	RINT ビットがセットされている場合、割り込み処理の例外ハンドラ・アドレスの縮小を行います。詳細は、**「4.5.1 (1)　直接ベクタ方式」**を参照してください。	R/W	不定

(19) INTBP — 割り込みハンドラ・アドレス・テーブルのベース・アドレス

割込みハンドラ・アドレスの選択方式として、テーブル参照方式を選択した場合の、テーブルのベース・アドレスを示すレジスタです。

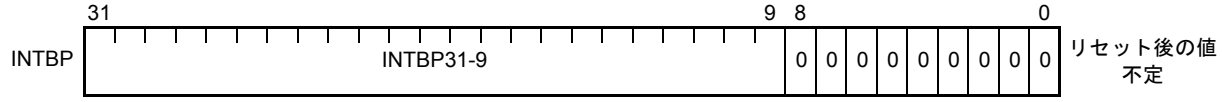


表 3.24　INTBP レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 9	INTBP31-9	割り込みのテーブル参照方式のベース・ポインタ・アドレスです。 テーブル参照方式を指定した割り込み（EIINT0 ～ 511）受け付け時の例外ハンドラ決定に利用するテーブルの先頭アドレスを示します。 INTBP8 ～ 0 は、暗黙的に 0 が利用されます。	R/W	不定
8 ～ 0	—	（将来のための予約です。必ず 0 を設定してください）	R	0

(20) PID — プロセッサ識別子

PID レジスタは、CPU 固有のプロセッサ識別子を保持します。PID レジスタはリードのみ可能です。

注　意

PID レジスタは、搭載された CPU コア、および CPU コアのコンフィグレーションを識別するための情報です。ソフトウェアの挙動を、PID レジスタの情報によって、動的に変化させるような利用方法を想定していません。

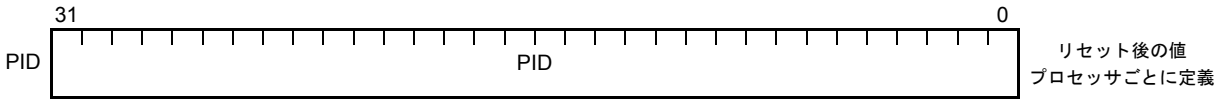


表 3.25　PID レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 24	PID	**アーキテクチャ識別子** プロセッサの属するアーキテクチャを示す識別子です。	R	注 1
23 ～ 8		**機能識別子** プロセッサの持つ機能を示す識別子です。 ビットごとに定義された機能の搭載／非搭載を示します （1：搭載、0：非搭載）。 ビット 23 ～ 10　予約 ビット 9　　単精度浮動小数点演算機能 ビット 8　　メモリ保護機能（MPU）	R	注 1
7 ～ 0		**バージョン識別子** プロセッサのバージョンを示す識別子です。	R	注 1

注 1.　詳細は、製品のハードウェアマニュアルを参照してください。

(21) SCCFG — SYSCALL の動作設定

SYSCALL 命令に関する動作設定を行います。SYSCALL 命令の使用前に必ず適切な値を設定してください。

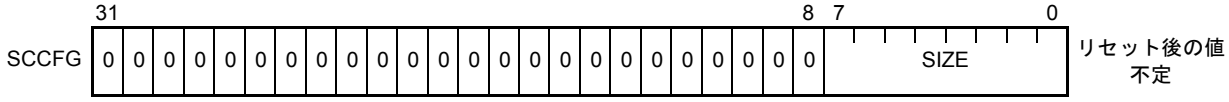


表 3.26　SCCFG レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 8	—	（将来のための予約です。必ず 0 を設定してください）	R	0
7 ～ 0	SIZE	SYSCALL 命令が参照するテーブルの最大エントリ数を指定します。 SYSCALL が参照する最大エントリ数は、SIZE が 0 の場合は 1 エントリ、255 の場合は 256 エントリです。SYSCALL 命令で分岐する関数の数に合わせて、最大エントリ数を適切に設定することで、メモリ領域を有効に活用できます。 最大エントリ数を越えるベクタが SYSCALL 命令で指定された場合には、先頭のエントリが選択されます。先頭のエントリには、エラー処理ルーチンを配置してください。	R/W	不定

(22) SCBP — SYSCALL ベース・ポインタ

SCBP レジスタは、SYSCALL 命令のテーブル・アドレスの指定、ターゲット・アドレスの生成に使用されます。SYSCALL 命令の使用前に、必ず適切な値を設定してください。

SCBP レジスタには必ずワード・アドレスを設定してください。

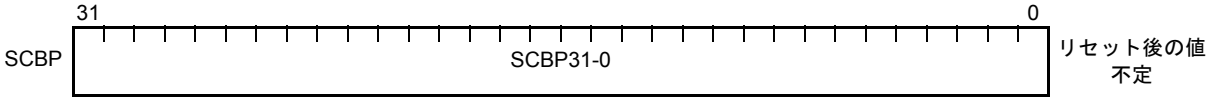


表 3.27　SCBP レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 2	SCBP31-2	SYSCALL 命令のベース・ポインタ・アドレスです。 SYSCALL 命令で利用するテーブルの先頭アドレスを示します。	R/W	不定
1、0	SCBP1-0	SYSCALL 命令のベース・ポインタ・アドレスです。 SYSCALL 命令で利用するテーブルの先頭アドレスを示します。 常に 0 を設定してください。	R	0

(23) MCFG0 — マシン・コンフィグレーション

CPU のコンフィグレーションを表すレジスタです。

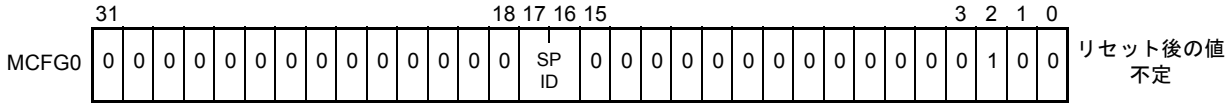


表 3.28　MCFG0 レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 18	—	（将来のための予約です。必ず 0 を設定してください）	R	0
17、16	SPID	システム・プロテクション番号を示します。 詳細は製品のハードウェアマニュアルを参照してください。	R/W	注 1
15 ～ 3	—	（将来のための予約です。必ず 0 を設定してください）	R	0
2	—	（将来のための予約です。必ず 1 を設定してください）	R	1
1、0	—	（将来のための予約です。必ず 0 を設定してください）	R	0

注 1.　製品のハードウェアマニュアルを参照してください。

(24) MCTL — マシンの制御

CPU の制御を行うレジスタです。

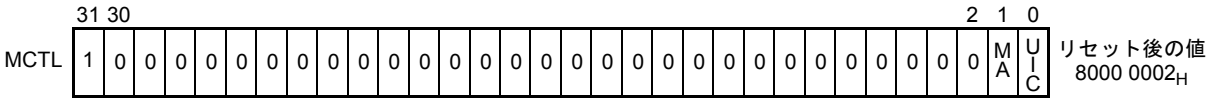


表 3.29　MCTL レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31	—	（将来のための予約です。必ず 1 を設定してください。）	R	1
30 ～ 2	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
1	MA	ハーフワード、ワードのデータ形式を扱うロード命令、ストア命令のミスアライン例外 (MAE) 発生を制御するビットです。 0：MAE 例外発生します。 1：MAE 例外を発生せずアクセスを行います。 詳細は、**「2.6.3　データ・アライメント」**を参照してください。	R/W	1
0	UIC	ユーザ・モードでの割り込み許可禁止操作を制御するビットです。 このビットをセット（1）するとユーザ・モードでの EI/DI 命令の実行が可能になります。	R/W	0

3.3　割り込み機能レジスタ

3.3.1　割り込み機能システム・レジスタ

割り込み機能システム・レジスタへのリード／ライトは、LDSR 命令、STSR 命令により、レジスタ番号と選択識別子からなる、システム・レジスタ番号を指定することで行います。

表 3.30　割り込み機能システム・レジスター覧

レジスタ番号（regID, selID）	名称	機能	アクセス権限
SR7, 1	FPIPR	FPI 例外割り込み優先度設定	SV
SR10, 2	ISPR	受け付け中割り込み優先度	SV
SR11, 2	PMR	割り込み優先度マスク	SV
SR12, 2	ICSR	割り込み制御ステータス	SV
SR13, 2	INTCFG	割り込み機能の設定	SV

(1) FPIPR — FPI 例外割り込み優先度設定

FPI 例外の割り込み優先度を設定するレジスタです。

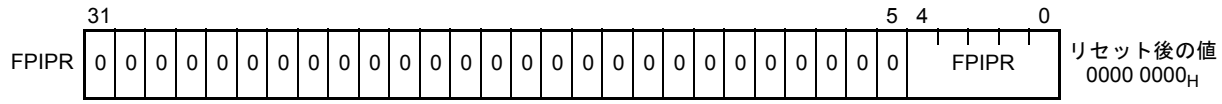


表 3.31　FPIPR レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 5	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
4 ～ 0	FPIPR	浮動小数点演算例外（インプレサイス）（FPI）の割り込み優先度を指定します。0-16 までの値を設定します。17 以上の値を設定禁止です。 FPI 例外は、指定された割り込み優先度として扱われます。同一優先度の割り込みと同時に発生した場合は、FPI 例外が優先されます。 **注　意** 17 以上の値を設定した場合、16 と同様に扱われます。	R/W	0

(2) ISPR — 受け付け中割り込み優先度

CPU で処理中の EIINT*n* の割り込み優先度を、優先度ごとに保持し、多重割り込み時の優先度によるプライオリティ・シーリングを行います。

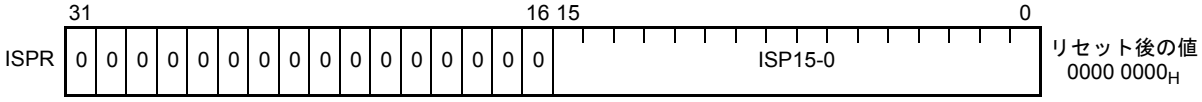


表 3.32　ISPR レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 16	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
15 ～ 0	ISP15-0	ビット位置に対応した優先度[注 1] の割り込み（EIINT*n*）の受け付け状況を示します。 0：ビット位置に対応する割り込み優先度の割り込み要求を受け付けていない 1：ビット位置に対応する割り込み優先度の割り込み要求を CPU コアで処理中 各ビット位置が、次のように各優先度に対応しています。 **ビット / 優先度** 0 / 優先度 0（最高優先度） 1 / 優先度 1 ： 14 / 優先度 14 15 / 優先度 15 割り込み（EIINT*n*）要求を受け付けた場合、受け付けた割り込み（EIINT*n*）要求に対応するビットが自動的にセット（1）されます。また、EIRET 命令実行時に PSW.EP = 0 の場合、ISP15-0 でセット（1）されているビットのうち、最も高優先度（0 が高優先度側）のビットがクリア（0）されます[注 2]。 このビットがセット（1）されている間、その優先度以下の割り込み（EIINT*n*）と FPI 例外[注 3] がマスクされ、例外の受け付け判定時に優先順位判定されず、受け付けられることがありません。詳細は、「4.1.5　割り込みの例外優先度と優先度マスク」を参照してください。 PMR レジスタを利用してソフトウェアによる優先度管理を行う場合は、INTCFG.ISPC ビットの機能によってクリアしてください。	R[注 4]	0

注 1.　詳細は、「4.1.5　割り込みの例外優先度と優先度マスク」を参照してください。
注 2.　INTCFG.ISPC の設定によって、割り込み受け付け、また EIRET 命令による自動更新が行われなくなります。通常は、自動更新を行う方法での利用を推奨します。
注 3.　FPI 例外は、割り込み（EIINT*n*）と同一の優先レベルであるため、割り込みと同様に ISPR の影響を受けます。FPI 例外の優先度は、FPIPR レジスタによって設定されます。
注 4.　INTCFG.ISPC の設定によって、R または R/W となります。通常は R として利用する方法を推奨します。

(3) PMR — 割り込み優先度マスク

指定した割り込み優先度のマスクを行うレジスタです。

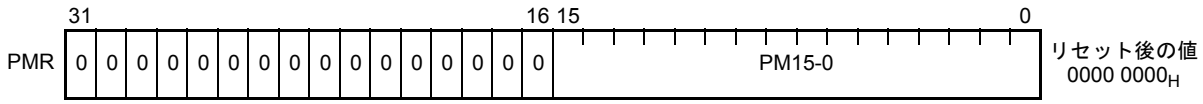


表 3.33　PMR レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 16	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
15 ～ 0	PM15-0	ビット位置に対応した割り込み要求をマスクします。 0：ビット位置に対応する優先度の割り込み処理を許可 1：ビット位置に対応する優先度の割り込み処理を禁止 各ビット位置が、次のように各優先度に対応しています。 ビット / 優先度 0 / 優先度 0（最高優先度） 1 / 優先度 1 : / : 14 / 優先度 14 15 / 優先度 15 および優先度 16（最低優先度） このビットがセット（1）されている間、その優先度の割り込み（EIINT*n*）と FPI 例外[注 1] がマスクされ、例外の受け付け判定時に優先順位判定されず、受け付けられることがありません[注 2]。	R/W	0

注 1.　FPI 例外は、割り込み（EIINT*n*）と同一の優先レベルであるため、割り込みと同様に PMR の影響を受けます。FPI 例外の優先度は、FPIPR レジスタによって設定されます。

注 2.　マスクは低優先側から、連続して 1 を設定してください。例えば $FF00_H$ のような設定は可能ですが、$F0F0_H$ や $00FF_H$ のような設定は禁止します。

(4) ICSR — 割り込み制御ステータス

CPU 内部の割り込み制御の状況を示すレジスタです。

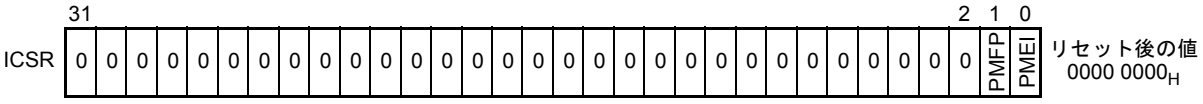


表 3.34　ICSR レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 2	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
1	PMFP	PMR の設定でマスクされている優先度の FPI 例外が存在することを示します。	R	0
0	PMEI	PMR の設定でマスクされている優先度の割り込み（EIINT*n*）が存在することを示します。	R	0

(5) INTCFG — 割り込み機能の設定

CPU 内部の割り込み機能に関する設定を行うレジスタです。

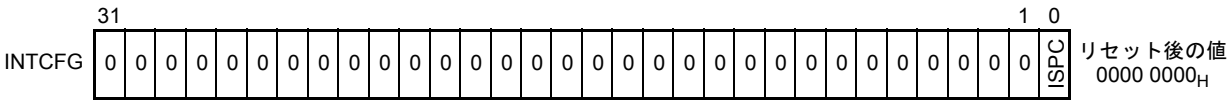


表 3.35　INTCFG レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 1	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
0	ISPC	ISPR レジスタの書き込み変更方法を設定するビットです。 0：ISPR を自動的に更新します。プログラムによる更新（LDSR）を無視します。 1：ISPR を自動的に更新しません。プログラムによる更新（LDSR）が行えます。 このビットがクリア（0）されている場合、ISPR レジスタの各ビットは割り込み（EIINT*n*）の受け付けにより自動的にセット（1）、EIRET 命令の実行によって自動的にクリア（0）が行われます。この状態ではプログラムによる LDSR を使った更新を無視します。 このビットがセット（1）されている場合、ISPR レジスタの各ビットは割り込み（EIINT*n*）の受け付けや、EIRET 命令の実行によっては、更新を行いません。この状態ではプログラムによる LDSR を使った更新が可能となります。 通常 ISPC ビットはクリア（0）した状態で利用してください。ソフトウェアによる割り込み優先度を管理する場合に、このビットをセット（1）し、PMR レジスタによる優先度管理を行ってください。	R/W	0

3.4　FPU 機能レジスタ

3.4.1　浮動小数点レジスタ

FPU は CPU の汎用レジスタ（r0 ～ r31）を使用します。浮動小数点演算専用のレジスタ・ファイルはありません。

- 単精度浮動小数点演算命令：
  32 個の 32 ビット・レジスタを指定できます。これは汎用レジスタの r0 ～ r31 に相当します。

3.4.2　浮動小数点機能システム・レジスタ

FPU では浮動小数点演算制御のために以下のシステム・レジスタが使用できます。浮動小数点機能システム・レジスタへのリード／ライトは、LDSR 命令、STSR 命令により、レジスタ番号と選択識別子からなる、システム・レジスタ番号を指定することで行います。

- FPSR：　例外の制御と監視を行います。また、比較演算の結果を保持し、FPU の動作モードを設定します。条件コード、例外モード、サブノーマル数フラッシュ許可、丸めモード制御、原因、例外許可、保存の各ビットがあります。
- FPEPC：　浮動小数点演算例外が発生した命令のプログラム・カウンタが格納されます。
- FPST：　FPSR レジスタのなかで演算ステータスにかかわるビットの内容を示します。
- FPCC：　FPSR.CC(7:0) ビットと同一の内容を示します。
- FPCFG：　FPSR レジスタのなかで演算設定にかかわるビットの内容を示します。
- FPEC：　FPI 例外の保留状態の確認、取り下げ等の制御を行います。

表 3.36　FPU 機能システム・レジスター覧

レジスタ番号（regID, selID）	名称	機能	アクセス権限
SR6, 0	FPSR	浮動小数点演算の設定／ステータス	CU0 かつ SV
SR7, 0	FPEPC	浮動小数点演算例外プログラム・カウンタ	CU0 かつ SV
SR8, 0	FPST	浮動小数点演算のステータス	CU0
SR9, 0	FPCC	浮動小数点演算の比較結果	CU0
SR10, 0	FPCFG	浮動小数点演算の設定	CU0
SR11, 0	FPEC	浮動小数点演算例外の制御	CU0 かつ SV

(1) FPSR — 浮動小数点演算の設定／ステータス

FPSR レジスタは、浮動小数点演算の実行状態や例外の発生を示します。

例外については、**「6.1.5　浮動小数点演算例外」**を参照してください。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FPSR	CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0	FN	IF	PEM	0	RM		FS	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
原因ビット（XC）						許可ビット（XE）					保存ビット（XP）					リセット後の値 注 1
E	V	Z	O	U	I	V	Z	O	U	I	V	Z	O	U	I	

注 1.　各ビットの説明を参照してください。

表 3.37　FPSR レジスタの内容 (1/2)

ビット	名称	意味	R/W	リセット後の値
31 ～ 24	CC(7:0)	CC（コンディション）ビットです。浮動小数点比較命令の結果がストアされます。CC(7:0) ビットは、比較命令と LDSR 命令以外の影響を受けません。 0：比較結果が偽 1：比較結果が真	R/W	不定
23	FN	近傍へのフラッシュを有効にするビットです。このビットがセット (1) されている場合、丸めモードが RN かつ演算結果がサブノーマル数の時に近傍の値にフラッシュを行います。詳細は**「6.1.11　近傍へのフラッシュ」**を参照してください。	R/W	0
22	IF	オペランド入力のフラッシュ発生を蓄積、表示します。サブノーマル数のフラッシュについては**「6.1.9　サブノーマル数のフラッシュ」**を参照してください。	R/W	0
21	PEM	プレサイス演算例外モードです。PEM ビットが 1 の場合、浮動小数点演算命令の実行により発生した例外は、プレサイス例外として扱います。	R/W	0
20	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
19、18	RM	丸めモード制御ビットです。RM ビットは、FPU がすべての浮動小数点演算命令で使用する丸めモードを規定します。	R/W	00

RM ビット 19	RM ビット 18	ニーモニック	説明
0	0	RN	表現可能な最も近い値に結果を丸めます。2 つの表現可能な値の中間である場合は、最下位ビットが 0 の方に結果を丸めます。
0	1	RZ	0 の方へ結果を丸めます。無限精度の正確な結果を絶対値で越えない、最も近い値になります。
1	0	RP	+ ∞の方へ結果を丸めます。無限精度の正確な結果以上で最も近い値になります。
1	1	RM	- ∞の方へ結果を丸めます。無限精度の正確な結果以下で最も近い値になります。

表 3.37　FPSR レジスタの内容 (2/2)

<table>
<tr><th>ビット</th><th>名称</th><th>意味</th><th>R/W</th><th>リセット後の値</th></tr>
<tr><td>17</td><td>FS</td><td>正規化できない値（サブノーマル数）のフラッシュを許可するビットです。FS ビットがセットされているとき、サブノーマル数のオペランド入力および演算結果は未実装演算例外（E）を起こさず、フラッシュされます。オペランド入力のサブノーマル数は同符号の 0 にフラッシュされます。演算結果のサブノーマル数は丸めモードによって 0 になるか最小正規化値になるかが規定されています。
<table>
<tr><th rowspan="2">サブノーマル数の演算結果</th><th colspan="4">丸めモードとフラッシュ後の値</th></tr>
<tr><th>RN 注 1</th><th>RZ</th><th>RP</th><th>RM</th></tr>
<tr><td>正</td><td>+0</td><td>+0</td><td>$+2^{Emin}$</td><td>+0</td></tr>
<tr><td>負</td><td>–0</td><td>–0</td><td>–0</td><td>-2^{Emin}</td></tr>
</table>
注 1.　丸めモードが RN かつ FPSR.FN がセット（1）されている時にはより精度の高い方向へフラッシュされます。「6.1.11　近傍へのフラッシュ」を参照してください。</td><td>R/W</td><td>1</td></tr>
<tr><td>16</td><td>—</td><td>（将来のための予約です。必ず 0 を設定してください。）</td><td>R</td><td>0</td></tr>
<tr><td>15 ～ 10</td><td>XC
（E, V, Z, O, U, I）</td><td>原因ビットです。詳細は「3.4.2 (1) (a)　原因ビット（XC）」を参照してください。</td><td>R/W</td><td>不定</td></tr>
<tr><td>9 ～ 5</td><td>XE
（V, Z, O, U, I）</td><td>許可ビットです。詳細は「3.4.2 (1) (b)　許可ビット（XE）」を参照してください。</td><td>R/W</td><td>0</td></tr>
<tr><td>4 ～ 0</td><td>XP
（V, Z, O, U, I）</td><td>保存ビットです。詳細は「3.4.2 (1) (c)　保存ビット（XP）」を参照してください。</td><td>R/W</td><td>不定</td></tr>
</table>

(a) 原因ビット（XC）

FPSR レジスタのビット 15 ～ビット 10 は原因ビットで、浮動小数点演算例外の発生とその要因を示します。IEEE754 で定義されている例外が起き、その例外に対応する許可ビットがセット（1）されていた場合、原因ビットをセットし、例外を発生します。1 つの命令で 2 つ以上の例外を検出した場合、それぞれのビットがセット（1）されます。

2 つ以上の例外を検出した場合、いずれかの例外に対応する許可ビットがセット（1）されていれば、例外を発生します。この場合、許可ビットがクリア（0）されている例外を含め、検出したすべての例外の原因ビットがセット（1）されます。

原因ビットは、浮動小数点演算例外を発生した浮動小数点演算命令（TRFSR 命令を除く）によって書き換えられます。E ビットは、ソフトウェアのエミュレートが必要な場合にセット（1）され、それ以外の場合はクリア（0）されます。そのほかのビットは、IEEE754 で定義されている例外が発生したかどうかによりクリア（0）もしくはセット（1）されます。

浮動小数点演算例外が発生した場合、演算結果はストアされず、原因ビットだけが影響を受けます。

LDSR 命令により原因ビットをセット（1）しても、浮動小数点演算例外は発生しません。

(b) 許可ビット（XE）

FPSR レジスタのビット 9 ～ビット 5 は許可ビットで、浮動小数点演算例外の発生を許可します。IEEE754 で定義されている例外が起きたとき、例外に対応する許可ビットがセット（1）されていれば、浮動小数点演算例外が発生します。

未実装演算例外（E）に対応する許可ビットはありません。未実装演算例外（E）は、常に浮動小数点演算例外を発生します。

対応する許可ビットがセット（1）されていない場合、例外は発生せず、IEEE754 によって定義されたデフォルトの結果がストアされます。

(c) 保存ビット（XP）

FPSR レジスタのビット 4 ～ビット 0 は保存ビットで、リセット後、検出した例外を蓄積、表示します。IEEE754 で定義されている例外が発生し、浮動小数点演算例外が発生しない場合に、保存ビットがセット（1）され、そのほかの場合は変化しません。保存ビットは、浮動小数点オペレーションではクリア（0）されません。しかし、LDSR 命令を使用して FPSR レジスタに新たな値を書き込むことで、ソフトウェアによるセット／クリアができます。

未実装演算例外（E）に対応する保存ビットはありません。未実装演算例外（E）は、常に浮動小数点演算例外を発生します。

備　考

例外の種類ごとの各ビットの対応関係については、「図 6.5　FPSR レジスタの原因／許可／保存ビット」を参照してください。

(2) FPEPC — 浮動小数点演算例外プログラム・カウンタ

許可ビットによって許可されている例外が発生した場合、例外が発生した命令のプログラム・カウンタ（PC）が格納されます。

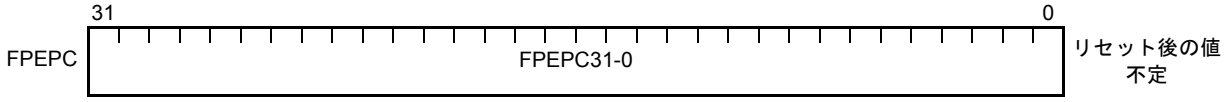


表 3.38　FPEPC レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 1	FPEPC31-1	許可ビットによって、許可された浮動小数点演算例外が発生した場合に、例外を発生させた浮動小数点演算命令の PC が格納されます。	R/W	不定
0	FPEPC0	許可ビットによって、許可された浮動小数点演算例外が発生した場合に、例外を発生させた浮動小数点演算命令の PC が格納されます。常に 0 を設定してください。	R	0

(3) FPST — 浮動小数点演算のステータス

FPSR レジスタのなかで演算ステータスにかかわるビットと同一の内容を示します。

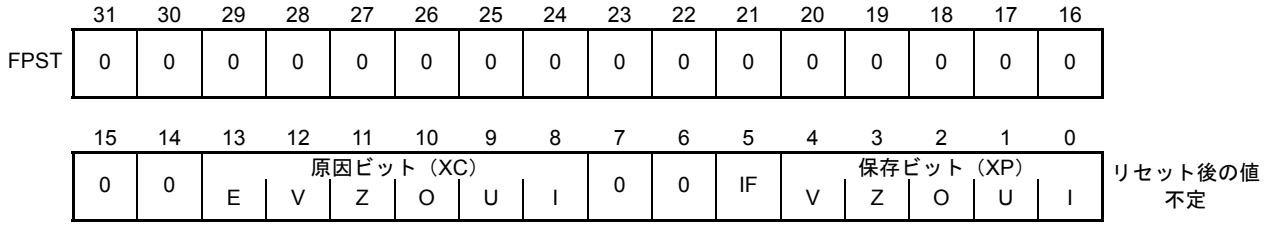


表 3.39　FPST レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 14	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
13 ～ 8	XC （E, V, Z, O, U, I）	原因ビットです。詳細は「**3.4.2 (1) (a)　原因ビット（XC）**」を参照してください。また、このビットへの書き込みは FPSR.XC ビットへ反映されます。	R/W	不定
7、6	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
5	IF	オペランド入力のフラッシュ発生を蓄積、表示します。サブノーマル数のフラッシュについては「**6.1.9　サブノーマル数のフラッシュ**」を参照してください。また、このビットへの書き込みは FPSR.IF ビットへ反映されます。	R/W	0
4 ～ 0	XP （V, Z, O, U, I）	保存ビットです。詳細は「**3.4.2 (1) (c)　保存ビット（XP）**」を参照してください。また、このビットへの書き込みは FPSR.XP ビットへ反映されます。	R/W	不定

(4) FPCC — 浮動小数点演算の比較結果

FPSR.CC[7:0] ビットと同一の内容を示します。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FPCC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0

リセット後の値
不定

表 3.40　FPCC レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 8	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
7 ～ 0	CC(7:0)	CC（コンディション）ビットです。浮動小数点比較命令の結果がストアされます。CC(7:0) ビットは、比較命令と LDSR 命令以外の影響を受けません。また、このビットへの書き込みは FPSR.CC(7:0) ビットへ反映されます。 0：比較結果が偽 1：比較結果が真	R/W	不定

(5) FPCFG — 浮動小数点演算の設定

FPSR レジスタのなかで演算設定にかかわるビットと同一の内容を示します。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FPCFG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	RM		0	0	0	許可ビット（XE）				
											V	Z	O	U	I

リセット後の値
$0000\ 0000_H$

表 3.41　FPCFG レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 10	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
9、8	RM	丸めモード制御ビットです。RM ビットは、FPU がすべての浮動小数点演算命令で使用する丸めモードを規定します。このビットへの書き込みは FPSR.RM ビットへ反映されます。	R/W	0

RM ビット 9	RM ビット 8	ニーモニック	説明
0	0	RN	表現可能な最も近い値に結果を丸めます。2 つの表現可能な値の中間である場合は、最下位ビットが 0 の方に結果を丸めます。
0	1	RZ	0 の方へ結果を丸めます。無限精度の正確な結果を絶対値で越えない、最も近い値になります。
1	0	RP	+ ∞の方へ結果を丸めます。無限精度の正確な結果以上で最も近い値になります。
1	1	RM	- ∞の方へ結果を丸めます。無限精度の正確な結果以下で最も近い値になります。

7 ～ 5	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
4 ～ 0	XE （V, Z, O, U, I）	許可ビットです。詳細は「**3.4.2 (1) (b)　許可ビット（XE）**」を参照してください。 また、このビットへの書き込みは FPSR.XE ビットへ反映されます。	R/W	0

(6) FPEC — 浮動小数点演算例外の制御

浮動小数点演算例外に関する制御を行うレジスタです。

注　意

FPEC レジスタの取り扱いについては、「4.4　例外の管理」を参照してください。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FPEC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FPIVD	リセット後の値 0000 0000$_H$

表 3.42　FPCFG レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 1	—	（将来のための予約です。必ず 0 を設定してください。）	R	0
0	FPIVD 注 1	FPI 例外の通知状況を示します。 このビットがセット（1）されている場合、CPU に対して FPI 例外を通知していて、かつ FPI 例外が受け付けられていない状態を示します。CPU が FPI 例外を受け付けた時点で、このビットは自動的にクリア（0）されます。 また、このビットがセット（1）されている間は、すべての浮動小数点演算命令を無効化します。 このビットがセット（1）されている状態から、LDSR 命令によってクリア（0）することで、FPI 例外の通知を取り下げることができます。FPI 例外の通知を取り下げると、CPU が FPI 例外を受け付けることはありません。 0：FPI 例外非通知状態（FPI 例外の通知を行っていません）。 1：FPI 例外通知状態（FPI 例外の通知を行っています）。	R/W	0

注 1.　FPIVD ビットに対する LDSR 命令による書き込み操作は、クリア（0）のみ可能です。セット（1）は行えません。

3.5　MPU 機能レジスタ

3.5.1　MPU 機能システム・レジスタ

MPU 機能システム・レジスタへのリード／ライトは、LDSR 命令、STSR 命令により、レジスタ番号と選択識別子からなる、システム・レジスタ番号を指定することで行います。

表 3.43　MPU 機能システム・レジスター覧（1/2）

レジスタ番号（regID, selID）	名称	機能	アクセス権限
SR0, 5	MPM	メモリ保護動作モードの設定	SV
SR1, 5	MPRC	MPU リージョン制御	SV
SR4, 5	MPBRGN	MPU ベース・リージョン番号	SV
SR5, 5	MPTRGN	MPU 終端リージョン番号	SV
SR8, 5	MCA	メモリ保護設定チェック・アドレス	SV
SR9, 5	MCS	メモリ保護設定チェック・サイズ	SV
SR10, 5	MCC	メモリ保護設定チェック・コマンド	SV
SR11, 5	MCR	メモリ保護設定チェック結果	SV
SR0, 6	MPLA0	保護領域の下限アドレス	SV
SR1, 6	MPUA0	保護領域の上限アドレス	SV
SR2, 6	MPAT0	保護領域の属性	SV
SR4, 6	MPLA1	保護領域の下限アドレス	SV
SR5, 6	MPUA1	保護領域の上限アドレス	SV
SR6, 6	MPAT1	保護領域の属性	SV
SR8, 6	MPLA2	保護領域の下限アドレス	SV
SR9, 6	MPUA2	保護領域の上限アドレス	SV
SR10, 6	MPAT2	保護領域の属性	SV
SR12, 6	MPLA3	保護領域の下限アドレス	SV
SR13, 6	MPUA3	保護領域の上限アドレス	SV
SR14, 6	MPAT3	保護領域の属性	SV
SR16, 6	MPLA4	保護領域の下限アドレス	SV
SR17, 6	MPUA4	保護領域の上限アドレス	SV
SR18, 6	MPAT4	保護領域の属性	SV
SR20, 6	MPLA5	保護領域の下限アドレス	SV
SR21, 6	MPUA5	保護領域の上限アドレス	SV
SR22, 6	MPAT5	保護領域の属性	SV
SR24, 6	MPLA6	保護領域の下限アドレス	SV
SR25, 6	MPUA6	保護領域の上限アドレス	SV
SR26, 6	MPAT6	保護領域の属性	SV
SR28, 6	MPLA7	保護領域の下限アドレス	SV
SR29, 6	MPUA7	保護領域の上限アドレス	SV
SR30, 6	MPAT7	保護領域の属性	SV
SR0, 7	MPLA8	保護領域の下限アドレス	SV
SR1, 7	MPUA8	保護領域の上限アドレス	SV
SR2, 7	MPAT8	保護領域の属性	SV
SR4, 7	MPLA9	保護領域の下限アドレス	SV
SR5, 7	MPUA9	保護領域の上限アドレス	SV

表 3.43　MPU 機能システム・レジスター覧 (2/2)

レジスタ番号（regID, selID）	名称	機能	アクセス権限
SR6, 7	MPAT9	保護領域の属性	SV
SR8, 7	MPLA10	保護領域の下限アドレス	SV
SR9, 7	MPUA10	保護領域の上限アドレス	SV
SR10, 7	MPAT10	保護領域の属性	SV
SR12, 7	MPLA11	保護領域の下限アドレス	SV
SR13, 7	MPUA11	保護領域の上限アドレス	SV
SR14, 7	MPAT11	保護領域の属性	SV
SR16, 7	MPLA12	保護領域の下限アドレス	SV
SR17, 7	MPUA12	保護領域の上限アドレス	SV
SR18, 7	MPAT12	保護領域の属性	SV
SR20, 7	MPLA13	保護領域の下限アドレス	SV
SR21, 7	MPUA13	保護領域の上限アドレス	SV
SR22, 7	MPAT13	保護領域の属性	SV
SR24, 7	MPLA14	保護領域の下限アドレス	SV
SR25, 7	MPUA14	保護領域の上限アドレス	SV
SR26, 7	MPAT14	保護領域の属性	SV
SR28, 7	MPLA15	保護領域の下限アドレス	SV
SR29, 7	MPUA15	保護領域の上限アドレス	SV
SR30, 7	MPAT15	保護領域の属性	SV

注　搭載されている MPLAn、MPUAn、MPATn レジスタ（n = 0 ～ 15）の数は、製品によって異なります。製品のハードウェアマニュアルを参照してください。

(1) MPM — メモリ保護動作モード

メモリ保護モード・レジスタはメモリ保護機能の基本的な動作状態を決定します。

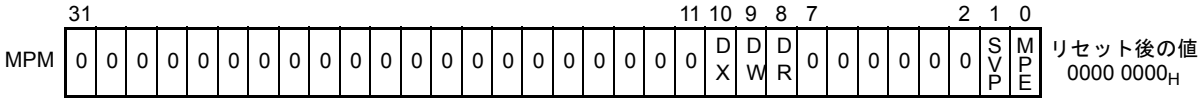


表 3.44　MPM レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 11	—	（将来のための予約です。必ず 0 を設定してください）	R	0
10	DX	保護領域が指定されていないアドレスに対する命令実行時のデフォルトの動作を設定します。本 CPU では 0 固定であり、デフォルト動作は禁止です。必ず 0 を設定してください。 0：保護領域が指定されていないアドレスに対する命令実行の禁止 1：保護領域が指定されていないアドレスに対する命令実行の許可 このビットの設定によって、保護領域境界が交差している場合の動作も変化します。詳細は、「5.1.4　保護領域設定時の注意事項」を参照してください。	R	0
9	DW	保護領域が指定されていないアドレスに対する書き込み操作時のデフォルトの動作を設定します。本 CPU では 0 固定であり、デフォルト動作は禁止です。必ず 0 を設定してください。 0：保護領域が指定されていないアドレスに対する書き込みの禁止 1：保護領域が指定されていないアドレスに対する書き込みの許可 このビットの設定によって、保護領域境界が交差している場合の動作も変化します。詳細は、「5.1.4　保護領域設定時の注意事項」を参照してください。	R	0
8	DR	保護領域が指定されていないアドレスに対する読み出し操作時のデフォルトの動作を設定します。本 CPU では 0 固定であり、デフォルト動作は禁止です。必ず 0 を設定してください。 0：保護領域が指定されていないアドレスに対する読み出しの禁止 1：保護領域が指定されていないアドレスに対する読み出しの許可 このビットの設定によって、保護領域境界が交差している場合の動作も変化します。詳細は、「5.1.4　保護領域設定時の注意事項」を参照してください。	R	0
7 ～ 2	—	（将来のための予約です。必ず 0 を設定してください）	R	0
1	SVP	SV モード（PSW.UM = 0）時に、保護領域ごとに MPAT レジスタの SX、SW、SR にしたがってアクセス制限を行うかどうかを指定します注 1 0：従来とおり、SV モードでは、暗黙的にすべてのアクセスは許可とします。 1：SV モードであっても、SX、SW、SR ビットにしたがってアクセス制限を行います注 2	R/W	0
0	MPE	MPU 機能の有効／無効を指定します。 0：無効 1：有効	R/W	0

注 1.　SVP ビットをセットすると、SV モードであっても保護領域ごとの設定にしたがってアクセス制限が課されるようになります。このため、SVP ビットをセットするプログラム自身がアクセス制限されることがないように、あらかじめ保護領域の設定を行ってください。

注 2.　SV モードでアクセス制限をする場合、設定によっては MDP 例外、MIP 例外処理自身が実行できなくなる場合があります。あらかじめ例外ハンドラ、例外処理に必要なメモリ領域は、アクセスを許可するように注意して設定を行ってください。

(2) MPRC — MPU リージョン制御

メモリ保護機能の特殊操作を行うビットが配置されたレジスタです。

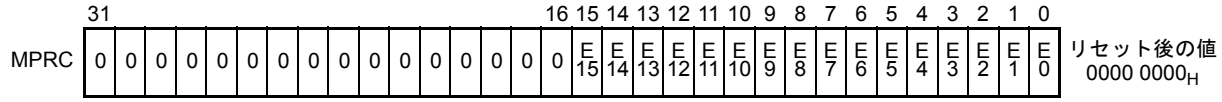


表 3.45　MPRC レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 16	—	（将来のための予約です。必ず 0 を設定してください）	R	0
15 ～ 0	E15-E0	各保護領域の Enable ビットです。En ビットが MPATn.E ビットの写像です（n = 15 ～ 0）。 保護領域数は、製品のハードウェアマニュアルを参照してください。	R/W	0

(3) MPBRGN — MPU ベース・リージョン番号

利用可能な MPU 領域の最小番号を示します。

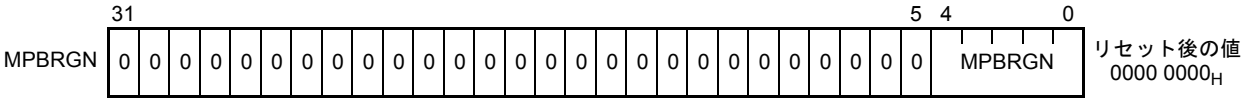


表 3.46　MPBRGN レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 5	—	（将来のための予約です。必ず 0 を設定してください）	R	0
4 ～ 0	MPBRGN	MPU 領域の一番小さい番号を示します。 常に 0 を示します。	R	0

(4) MPTRGN — MPU 終端リージョン番号

利用可能な MPU 領域の最大番号 + 1 を示します。

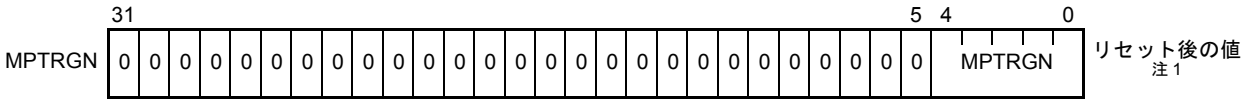


表 3.47　MPTRGN レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 5	—	（将来のための予約です。必ず 0 を設定してください）	R	0
4 ～ 0	MPTRGN	MPU 領域の一番大きい番号 + 1 を示します。 常にハードウェアが搭載する MPU 領域の最大数を示します。	R	注 1

注 1.　製品のハードウェアマニュアルを参照してください。

(5) MCA — メモリ保護設定チェック・アドレス

メモリ保護設定のチェックを行う領域のベース・アドレスを指定します。

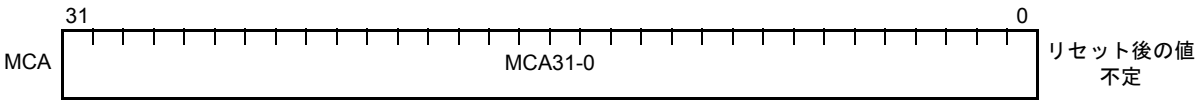


表 3.48　MCA レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 0	MCA31-MCA0	メモリ保護設定のチェックを行う対象のメモリ領域の先頭アドレスをバイト単位で指定します。	R/W	不定

(6) MCS — メモリ保護設定チェック・サイズ

メモリ保護設定のチェックを行う領域のサイズを指定します。

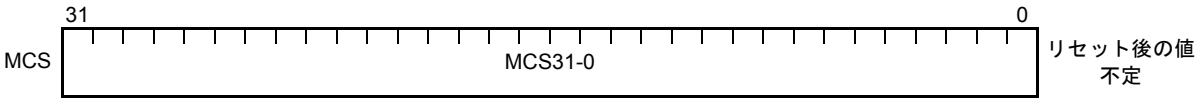


表 3.49　MCS レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 0	MCS31-MCS0	メモリ保護設定のチェックを行う対象のメモリ領域のサイズをバイト単位で指定します。指定されたサイズは符号なしの整数として扱うため、MCA レジスタの値からアドレス値が減少する方向へ領域のチェックを行うことができません。 MCS レジスタには 0000 0000_H を設定しないでください。	R/W	不定

(7) MCC — メモリ保護設定チェック・コマンド

メモリ保護設定のチェックを開始するためのコマンド･レジスタです。

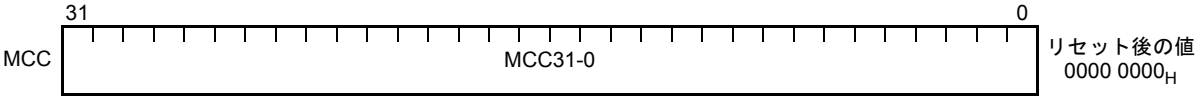


表 3.50　MCC レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 0	MCC31-MCC0	MCC レジスタへの任意の値を書き込むと、メモリ保護設定のチェックが開始されます。事前に MCA/MCS レジスタを設定し、このレジスタへの書き込み操作を行うことで、MCR に結果が格納されます。任意の書き込み値で、チェックを開始するため、r0 をソース･レジスタとして、余分なレジスタを使用することなく、チェックを開始できます。また、チェックは、PSW.UM ビットの状態にかかわらず、各領域設定にしたがった結果を反映します。 MCC レジスタからの読み出し値は、常に 0000 0000_H となります。	R/W	0

(8) MCR — メモリ保護設定チェック結果

メモリ保護設定のチェックの結果を格納するレジスタです。

ビット 31 ～ 9、7、6 には、必ず 0 を設定してください。

注　意

1. チェック対象の領域の指定が 0000 0000$_H$ または 7FFF FFFF$_H$ をまたぐ場合、領域指定が誤っていると判断し、MCR.OV ビットがセット（1）されます。このため、チェック結果を参照する場合には、必ず MCR.OV ビットを確認し、結果が不正でないこと（OV = 0）を確認してから、その他のチェック結果を利用してください。

2. デフォルト設定（MPM.DX, DW, DR）をセット（1）している場合、正しい結果が得られない場合があります。デフォルトを許可にする場合、メモリ保護設定チェック機能は利用しないでください。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
MCR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	OV	0	0	SXE	SWE	SRE	UXE	UWE	URE	リセット後の値 不定

表 3.51　MCR レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 9	—	（将来のための予約です。必ず 0 を設定してください）	R	0
8	OV	指定された領域が 0000 0000$_H$ または、7FFF FFFF$_H$ をまたがる場合に、1 が格納されます。それ以外の場合は、0 が格納されます。	R/W	不定
7、6	—	（将来のための予約です。必ず 0 を設定してください）	R	0
5	SXE	指定された領域が、いずれか 1 つの保護領域の中に収まっており、かつその保護領域がスーパバイザ実行許可であった場合に、1 が格納されます。それ以外の場合は、0 が格納されます。	R/W	不定
4	SWE	指定された領域が、いずれか 1 つの保護領域の中に収まっており、かつその保護領域がスーパバイザ・ライト許可であった場合に、1 が格納されます。それ以外の場合は、0 が格納されます。	R/W	不定
3	SRE	指定された領域が、いずれか 1 つの保護領域の中に収まっており、かつその保護領域がスーパバイザ・リード許可であった場合に、1 が格納されます。それ以外の場合は、0 が格納されます。	R/W	不定
2	UXE	指定された領域が、いずれか 1 つの保護領域の中に収まっており、かつその保護領域がユーザ・モード実行許可であった場合に、1 が格納されます。それ以外の場合は、0 が格納されます。	R/W	不定
1	UWE	指定された領域が、いずれか 1 つの保護領域の中に収まっており、かつその保護領域がユーザ・モード・ライト許可であった場合に、1 が格納されます。それ以外の場合は 0 が格納されます。	R/W	不定
0	URE	指定された領域が、いずれか 1 つの保護領域の中に収まっており、かつその保護領域がユーザ・モード・リード許可であった場合に、1 が格納されます。それ以外の場合は、0 が格納されます。	R/W	不定

(9) MPLAn — 保護領域の下限アドレス

領域 n の下限アドレスを示すレジスタです（n = 0 ～ 15）。保護領域数 n は製品によって異なります。製品のハードウェアマニュアルを参照してください。

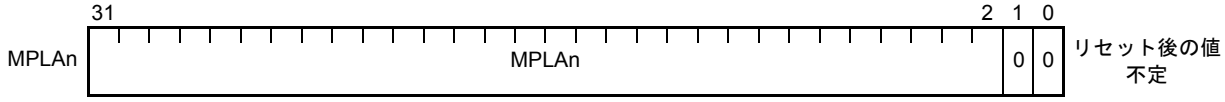


表 3.52　MPLAn レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 2	MPLA31-2	領域 n の下限アドレスを示します。 MPLAn.MPLA1 ～ 0 は暗黙的に 0 を使用します。	R/W	不定
1、0	—	（将来のための予約です。必ず 0 を設定してください）	R	0

(10) MPUAn — 保護領域の上限アドレス

領域 n の上限アドレスを示すレジスタです（n = 0 ～ 15）。保護領域数 n は製品によって異なります。製品のハードウェアマニュアルを参照してください。

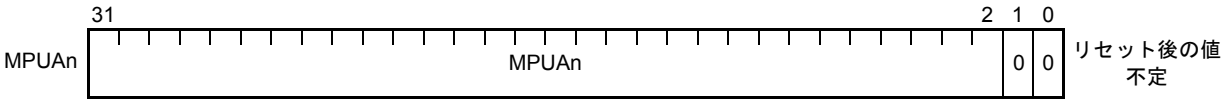


表 3.53　MPUAn レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 2	MPUA31-2	領域 n の上限アドレスを示します。 MPUAn.MPUA1 ～ 0 は暗黙的に 1 を使用します。	R/W	不定
1、0	—	（将来のための予約です。必ず 0 を設定してください）	R	0

(11) MPATn — 保護領域の属性

領域 n の属性を示すレジスタです（n = 0 ～ 15）。保護領域数 n は製品によって異なります。製品のハードウェアマニュアルを参照してください。

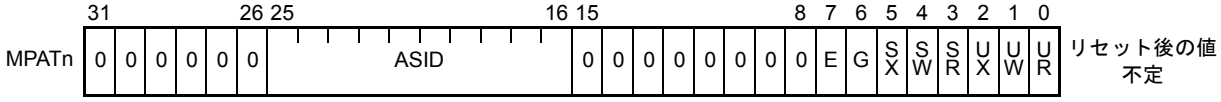


表 3.54　MPATn レジスタの内容

ビット	名称	意味	R/W	リセット後の値
31 ～ 26	—	（将来のための予約です。必ず 0 を設定してください）	R	0
25 ～ 16	ASID	領域一致の条件として利用する ASID の値を示します。	R/W	不定
15 ～ 8	—	（将来のための予約です。必ず 0 を設定してください）	R	0
7	E	領域 *n* の有効／無効を示します。 0：領域 *n* を無効とする 1：領域 *n* を有効とする	R/W	0
6	G	0：ASID 一致を条件とする 1：ASID 一致を条件としない このビットが 0 の場合は、MPATn.ASID = ASID.ASID であることが領域一致の条件となります。このビットが 1 の場合は、MPATn.ASID と ASID.ASID の値を領域一致の条件としません。	R/W	不定
5	SX	スーパバイザ・モードでの実行権を示します[注 1]。 0：実行禁止 1：実行許可	R/W	不定
4	SW	スーパバイザ・モードでの書き込み許可を示します[注 1]。 0：書き込み禁止 1：書き込み許可	R/W	不定
3	SR	スーパバイザ・モードでの読み出し許可を示します[注 1]。 0：読み出し禁止 1：読み出し許可	R/W	不定
2	UX	ユーザ・モードでの実行権を示します。 0：実行禁止 1：実行許可	R/W	不定
1	UW	ユーザ・モードでの書き込み許可を示します。 0：書き込み禁止 1：書き込み許可	R/W	不定
0	UR	ユーザ・モードでの読み出し許可を示します。 0：読み出し禁止 1：読み出し許可	R/W	不定

注 1.　SV モードでアクセス制限をする場合、設定によっては MDP 例外、MIP 例外処理自身が実行できなくなる場合があります。あらかじめ例外ハンドラ、例外処理に必要なメモリ領域は、アクセスを許可するように注意して設定を行ってください。

第 4 章　例外／割り込み

例外とは、特定の要因によって実行中のプログラムから別のプログラムへの強制的な分岐動作を発生する例外的事象です。

それぞれの例外ごとの分岐先のプログラムを“ 例外ハンドラ ”と呼びます。

注　意

本 CPU では、割り込みを例外の一種として扱います。

4.1　例外の仕組み

ここでは、各例外の性質を特徴付ける要素について説明し、例外の仕組みを示します。

4.1.1　例外要因一覧

表 4.1 例外要因一覧

例外	名称	発生元	実行形態[注1]	退避リソース	復帰／回復	例外要因コード[注5]	優先順位[注2]		受付条件（PSW）		更新（PSW）				
							優先レベル	優先度	ID	NP	UM	ID	NP	EP	EBV
RESET	リセット	リセット入力[注3]	中断型	—	—	なし	1	—	x	x	0	1	0	0	0
FENMI	FENMI 割り込み	割り込みコントローラ[注3]	中断型	FE	不可	$E0_H$	3	1	x	x	0	1	1	0	s
SYSERR	システム・エラー	システム・エラー入力[注3]	中断型	FE	不可	10_H-$1F_H$[注3]	3	2	x	x	0	1	1	1	s
FEINT	FEINT 割り込み	割り込みコントローラ[注3]	中断型	FE	可能	$F0_H$	3	3	x	0	0	1	1	0	s
FPI	FPU 例外（インプレサイス）	FPU 命令の実行	中断型	EI	復帰可能回復不可	72_H	4	[注4]	0	0	0	1	s	1	s
EIINT0-511	ユーザ割り込み	割り込みコントローラ[注3]	中断型	EI	可能	1000_H-$11FF_H$[注6]	4	[注4]	0	0	0	1	s	0	s
MIP	メモリ保護例外 (実行権)	メモリ保護違反	再実行型	FE	可能	90_H	10	1	x	x	0	1	1	1	s
SYSERR	システム・エラー	命令フェッチ時のエラー入力[注3]	再実行型	FE	不可	10_H-$1F_H$[注3]	10	3	x	x	0	1	1	1	s
RIE	予約命令例外	予約命令の実行	再実行型	FE	可能	60_H	10	4	x	x	0	1	1	1	s
UCPOP	コプロセッサ使用不可例外	コプロセッサ命令の実行／アクセス権限違反	再実行型	FE	可能	80_H-82_H[注9]	10	5	x	x	0	1	1	1	s
PIE	特権命令例外	特権命令の実行／アクセス権限違反	再実行型	FE	可能	$A0_H$	10	6	x	x	0	1	1	1	s
MAE	ミスアライン例外	ミスアライン・アクセスの発生	再実行型	FE	可能	$C0_H$	11	[注7]	x	x	0	1	1	1	s
MDP	メモリ保護例外 (アクセス権)	メモリ保護違反	再実行型	FE	可能	91_H	11	[注7]	x	x	0	1	1	1	s
FPP	浮動小数点演算例外	FPU 命令の実行	再実行型	EI	可能	71_H	11	[注7]	x	x	0	1	s	1	s
SYSCALL	システム・コール	SYSCALL 命令の実行	完了型	EI	可能	8000_H-$80FF_H$	12	[注8]	x	x	0	1	s	1	s
FETRAP	FE レベル・トラップ	FETRAP 命令の実行	完了型	FE	可能	31_H-$3F_H$	12	[注8]	x	x	0	1	1	1	s
TRAP0	EI レベル・トラップ 0	TRAP 命令の実行	完了型	EI	可能	40_H-$4F_H$	12	[注8]	x	x	0	1	s	1	s
TRAP1	EI レベル・トラップ 1	TRAP 命令の実行	完了型	EI	可能	50_H-$5F_H$	12	[注8]	x	x	0	1	s	1	s

備考 s: 保持、x: 受け付け条件でない

注 1. 詳細は、**「4.1.3 例外の実行形態」**を参照してください。

注 2. 例外の受け付け優先順位は、優先レベル→優先度の順で、より小さい値を高優先として判断します。詳細は、**「4.1.4 例外の受け付け条件と優先順位」**を参照してください。

注 3. 詳細は、製品のハードウェアマニュアルを参照してください。

注 4. EIINT0-511 と FPI の優先度は、レジスタ設定によって変化します。詳細は、**「4.1.5 割り込みの例外優先度と優先度マスク」**を参照してください。

注 5. 例外要因コードの下位 16 ビットを示します。例外要因コードの上位 16 ビットは、例外ごとに定義された詳細コードが入ります。各機能の説明で特に記載がない場合は、0000_H です。

注 6. チャネルに応じて、1000_H-$11FF_H$（チャネル 0-511）が選択されます。

注 7. 命令のオペレーション順序に依存します。

注 8. 命令実行により発生するため排他的に発生します。同一優先レベル内の優先度はありません。

注 9. コプロセッサ使用権（CU0-2）にそれぞれ 80_H-82_H が対応します。

4.1.2　例外要因の概要

本 CPU の例外要因の概要について説明します。

(1) RESET

リセット入力で発生します。詳細は**「第 8 章　リセット」**を参照してください。

(2) FENMI, FEINT, EIINT

割り込みコントローラからの入力によって、任意のプログラムを起動する割り込みです。割り込み機能に関する詳細は、**「3.3　割り込み機能レジスタ」**、および製品に搭載された割り込みコントローラの仕様を参照してください。

(3) SYSERR

システム・エラー例外です。ハードウェア仕様によって定められたエラーの検出時に発生します。命令フェッチ・アクセスの際に生じたエラーは再実行型の SYSERR 例外として通知されます。それ以外のエラーは中断型の SYSERR 例外として通知されます。

注　意

SYSERR 例外の要因は、ハードウェア機能によって定められます。詳細は、製品のハードウェアマニュアルを参照してください。

(4) FPI, FPP

浮動小数点演算命令を実行した際に、演算の過程で発生する例外です。詳細は、**「6.1　浮動小数点演算」**を参照してください。

(5) MIP, MDP

MPU 機能で違反を検出した場合に発生する例外です。例外の検出は、メモリ・アクセスを行う命令がアドレス計算を行う際に行われます。詳細は、**「5.1　メモリ保護機能（MPU）」**を参照してください。

(6) RIE

予約命令例外です。動作の定義された命令以外のオペコードを実行しようとした場合に発生する例外です。動作は定義された RIE 命令と同一です。詳細は**「7.1.3　予約命令」**、および**「第 7 章　命令」**の RIE 命令を参照してください。

(7) PIE

特権命令例外です。実行しようとした命令が要求する特権を所持していない場合などに発生する例外です。詳細は、**「2.1.3　CPU 動作モードと特権」**、**「2.2　命令の実行」**、**「2.5.3(1) LDSR、STSR」**を参照してください。

(8) UCPOP

コプロセッサが使用不可能な場合に、コプロセッサ命令を実行しようとした場合に発生する例外です。詳細は**「2.4　コプロセッサ」**を参照してください。

(9) MAE

アドレス計算の結果がミスアラインである場合に発生する例外です。詳細は**「2.6.3　データ・アライメント」**を参照してください。

(10) TRAP, FETRAP, SYSCALL

命令の実行結果によって発生する例外です。詳細は**「第 7 章　命令」**を参照してください。

4.1.3　例外の実行形態

本 CPU では、その実行形態によって、例外を次の 3 つの種類に分類します。

- 中断型例外
- 再実行型例外
- 完了型例外

(1) 中断型例外

命令のオペレーションを実行する前に、その命令を中断して受け付けられる例外です。割り込みや、インプレサイス（不正確な）例外などが中断型例外にあたります。

割り込みの場合は、現在実行中の命令の実行結果によって発生するわけではなく、その命令と無関係に発生します。割り込み発生時には他の例外と異なり、PSW.EP ビットがクリア（0）されます。このため、復帰命令実行時に、外部の割り込みコントローラに対して、例外処理ルーチンの終了を通知します。割り込みからの復帰命令実行時には必ず PSW.EP ビットがクリア（0）されている状態で実行してください。

注　意

PSW.EP ビットは、割り込み（INT0-INT511, FEINT, FENMI）の受け付け時にのみクリア（0）します。その他の例外では PSW.EP ビットをセット（1）します。

PSW.EP ビットがセット（1）された状態で、割り込みによって起きた例外処理ルーチンからからの復帰命令を実行すると、外部の割り込みコントローラ上のリソースが解放されず、誤動作を引き起こす可能性があります。

インプレサイス例外は、中断する命令以前の命令の実行の結果が不正である場合に、遅延して発生する不正確な例外です。インプレサイス例外は、その原因となった命令の後続の命令が既に実行を完了している場合があり、例外の原因となった時点の CPU の状態が保存されていないため、例外の処理後に元の処理を回復して再実行することができません。

中断型例外の復帰 PC は、中断された命令の PC（CurrentPC）となります。

(2) 再実行型例外

命令のオペレーションの実行中に発生し、その命令の実行を完了せずに受け付ける例外で、命令実行時に命令を完了せず、また後続の命令を実行することもなく、正確に例外を受け付けるため、プレサイス（正確な）例外とも呼ばれます。この例外の発生によって、汎用レジスタやシステム・レジスタの更新は行われず、また例外からの復帰 PC も例外を発生した命令自身を指しているため、例外が起きる前の状態から再実行することが可能です。

再実行型例外の復帰 PC は、例外を引き起こした命令の PC（CurrentPC）となります。

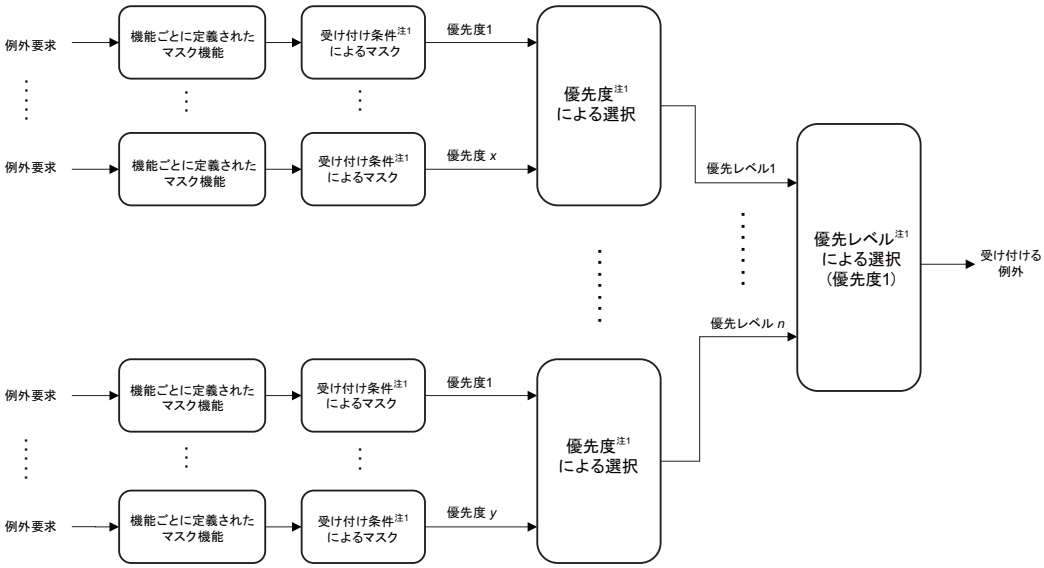
(3) 完了型例外

命令のオペレーションの実行の結果として発生し、その命令の実行を完了後に受け付ける例外です。ソフトウェア例外などが完了型例外にあたります。完了型例外は、命令の正常実行の結果として発生するため、復帰した場合に再開する命令は、完了型例外を起こした命令の次の命令となります。例外処理後元の処理を正常に継続することが可能です。

完了型例外の復帰 PC は、次の命令の PC（NextPC）となります。

4.1.4　例外の受け付け条件と優先順位

CPU は、例外の受け付け条件と優先順位にしたがってあるタイミングにおいてはひとつの例外のみ受け付けます。このとき受け付ける例外は次の**図 4.1** のように受け付け条件と優先順位にしたがって決定します。



注 1.　**表 4.1** を参照してください。

図 4.1　例外の受け付け条件と優先順位

表 4.1 において、受け付け条件の欄に “0” とある例外は、該当ビットが “0” であるときに例外の受け付けが可能となります。このような例外では、該当ビットが “1” であると例外の受け付けが保留されますが、該当ビットが “0” に変化して受け付け条件が成立すると、例外の受け付けが可能状態となります。特に指定のない場合は、そのビットは受け付け条件ではありません。複数のビットが条件とされている場合は、すべての条件を同時に満たす必要があります。

同時に受け付け条件を満たしている例外が複数ある場合は、優先順位にしたがって、受け付ける例外を 1 つ選択します。優先順位の判定は多段的に行い、優先レベル、優先度の順に数字が小さい方が高優先となります。

中断型の例外は受け付けられなかった場合には保留されます。リセットと同時に発生した場合は、保留されません。詳細は**「4.2.1　特殊な動作」**を参照してください。

受け付け条件、優先レベル、優先度に関しては、**「表 4.1　例外要因一覧」**を参照してください。

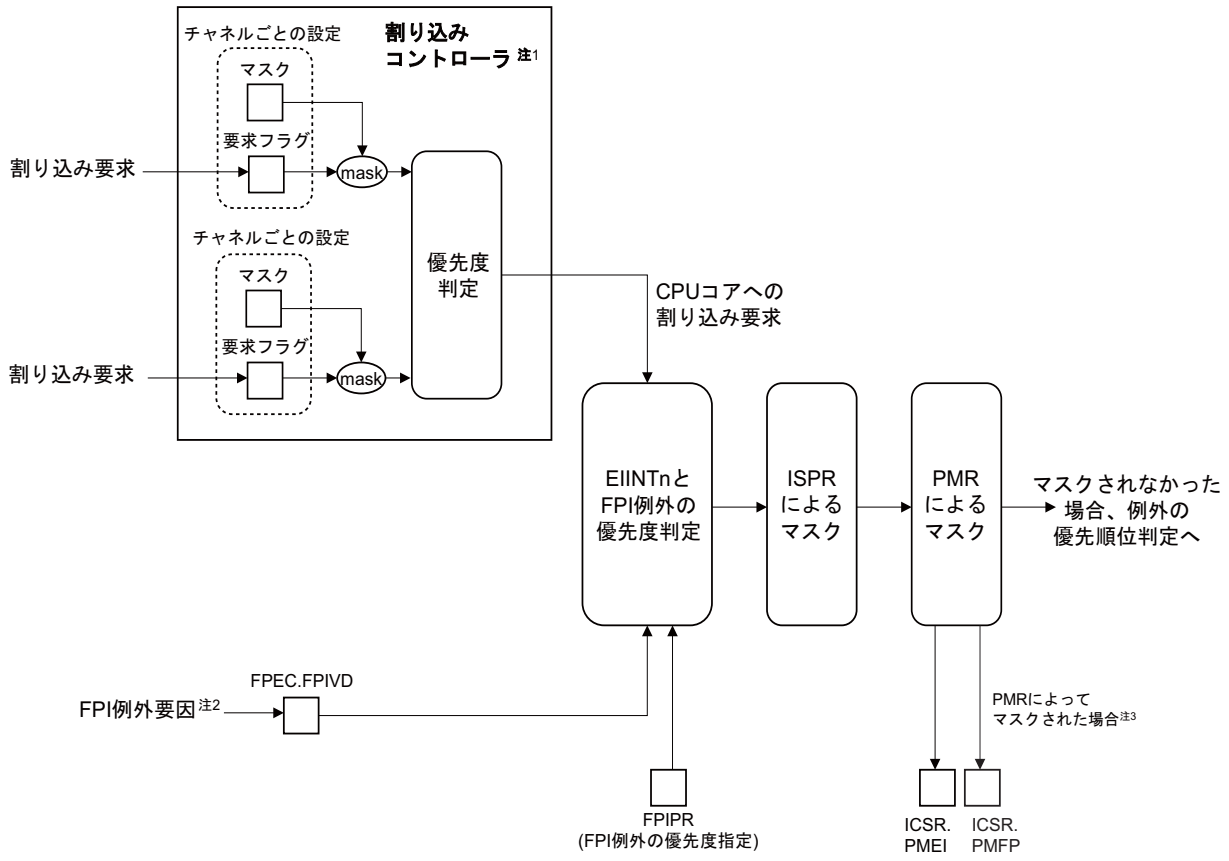
4.1.5　割り込みの例外優先度と優先度マスク

割り込み（EIINTn）とインプレサイス浮動小数点例外（FPI）はレジスタ設定によって、例外優先度や優先度ごとにマスクを行うことが可能です。この機能によって、より柔軟なソフトウェア構造をとりつつ、またソフトウェアによるメンテナンスなしに割り込みシーリングが行えます。

注　意

ISPR レジスタ、PMR レジスタ、ICSR レジスタは、V850E2 製品においては、割り込みコントローラの機能として定義されていました。本 CPU では、CPU の機能として定義していますが、基本的な機能は同等です。ただし、一部機能が異なるので注意してください。

図 4.2 に割り込みの例外優先度と優先度マスクの機能についての機能概要を示します。



注 1.　割り込みコントローラについては、製品のハードウェアマニュアルを参照してください。

注 2.　FPI 例外要因は、FPU 機能によって例外許可されており、かつインプレサイス例外が指定されている場合に発生します。詳細は、**「6.1.5　浮動小数点演算例外」**および**「6.1.7　プレサイス例外とインプレサイス例外」**を参照してください。

注 3.　ICSR レジスタの PMEI ビット、PMFP ビットに表示されるのは、PMR によってマスクされた EIINT*n*、または FPI です。PMR より先に ISPR または各機能のマスク機能でマスクされた場合には、PMEI ビット、PMFP ビットには影響しません。

図 4.2　　割り込みの例外優先度と優先度マスク

(1) 割り込み優先度

割り込み（EIINT*n*）とインプレサイス浮動小数点例外（FPI）はレジスタ設定によって、例外優先度を変更することができます。EIINT*n* と FPI は同一の優先レベルで定義されており、例外優先度の変更によって EIINTn と FPI の間の優先関係を制御することが可能です。

EIINT*n* と FPI の優先度の関係は、次の**図 4.3** のようになっています。同一優先度では FPI が優先となります。FPI の優先度は FPIPR レジスタによって設定することができ、いずれか一つの優先度が選択されます。

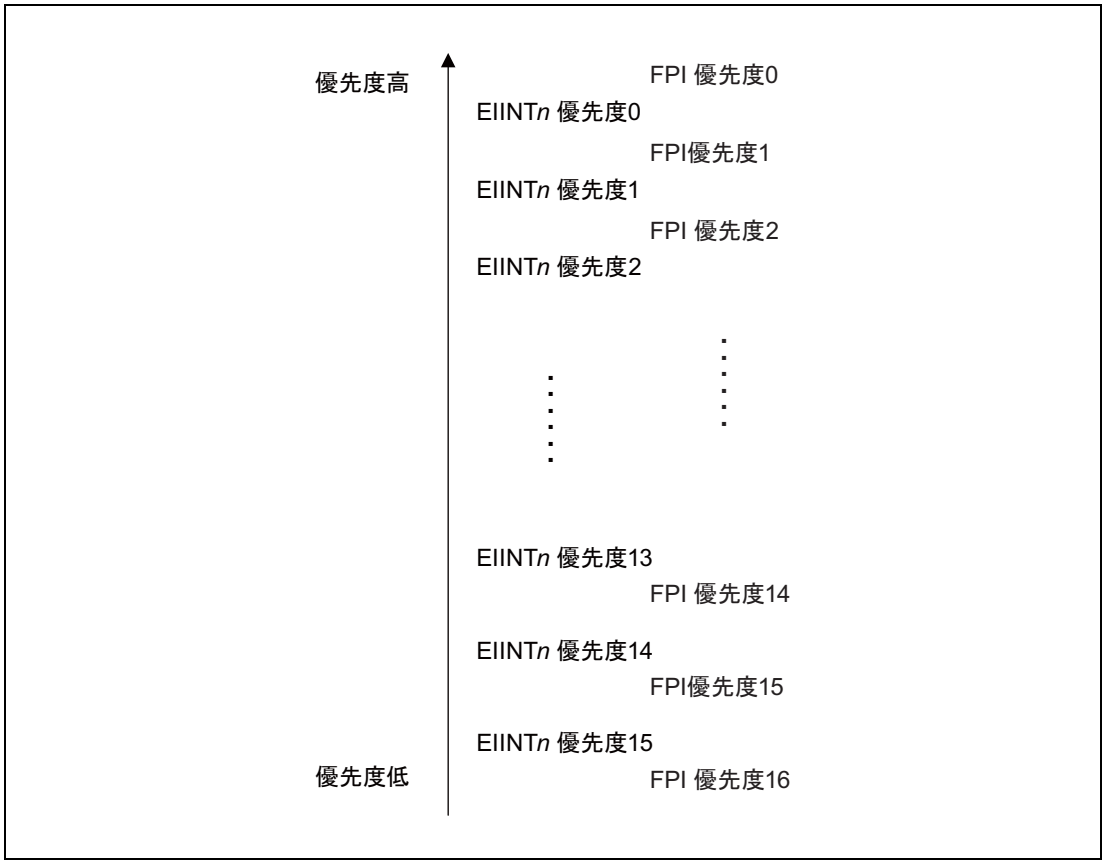


図 4.3　EIINT*n* と FPI の優先関係

(2) 割り込み優先度マスク

また、EIINT*n* と FPI は ISPR レジスタ、PMR レジスタによって優先度ごとにマスクされる場合があります。それぞれのレジスタは次のように使うことを想定しています。

ISPR レジスタは、ハードウェアが割り込みを受け付けた場合に優先度に対応するビットが自動的にセット（1）され、その優先度以下の割り込みをマスクします。また、割り込みに対応する EIRET 命令が実行された場合、自動的に ISPR レジスタのビットをクリア（0）することで、マスクを解除します。

このように割り込みシーリングを自動的に行うことで、ソフトウェアによる管理を必要とせず、容易に多重割込処理を行うことが可能です。

PMR レジスタを用いることで、ソフトウェアにより特定の割り込み優先度をマスクすることが可能です。プログラム中で一時的に割り込みシーリングのレベルを引き上げたい場合などに利用してください。ISPR レジスタのマスク設定と PMR のマスク設定は重複し、いずれかでマスクされていればその割り込みはマスクされています。通常は、ISPR のシーリング値から更にシーリング値を引き上げる場合に PMR レジスタを利用してください。

INTCFG レジスタの機能を利用して、ISPR レジスタが割り込みの受け付け／復帰によって行う自動更新を無効にすることができます。ISPR レジスタの機能を利用せず、ソフトウェアで割り込みシーリングを行う場合は、INTCFG レジスタの ISPC ビットをセット（1）したあと、ISPR レジスタをクリアし、以降は PMR レジスタを利用してシーリング値をソフトウェアで管理してください。

また、PMR を利用している状況で PMR によってマスクされている割り込みの存在は、ICSR レジスタで確認が可能です。

(3) EIINT*n* と FPI の動作の違い

EIINT*n* と FPI は例外の受け付けまでの挙動は同様に取り扱われますが、受け付け後の動作は一部異なります。

FPI 例外の受け付けでは、ISPR レジスタが更新されないため、FPI 例外処理中に EI 命令などによって、PSW.ID ビットをクリア（0）し、割り込み禁止状態を解除した場合、FPI 例外よりも低優先の割り込みが多重で発生する場合があります。

FPI 例外の優先度は、一般的に FPU を利用するプログラムよりも高い優先度を設定して利用します。このため、FPI 例外中により低優先の割り込みを受け付けてしまうと、FPI 例外処理が終わる前に更に FPI 例外が発生してしまう可能性があります。このため、FPI 例外中で割り込み禁止を解除する前には、PMR などを利用して適切に割り込み優先度マスクを行う必要があります。

4.1.6　復帰と回復

例外処理を行った場合、例外の受け付けによって中断した元のプログラムに対して影響を与える可能性があります。この影響は「復帰」と「回復」という 2 つの観点で表現されます。

- 復帰：元のプログラムの中断した位置から再実行再開が可能／不可能であることを示します。
- 回復：元のプログラムを中断した時点のプロセッサ状態（汎用レジスタ、システム・レジスタなどのプロセッサ資源の状態）への回復が可能／不可能であることを示します。

復帰不可、回復不可（**表 4.1** では、「不可」と記載）の例外は、復帰 PC などを破壊してしまう場合があり、その例外から復帰命令を用いて、元の処理に戻ることができません。例外の起きる状況を選べない場合に、復帰不可、回復不可となります。

回復不可の例外は、復帰して元のプログラム・フローには戻れますが、その時点では例外発生前の状態を再現できないため、以後のプログラム動作の継続に注意が必要です。

4.1.7　コンテキスト退避

例外発生時に現在のプログラム・シーケンスを保存するために、次のリソースを機能定義にしたがって適切に退避します。

- プログラム・カウンタ（PC）
- プログラム・ステータス・ワード（PSW）
- 例外要因コード（EIIC, FEIC）
- 作業用システム・レジスタ（EIWR, FEWR）

退避先として利用するリソースは、例外の種類、によって決定されます。ここでは、退避リソースの決定について説明を行います。

(1) コンテキスト退避

受け付け条件が定められている一部の例外は、ほかの例外受け付け時に自動的にセットされる保留ビット（PSW.ID, NP ビット）によって、例外処理の開始時点では受け付けられない状態となっています。

同一レベルの例外を再度受け付け可能な多重例外処理を可能にするためには、これらの復帰レジスタ、およびそれぞれの例外要因ごとに定められた特定の情報をスタックなどへ退避しておく必要があります。これらの退避が必要な情報を「コンテキスト」と呼びます。

原則として、コンテキストの退避前に同一のレベルへの例外を発生させないように注意する必要があります。

コンテキスト退避のための作業を行う際に利用できる作業用システム・レジスタと、多重例外処理を可能にするために最低限、退避が必要なシステム・レジスタを、基本コンテキスト・レジスタと呼びます。基本コンテキスト・レジスタはレベルごとに用意されています。

表 4.2　基本コンテキスト・レジスタ

例外レベル	基本コンテキスト・レジスタ
EI レベル	EIPC, EIPSW, EIIC, EIWR
FE レベル	FEPC, FEPSW, FEIC, FEWR

4.2　例外受け付け時の動作

命令の実行時点で通知されている例外を、優先順位にしたがって、ひとつひとつ例外受け付けを行うかどうかを確認していきます。例外ごとの受け付け動作の手順は次のとおりです。

<1> 受け付け条件を満たしているか、また優先順位にしたがって受け付けを行うかどうかを確認する。

<2> 現在の PSW の値にしたがって、例外ハンドラ・アドレスを計算する注 1。

<3> FE レベル例外の場合、それぞれ次の処理を行う。

- PC を FEPC に退避
- PSW を FEPSW に退避
- FEIC に例外要因コードを格納
- PSW を更新する注 2
- PC に <2> で計算した例外ハンドラ・アドレスを格納し、例外ハンドラへ制御を移す。

<4> EI レベル例外の場合、それぞれ次の処理を行う。

- PC を EIPC に退避
- PSW を EIPSW に退避
- EIIC に例外要因コードを格納
- PSW を更新する注 2
- PC に <2> で計算した例外ハンドラ・アドレスを格納し、例外ハンドラへ制御を移す。

注 1. 詳細は、**「4.5　例外ハンドラ・アドレス」**を参照してください。

注 2. 更新する値は、**「表 4.1　例外要因一覧」**を参照してください。

<1> から <4> の手順を次の図で示します。

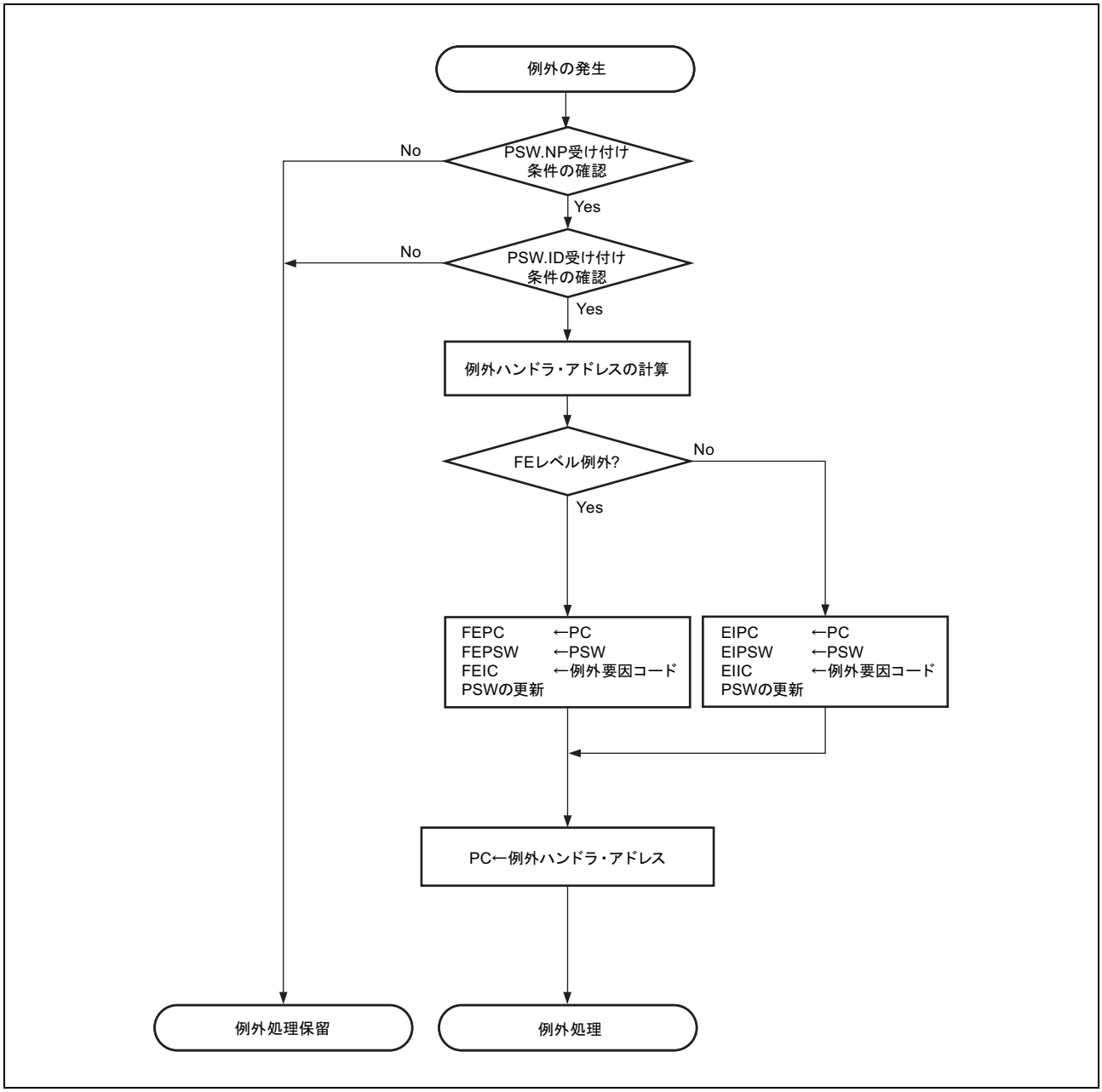


図 4.4　　例外受け付け時の動作

4.2.1　特殊な動作

(1) PSW レジスタの EP ビット

割り込みを受け付けた場合、PSW.EP ビットをクリア（0）します。割り込み以外の例外を受け付けた場合、PSW.EP ビットをセット（1）します。

EP ビットの状態によって、EIRET, FERET 命令の実行時の動作が変化します。EP ビットがクリア（0）されている場合、ISPR.ISP15-0 でセット（1）されているビットのうち、最も高優先度（0 が高優先度側）のビットがクリア (0) します。また、外部の割り込みコントローラに対して、例外処理ルーチンの終了を通知します。これは、割り込みの受け付け／割り込みからの復帰によって、割り込みコントローラ上の要求フラグなどのリソースを適切に制御するために必要な機能です。

割り込みからの復帰する際には、必ず EP ビットをクリア（0）した状態で復帰命令を実行してください。

(2) コプロセッサ使用不可例外

コプロセッサ使用不可例外は、製品ごとの機能仕様によって、PSW レジスタの CU ビットの状態に対応して例外の発生するオペコードが変化します。

コプロセッサ命令と定義されたオペコードに対して、製品上で搭載されていない、あるいは、動作状態によって使用が許可されていない場合に、これらのコプロセッサ命令を実行しようとした場合、またはコプロセッサのシステム・レジスタに LDSR 命令、STSR 命令でアクセスしようとした場合、ただちにコプロセッサ使用不可例外（UCPOP）が発生します。

詳細は、**「2.4.3　コプロセッサ使用不可例外」**を参照してください。

(3) 予約命令例外

将来の機能拡張のために予約され、命令が定義されていないオペコードに対して実行を行う際、予約命令例外（RIE）が発生します。

ただし、個々のオペコードごとに、次の 2 種類の動作のいずれかを行うことをハードウェア仕様により定義することがあります。

- 予約命令例外が発生する
- いずれかの定義された命令として動作する

常に予約命令例外が発生するオペコードが RIE 命令として定義されています。

(4) リセット

リセットも、例外と同様の動作を行いますが、EI レベル例外、FE レベル例外のいずれにも属しません。動作としては、受け付け条件のない例外と同様ですが、各レジスタはリセット後の値に変化します。また、リセットからの復帰等も行えません。

また、CPU 初期化と同時に発生していた例外は、すべて取り消され、CPU 初期化後にも受け付けられることはありません。

詳細は**「第 8 章　リセット」**を参照してください。

4.3　例外からの復帰

例外処理からの復帰には、それぞれの例外レベルに対応した復帰命令（EIRET, FERET）の実行によって行います。

スタックなどにコンテキストを退避している場合は、復帰命令の実行前にコンテキストの復帰を必ず行ってください。また、回復不可能な例外からの復帰時には、元のプログラムが例外を起こす直前の状態には回復できず、例外が起きなかった場合の実行結果と異なる可能性があることに注意してください。

EI レベルの例外処理からの復帰は、EIRET 命令、FE レベル例外処理からの復帰は FERET 命令により行われます。

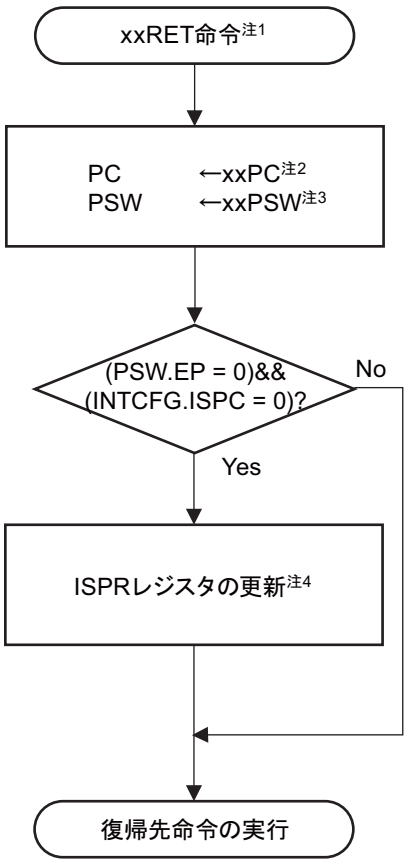
EIRET 命令、FERET 命令の実行により、CPU は次の処理を行い復帰 PC のアドレスへ制御を移します。

<1>　EIRET 命令を実行した場合は EIPC, EIPSW レジスタから復帰 PC, PSW を取り出します。
　　FERET 命令を実行した場合は FEPC, FEPSW レジスタから復帰 PC, PSW を取り出します。

<2>　取り出した復帰 PC のアドレスに制御を移します。

<3>　EIRET 命令実行時には、EP = 0 で、INTCFG.ISPC = 0 の場合、ISPR レジスタを更新します。
　　FERET 命令実行時には、ISPR レジスタの更新は行いません。

EIRET 命令、FERET 命令を使った例外処理からの復帰フローを次に示します。



注 1.　EI レベル例外からの復帰時は EIRET 命令、FE レベル例外からの復帰時は FERET 命令です。
注 2.　EI レベル例外からの復帰時は EIPC、FE レベル例外からの復帰時は FEPC です。
注 3.　EI レベル例外からの復帰時は EIPSW、FE レベル例外からの復帰時は FEPSW です。
注 4.　EIRET 命令の場合のみ

図 4.5　　復帰命令処理フロー

4.4　例外の管理

本 CPU は、マルチプログラミングにおけるプログラム間の相互干渉を防ぐことを目的とした、例外を管理する次の機能を備えています。

- 例外同期命令（SYNCE）
- 保留中例外の確認機能
- 保留中例外の取り下げ機能

本 CPU で定義される例外には、例外の原因が発生したあと、例外処理が開始されるまでの間に遅延が生じる可能性のあるインプレサイス例外が存在します。

本 CPU では、例外管理機能を利用することで、プログラムの切り替えや終了前にそのプログラムに起因するすべての例外を待ち合わせ、順序よく処理することが可能です。これにより、あるプログラムの不正な処理の影響が、別のプログラムにおよぶことを防止します。また、例外を処理することなくプログラムの終了処理を完了してしまうことを防止できます。

4.4.1　例外同期命令

インプレサイス例外は SYNCE 命令により同期することができます。本 CPU ではインプレサイス浮動小数点演算例外（FPI）が該当します。インプレサイス例外の受け付けを任意の時点で行いたい場合は、次の手順にしたがってください。

(1) 受け付けを行うインプレサイス例外の受け付け条件を満たすようなマスクを設定します（PSW.ID, NP のクリアなど）。

(2) 例外同期化命令（SYNCE）を実行します。この時点で、SYNCE 命令以前の命令によって発生するすべてのインプレサイス例外は、必ず CPU に通知が完了した状態になります。ただし、例外の受け付けは（1）で設定した受け付け条件によってマスクされ、保留される場合があります。

(3) (2) の結果、マスクを行わなかった例外があった場合、例外が受け付けられます。このとき、複数の例外要因があった場合は、優先順位にしたがって順番に受け付けられます。

4.4.2　保留中例外の確認と取り下げ

保留されているインプレサイス例外があるかどうか確認したい場合は、次の手順にしたがってください。

(1) 確認を行うインプレサイス例外の受け付け条件を満たさないように、マスクを設定します（PSW.ID, NP のセットなど）。

(2) 例外同期化命令（SYNCE）を実行します。この時点で、SYNCE 命令以前の命令によって発生するすべてのインプレサイス例外は、必ず CPU に通知が完了した状態になります。確認を行う例外は（1）で設定したマスクによって、受け付けが行われず保留されます。ただし、その他の例外が受け付けられる場合があります。

(3) 確認を行うインプレサイス例外の例外通知ビットを読み出します。1 であれば例外が保留されています。

(4) （1）で設定したマスクを必要に応じて解除します。

保留中の例外を受け付けず、例外処理を行わずに取り下げたい場合は、次の手順にしたがってください。

(1) 取り下げを行うインプレサイス例外の受け付け条件を満たさないように、マスクを設定します（PSW.ID, NP のセットなど）。

(2) 例外同期化命令（SYNCE）を実行します。この時点で、SYNCE 命令以前の命令によって発生するすべてのインプレサイス例外は、必ず CPU に通知が完了した状態になります。取り下げを行う例外は（1）で設定したマスクによって、受け付けが行われず保留されます。ただし、その他の例外が受け付けられる場合があります。

(3) 取り下げを行うインプレサイス例外の例外通知ビットをクリアします。

(4) 取り下げが完了したら、（1）で設定したマスクを必要に応じて解除します。

インプレサイス例外の取り下げ機能は、次のレジスタの機能によって提供されます。

表 4.3　保留中インプレサイス例外の確認と取り下げ

例 外	原　因	インプレサイス例外通知ビット	動作
FPI 例外	FPU 命令	FPEC レジスタ FPIVD ビット	FPIVD ビットをクリアすると FPI 例外の通知をクリアすると共に、後続 FPU 命令無効化も解除されます。詳細は、「**3.4.2　浮動小数点機能システム・レジスタ (6)　FPEC — 浮動小数点演算例外の制御**」を参照してください。

4.5　例外ハンドラ・アドレス

本 CPU では、リセット入力時や、例外受付時、割り込み受付時などに実行する例外ハンドラ・アドレスを、設定に応じて変更が可能です。

4.5.1　リセット／例外／割り込み

リセットと例外は、PSW.EBV ビットと、RBASE レジスタ、EBASE レジスタによって例外ハンドラ・アドレスの基準位置が変更できる直接ベクタ方式によって例外ハンドラ・アドレスを決定します。さらに割り込みは、チャネルごとに直接ベクタ方式とテーブル参照方式の選択が可能であり、テーブル参照方式を指定した場合は、メモリ上に配置した例外ハンドラ・テーブルの示すアドレスへ分岐することが可能です。

注　意

直接ベクタ方式において、EIINT*n* の例外ハンドラ・アドレスが、V850E2 製品とは異なります。V850E2 製品では、割り込みチャネル（EIINT*n*）ごとに異なる例外ハンドラ・アドレスが割り当てられていましたが、本 CPU では割り込み優先度ごとに 1 つの例外ハンドラ・アドレスが割り当てられています。このため、同一の割り込み優先度に指定した割り込みは、すべて同じ例外ハンドラへ分岐します。

(1) 直接ベクタ方式

CPU は、RBASE レジスタ、または EBASE レジスタで示されるベース・アドレスに**「表 4.4　ベース・レジスタ／オフセット・アドレスの選択」**のオフセットを加算した値を、例外ハンドラ・アドレスとして利用します。

RBASE レジスタ、EBASE レジスタのいずれをベース・アドレスとして利用するかは、PSW.EBV ビットによって選択します[注 1]。PSW.EBV ビットがセット（1）されている場合は、EBASE レジスタの値をベース・アドレスとして利用します。クリア（0）されている場合は、RBASE レジスタの値をベース・アドレスとして利用します。

ただし、リセット入力と一部の例外[注 2]は、常に RBASE レジスタを参照します。

また、ユーザ割り込みは、それぞれのベース・レジスタの RINT ビットを参照し、その状態によってオフセット・アドレスの縮小を行います。RBASE.RINT ビット、あるいは EBASE.RINT ビットがセット（1）されている場合は、すべてのユーザ割り込みは、オフセット 100_H として扱われます。クリア（0）されている場合は、それぞれ**「表 4.4　ベース・レジスタ／オフセット・アドレスの選択」**にしたがって、オフセット・アドレスが決定されます。

注 1. 例外受け付け自身で、PSW.EBV ビットの状態を更新する場合があります。その場合は、更新後の値を基準に、ベース・レジスタを選択します。詳細は、**「4.5　例外ハンドラ・アドレス」**を参照してください。

注 2. 常に RBASE を参照する例外は、ハードウェア仕様によって定められます。

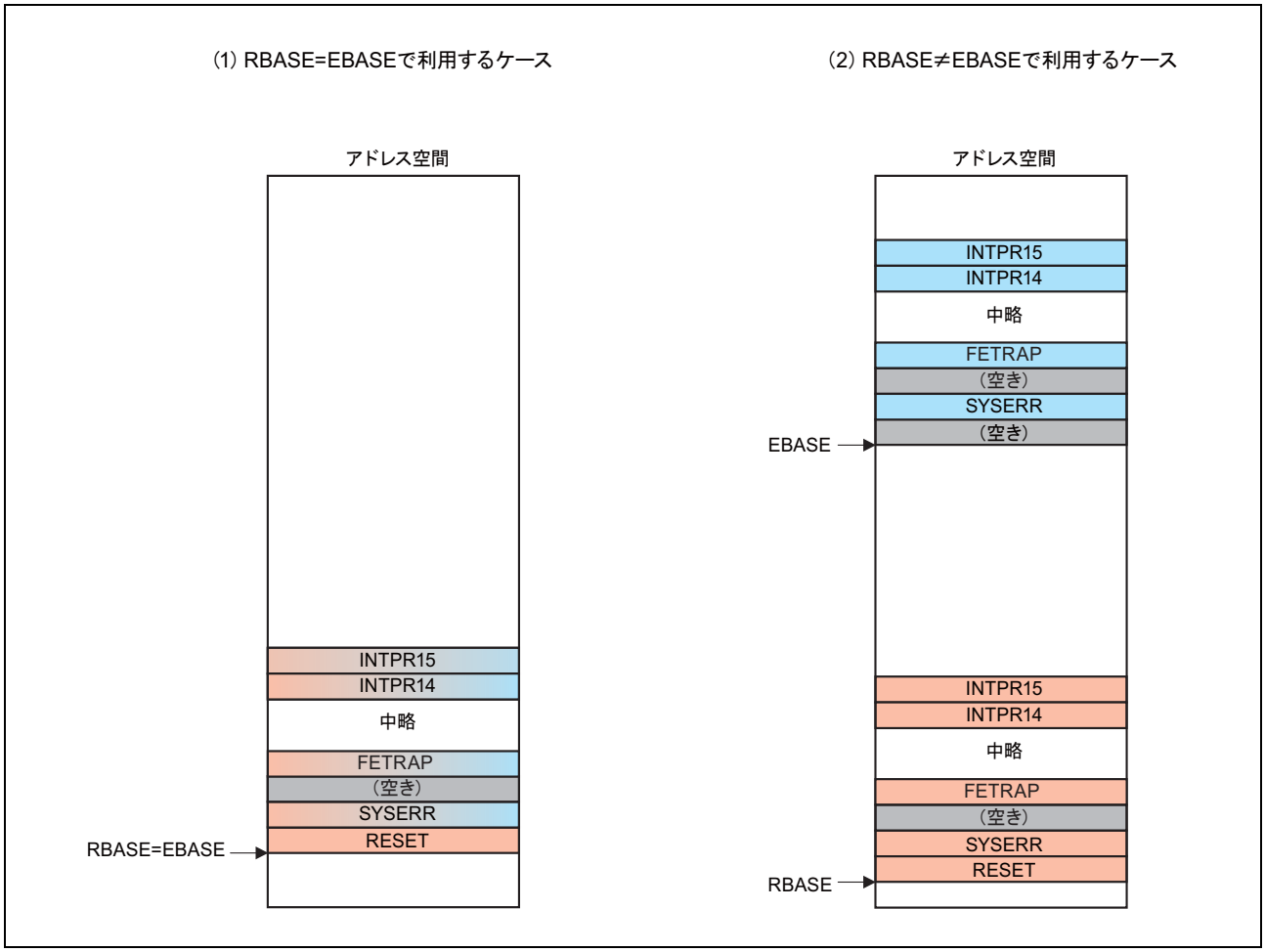


図 4.6　　直接ベクタ方式

備　考

INTPRx は、**「表 4.4　ベース・レジスタ／オフセット・アドレスの選択」**の EIINTn（優先度 x）と同意です。

次の表に、例外ハンドラ・アドレス決定のためのベース・レジスタ選択と、オフセット・アドレスの縮小が、各例外に対してどのように機能するかを示します。PSW のビットの値は、例外の受け付けによる更新後の値を基準に例外ハンドラを決定します。

表 4.4　ベース・レジスタ／オフセット・アドレスの選択

	PSW.EBV = 0	PSW.EBV = 1	RINT = 0	RINT = 1
	ベース・レジスタ		オフセット・アドレス	
RESET	RBASE	なし[注1]	000_H	000_H
SYSERR		EBASE	010_H	010_H
FETRAP			030_H	030_H
TRAP0			040_H	040_H
TRAP1			050_H	050_H
RIE			060_H	060_H
FPP/FPI			070_H	070_H
UCPOP			080_H	080_H
MIP/MDP			090_H	090_H
PIE			$0A0_H$	$0A0_H$
Debug[注2]			$0B0_H$	$0B0_H$
MAE			$0C0_H$	$0C0_H$
(R.F.U.)			$0D0_H$	$0D0_H$
FENMI			$0E0_H$	$0E0_H$
FEINT			$0F0_H$	$0F0_H$
EIINT*n*（優先度 0）			100_H	100_H
EIINT*n*（優先度 1）			110_H	
EIINT*n*（優先度 2）			120_H	
EIINT*n*（優先度 3）			130_H	
EIINT*n*（優先度 4）			140_H	
EIINT*n*（優先度 5）			150_H	
EIINT*n*（優先度 6）			160_H	
EIINT*n*（優先度 7）			170_H	
EIINT*n*（優先度 8）			180_H	
EIINT*n*（優先度 9）			190_H	
EIINT*n*（優先度 10）			$1A0_H$	
EIINT*n*（優先度 11）			$1B0_H$	
EIINT*n*（優先度 12）			$1C0_H$	
EIINT*n*（優先度 13）			$1D0_H$	
EIINT*n*（優先度 14）			$1E0_H$	
EIINT*n*（優先度 15）			$1F0_H$	

注 1.　EBV = 0 に更新する例外のため
注 2.　デバッグ用の例外です。

ベース・レジスタの選択は、リセットと一部のハードウェア・エラーの例外処理を、RAM やキャッシュ領域などのソフト・エラーの影響を受けやすい領域ではなく、ROM などの比較的信頼性の高いプログラムで動作させるための仕組みです。ユーザ割り込みのオフセット・アドレスの縮小は、システム中の特定の動作モードにおいて、例外ハンドラのメモリ占有サイズを小さく保つための機能です。主にシステムのメンテナンスや診断など、最低限の機能のみが動作するモードなどにおいて、メモリ領域の消費を最小限に抑えます。

(2) テーブル参照方式

直接ベクタ方式では、ユーザ割り込みの例外ハンドラは、それぞれの割り込み優先度ごとに1つであり、複数の同一優先度を示す割り込みチャネルは、同じ割り込みハンドラへ分岐しますが、ユーザによっては、割り込みハンドラごとに開始時点からことなるコード領域を利用したい場合などがあります。

テーブル参照方式においては、割り込みコントローラなどにおいて、割り込みチャネルのベクタ選択方式を「テーブル参照方式」に設定すると、その割り込みチャネルに対応する割り込み要求を受け付けた場合の例外ハンドラ・アドレスの決定方法が次のように変わります。

<1>　次のいずれか場合は、直接ベクタ方式にしたがい例外ハンドラ・アドレスを決定します。

- PSW.EBV = 0 かつ RBASE.RINT = 1 の場合
- PSW.EBV = 1 かつ EBASE.RINT = 1 の場合
- 割り込みチャネルの設定が「テーブル参照方式」でない場合

<2>　<1> 以外の場合、テーブルの参照位置を計算します。
例外ハンドラ・アドレス読み出し位置 = INTBP レジスタ + チャネル番号 * 4 バイト

<3>　<2> で計算した割り込みハンドラ・アドレス読み出し位置から、ワード・データを読み出します。

<4>　<3> で読み出した、ワード・データを例外ハンドラ・アドレスとして利用します。

注　意

割り込みチャネルの設定については、製品のハードウェアマニュアルを参照してください。

以下に、各割り込みチャネルに対応する例外ハンドラ・アドレス読み出し位置の表と、メモリ上の配置イメージを示します。

表 4.5　例外ハンドラ・アドレスの拡張

種類	例外ハンドラ・アドレス読み出し位置
EIINT 割り込みチャネル 0	INTBP + 0 * 4
EIINT 割り込みチャネル 1	INTBP + 1 * 4
:	:
EIINT 割り込みチャネル 510	INTBP + 510 * 4
EIINT 割り込みチャネル 511	INTBP + 511 * 4

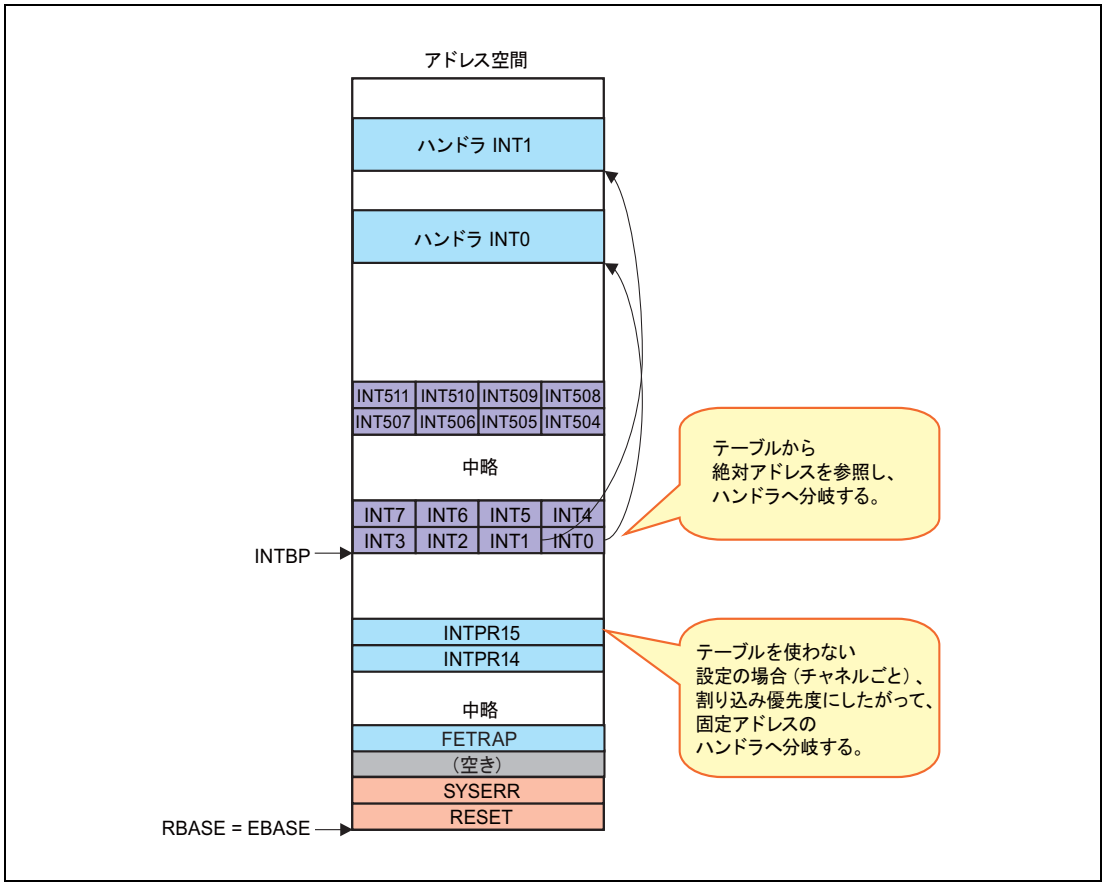


図 4.7　テーブル参照方式利用時のイメージ

割り込みチャネルごとの例外ハンドラ・アドレス選択方式の設定については、製品のハードウェアマニュアルを参照してください。

4.5.2　システム・コール

システム・コール例外は、オペコードによって指定されるベクタの値と SCCFG.SIZE ビットの値によって、参照するテーブル・エントリが選択され、そのテーブル・エントリの内容と SCBP レジスタの値にしたがって例外ハンドラ・アドレスを計算します。

たとえば、SCCFG.SIZE によってテーブル・サイズ n が指定された場合、次のようにテーブル・エントリを選択します。「SYSCALL 命令で指定されたベクタ（vector8）> テーブル・サイズ n」の場合には、ベクタ n ＋ 1 ～ 255 によって参照されるテーブル・エントリは、テーブル･エントリ 0 であることに注意してください。

表 4.6　システム・コール

ベクタ	例外要因コード	参照するテーブル・エントリ
0	0000 8000_H	テーブル・エントリ 0
1	0000 8001_H	テーブル・エントリ 1
2	0000 8002_H	テーブル・エントリ 2
（中略）	:	:
n – 1	0000 8000_H ＋（n – 1）$_H$	テーブル・エントリ n – 1
n	0000 8000_H ＋ n_H	テーブル・エントリ n
n ＋ 1	0000 8000_H ＋（n ＋ 1）$_H$	テーブル・エントリ 0
（中略）	:	:
254	0000 $80FE_H$	テーブル・エントリ 0
255	0000 $80FF_H$	テーブル・エントリ 0

注　意

テーブル・エントリ 0 は、SCCFG.SIZE で指定する n を越えたベクタが指定された場合にも選択されるため、エラー処理ルーチンを配置してください。

4.5.3　利用モデル

RBASE, EBASE と PSW.EBV ビットの関係と、想定する利用モデルを示します。主にリセット時には主要コードがアドレス空間上に存在せず、ブート・ストラップ処理を経てから、はじめてアドレス空間上（多くの場合、DRAM など）に展開され実行可能になるような場合、あるいは例外処理ルーチンに命令キャッシュなどを利用する場合などに利用します。

リセット直後は、PSW.EBV = 0 の状態で、RBASE が指し示す最低限のメンテナンス・コードが配置された ROM 領域を利用し、動作します。ブート・ストラップ処理後、RAM 上に必要なコードが展開されたあと、EBASE レジスタに RAM 上のコード位置を設定し、PSW.EBV をセット（1）します[注 1]。

通常はこの状態でソフトウェアが動作します。通常動作範囲の例外や割り込みなどは、PSW.EBV = 1 のまま受け付けるため、EBASE が示す RAM 上のコードで動作しますが、その RAM コード自体の正当性が確保されないような事象が起きた場合（RAM エラーやキャッシュ・エラーなど）は、PSW.EBV をクリア（0）する例外が発生します[注 2]。この場合、例外ハンドラ自身を EBASE が示す位置のコードでは正しく実行できない可能性があると判断し、RBASE の示す ROM コード上の例外ハンドラに移り、PSW.EBV ビットがクリア（0）されます。

ひとたび、PSW.EBV ビットがクリア（0）されると、この状態で通常の例外が起きた場合も、PSW.EBV ビットの状態は引き継がれ、RAM コードが正しく実行できる状態が確保され、メンテナンス・コードによって、再び PSW.EBV ビットがセット（1）されるまでは、RBASE が示す ROM 上のコードで動作します。

注 1. 通常、PSW.EBV ビットは、EIRET 命令、FERET 命令を用いてセット（1）してください。

注 2. どの例外のどの要因が、PSW.EBV をクリア（0）する例外かどうかは、ハードウェア仕様によって決まります。

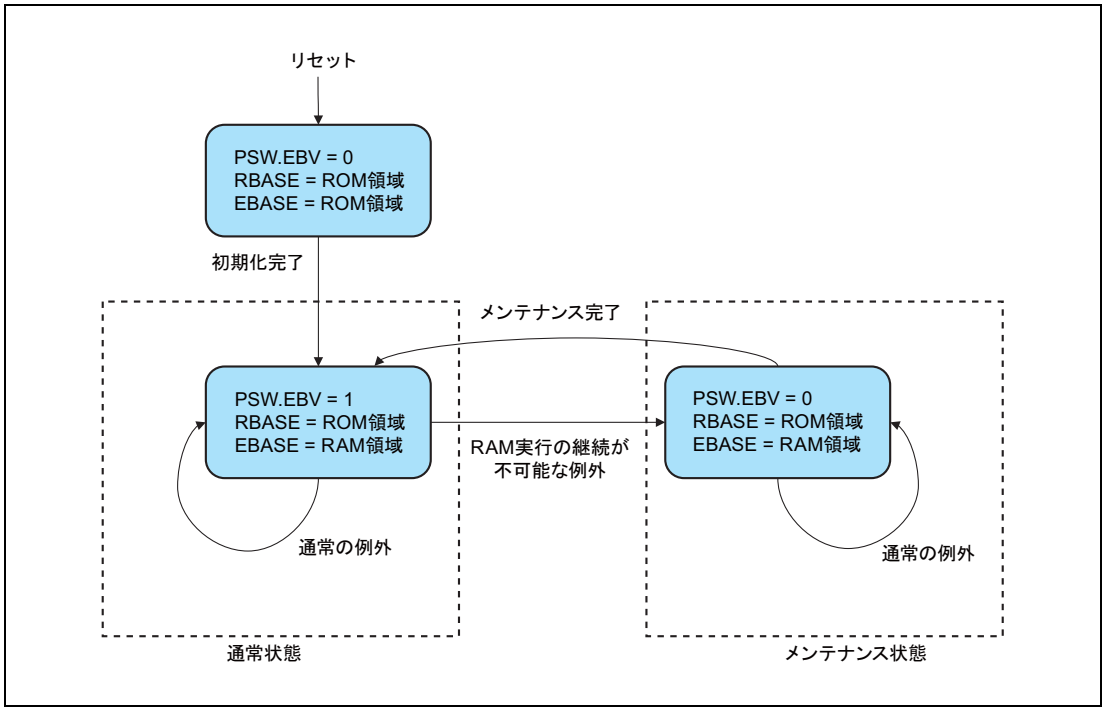


図 4.8　利用モデルの一例（動作フロー）

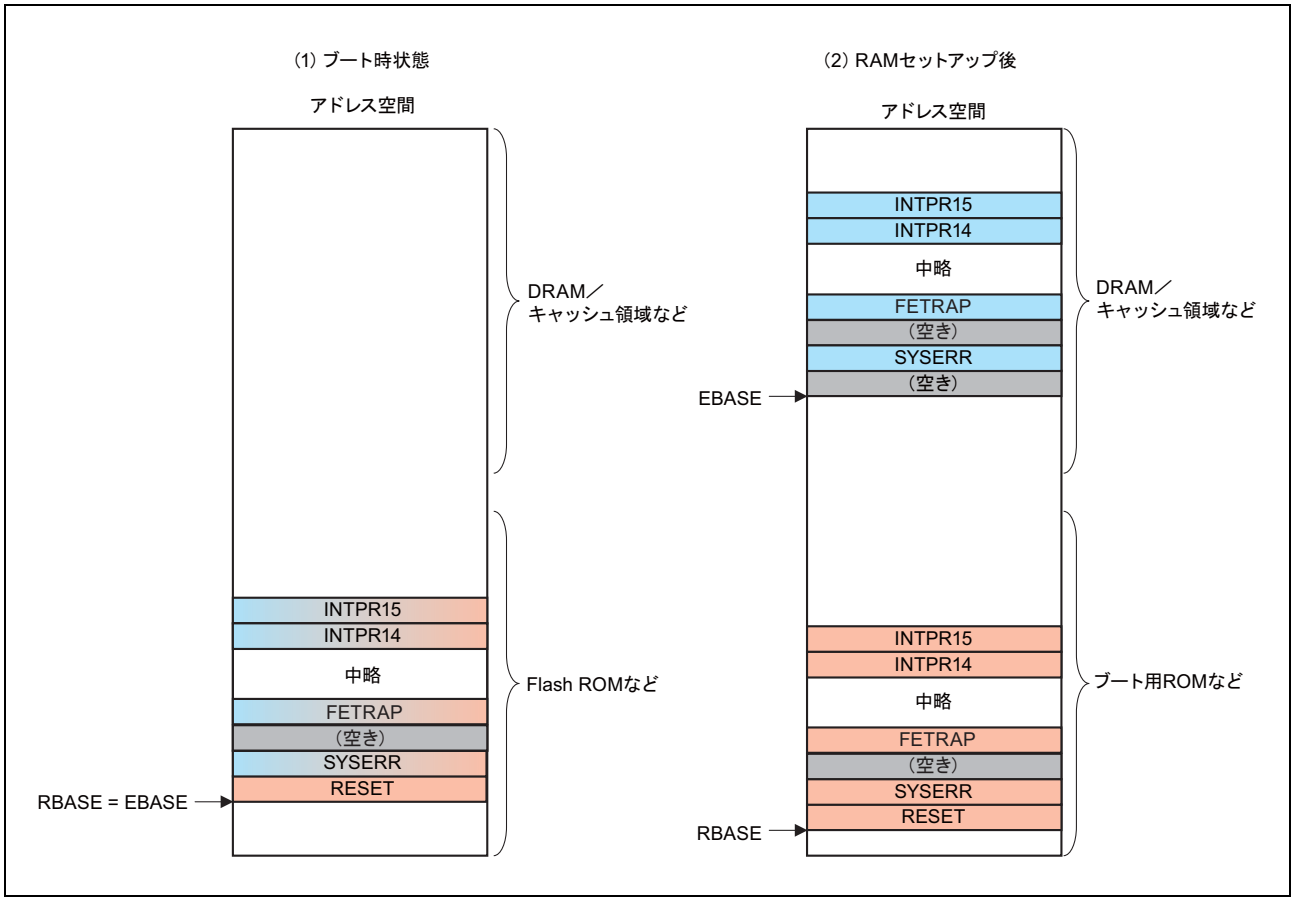


図 4.9　　利用モデルの一例（アドレス・マップ）

第 5 章　メモリ管理

本 CPU では、メモリの管理のための次の機能を備えています。

- メモリ保護機能（MPU）
- 相互排除機能
- 同期化機能

5.1　メモリ保護機能（MPU）

信頼済みでないプログラムや暴走などによるシステム・リソースの不正使用を検出／抑止し、システムの一貫性を維持するためのメモリ保護機能（Memory Protection Unit, MPU）を提供しています。

5.1.1　特　徴

(1) メモリ・アクセス制御

アドレス空間上に保護領域を複数、配置可能です。これによって、ユーザ・プログラムに許可されていない実行、またはデータ操作を検出し、不正な実行、データ操作を防ぐことができます。各領域は上限アドレス／下限アドレスで指定するため、アドレス空間を効率よく細かい粒度で使用できます。

(2) CPU 動作モードごとのアクセス権管理

本 CPU では、リソースへのアクセス制御を行うための状態ビットを複数持っており、これらの状態ビットの組み合わせによって、プログラムごとの信頼性の違いに応じて、適切な保護を行います。

5.1.2　MPU 動作設定

保護領域を利用する前に、スーパバイザ・モードにおいて、MPU 機能の動作設定を行ってください。通常、この設定は OS などの管理ソフトウェアによって行われることを想定しています。

スーパバイザ・モードでの設定は、初期設定とプログラム切り替え時の設定変更、例外処理時の設定変更の 3 つに分けられます。処理フローを次に示します。

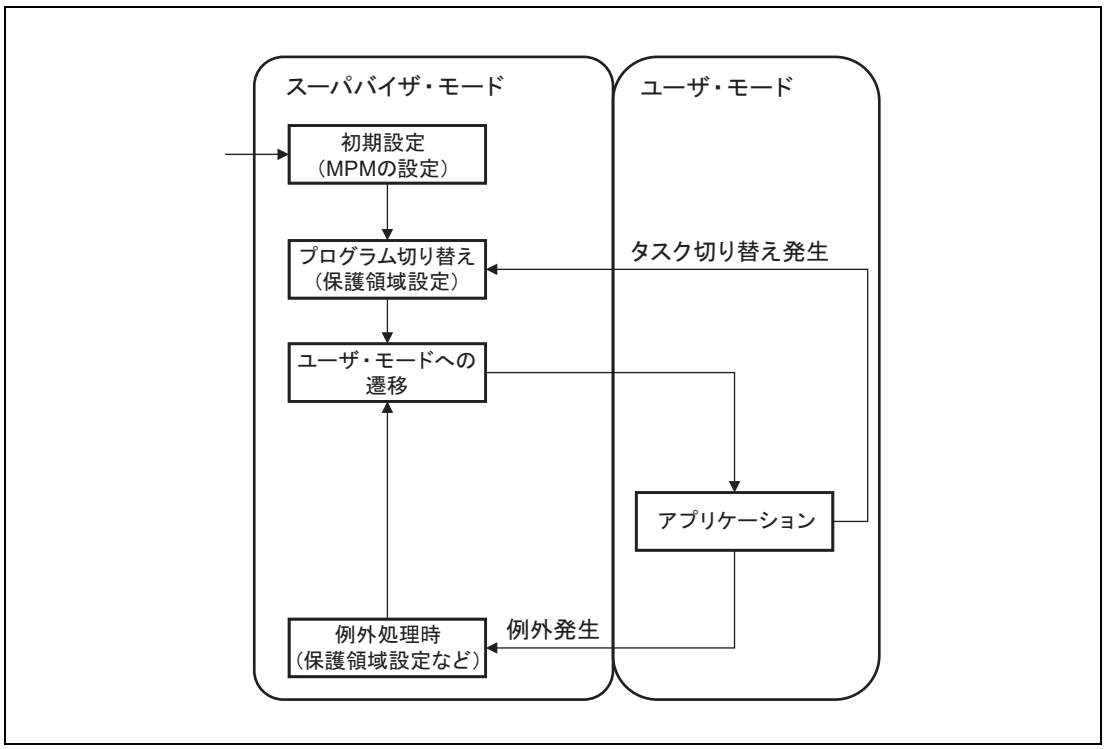


図 5.1　MPU 処理フロー例

初期設定は、MPM レジスタを適切な値に設定します。MPE ビットによる MPU の有効化は、必ず行ってください。SVP ビットは OS 等のスーパバイザにおいても保護を行いたい場合のみ、セット（1）してください。

注　意

SVP ビットをセット（1）する場合のみ、次のような処置を事前に行ってください。

- **保護領域の SR, SW, SX ビットのいずれかをセット（1）する場合、あらかじめその保護領域の MPUAn, MPLAn レジスタを正しく設定してください。**
- **保護領域の SR, SW, SX ビットのいずれもセット（1）しない場合、特に処置は必要ありません。**
- **SVP ビットをセット（1）することで、SVP ビットをセット（1）する管理プログラム（OS）自身が実行不能にならないように、注意して利用してください。もし、設定を誤った場合は、MIP 例外または MDP 例外が再帰的に発生しつづけることで、実行の継続が不可能になる場合があります。**

プログラムの切り替え時に、切り替え後のプログラムに対する保護領域を設定する場合があります。

保護領域の設定については、**「5.1.3　保護領域設定」**を参照してください。

例外処理においては、通常のエラー処理として、復帰を行わない処理とは異なり、例外を起こしたアドレスの利用可否を管理プログラムで判断し、実行を継続するデマンド・ページング時に、保護領域を入れ替える場合があります。

プログラム切り替え時と同様、保護領域の設定変更については、**「5.1.3　保護領域設定」**を参照してください。

5.1.3　保護領域設定

(1) 保護領域の設定

それぞれの保護領域を適切に設定してください。レジスタの詳細は**「第 3 章　レジスタ・セット」**を参照してください。

ここでは、特に注意が必要な点について補足説明を行います。

(a) E ビット

その保護領域設定の有効／無効を示します。無効の場合、すべての設定値は無効となります。このビットをセット（1）すると同時に他の保護領域設定（MPUA, MPLA, MPAT）には、必ず有効な値が格納されていることを保証してください。

(b) UX, UR, UW ビット

ユーザ・モードにおけるその保護領域のアクセス権を示します。

(c) SX, SR, SW ビット

スーパバイザ・モードにおける、その保護領域のアクセス権を示します。このビットは、MPM.SVP ビットがセット（1）されている場合のみ有効です。MPM.SVP ビットがクリア（0）されている場合は、SX, SR, SW ビットの値にかかわらず、スーパバイザ・モードでは保護が行われず、すべてのアドレス空間にアクセスが可能です。

(d) G ビット、ASID フィールド

グローバル・ビットと、比較用の ASID フィールドです。このビットがクリア（0）されている場合、ASID レジスタの値と、MPAT.ASID フィールドの値を比較し、一致した場合にのみ、保護領域の設定を適用して、アクセスの可否を決定します。セット（1）した場合は、ASID にかかわらず保護領域の設定が適用されます。

5.1.4　保護領域設定時の注意事項

(1) 保護領域境界の交差

保護領域の範囲が重複して設定されている場合は、交差部分に対するアクセス制御の設定は、MPM.DX, DW, DR ビットの設定によって変化します。デフォルト動作が禁止である場合は許可優先となり、許可である場合は、禁止優先となります。

つまり、あるアドレスにおいて、デフォルト動作が禁止であり、複数の保護領域が設定されている場合は、いずれかの保護領域で許可にされていれば、許可として判断されます。また、デフォルト動作が許可である場合は、いずれかの保護領域で禁止にされていれば、禁止として判断されます。

なお、本 CPU では MPM.DX, DW, DR ビットは 0 固定であり、デフォルト動作は禁止です。

(2) 無効な保護領域の設定

次の場合に、保護領域の設定は無効となります。

- 下限アドレスを上限アドレスよりも大きな値に設定した場合

注　意

ただし、アドレスを符号なし整数（0_H ～ FFFF FFFF$_H$）として取り扱います。

(3) 連続した MPU アクセス許可領域をまたぐメモリアクセス

ロードストア・アクセスに対する MPU メモリ保護ではメモリ・アクセスがある 1 つのアクセス許可領域に完全に含まれている必要があります。アクセス許可領域が連続した空間に配置されている場合でも、領域をまたぐ場合にはアクセスが許可されません。本 CPU では領域をまたぐメモリ・アクセスは ld.dw/st.dw 命令によるダブルワード・アクセスでのみ発生します。prepare/dispose 命令、pushsp/popsp 命令はワードアクセスの繰り返しとして扱われ、MPU アクセス許可領域をまたいでアクセスが可能です。

命令フェッチに対するメモリ保護では、ある命令が許可領域をまたいだ場合で命令コード全体がいずれかのエントリで許可されていれば許可されます。

5.1.5　アクセス制御

本 CPU は、**「5.1.3　保護領域設定」**までの設定にしたがって、適切なアクセス制御を行います。以下のような場合に該当する場合、CPU は実際のアクセスを制限し、命令の実行完了前に違反を検出し、例外を発生させることで論理的な正当性を確保します。

- 実行許可とされた領域以外のアドレスを、命令オペコードに含む命令を実行しようとした場合
- リード許可とされた領域以外のアドレスに対してリードを行う命令を実行しようとした場合
- ライト許可とされた領域以外のアドレスに対してライトを行う命令を実行しようとした場合

アクセス制御の内容は、ハードウェア仕様によって、細部は異なりますが以下の点は共通です。

- 禁止と判断されたアクセスの結果は、メモリ、I/O 装置などに反映されない。
- 許可と判断されたアクセスの結果は、メモリ、I/O 装置などに反映される。

注　意

1. **許可とされたアクセスのうち、他の機能で禁止として、アクセスがブロックされる場合などがあります。**
2. **一部のケースにおいては、禁止と判断されたアクセスがメモリや I/O 装置に対して実行されることがあります。以下のケースが該当します。**
   - **Local RAM へのリード**
   - **命令キャッシュのプリフェッチによる Code Flash のリード**

   Local RAM をリードした命令や、プリフェッチした結果を実行しようとする命令は、例外によって実行を抑止されますので、これらのアクセスが命令実行に影響を与えることはありません。ただし、デバッガによって Local RAM や Code Flash のアクセスを監視しているとき、禁止と判断されたアクセスが観測される場合があります。

5.1.6　違反と例外

本 CPU では、命令フェッチ・アクセス時と、オペランド・アクセス時に、保護領域設定にしたがって違反を検出し、例外を発生します。

- 実行保護違反（命令フェッチ・アクセス時）
- データ保護違反（オペランド・アクセス時）

(1) 実行保護違反（MIP 例外）

命令の実行時に検出される違反です。プログラム領域上で実行が許可されていない領域に配置された命令を実行しようとした場合、実行保護違反が検出されます。

実行保護違反が検出された場合、常に MIP 例外を発生します。

(2) データ保護違反（MDP 例外）

命令のデータ・アクセス時に検出される違反です。メモリ・アクセス命令が、データ領域上で許可されていない領域のデータをアクセスしようとした際に検出されます。

データ保護違反が検出された場合、常に MDP 例外を発生します。

(3) 例外要因コードと例外アドレス

命令保護違反、データ保護違反を検出した場合に、例外要因コードは**表 5.1** にしたがって決定されます。決定した例外要因コードは、FEIC レジスタにセットされます。

命令保護違反時は違反を検出した命令の PC、データ保護違反時はアクセス・アドレスが MEA レジスタに格納されます。MIP 例外と MDP 例外は同時に発生することがないため、MEA レジスタは MDP 例外と共用です。また、データ保護違反時には MEI レジスタに違反を起こした命令の情報が格納されます。

表 5.1　メモリ保護違反時の例外要因コード

例外	違反時の動作モード	ビット番号とビット名										
		31-25	24	23	22	21	20	19	18	17	16	15-0
		—	MS	BL	RMW	SX	SW	SR	UX	UW	UR	—
MIP	ユーザ・モード	0	0	0	0	—	—	—	—	—	—	90_H
	スーパバイザ・モード	0	0	0	0	—	—	—	—	—	—	90_H
MDP	ユーザ・モード	0	注 5	注 4	注 3	0	0	0	0	注 2	注 1	91_H
	スーパバイザ・モード	0				0	注 2	注 1	0	0	0	91_H

注 1.　リード操作を含む命令でリード違反が起きた場合は SR または UR ビットがセット（1）されます。
注 2.　ライト操作を含む命令でライト違反が起きた場合は SW または UW ビットがセット（1）されます。
注 3.　SET1, NOT1, CLR1, CAXI 命令で違反が発生した場合にセット（1）されます。
注 4.　PREPARE, DISPOSE, PUSHSP, POPSP 命令で違反が発生した場合にセット（1）されます。
注 5.　違反を引き起こした命令がミスアライン・アクセスを行う場合にセット（1）されます。

備考　UR　：ユーザ・モード（PSW.UM = 1）で、リード操作時に違反を検出した。
UW　：ユーザ・モード（PSW.UM = 1）で、ライト操作時に違反を検出した。
UX　：ユーザ・モード（PSW.UM = 1）で、命令実行時に違反を検出した。
SR　：スーパバイザ・モード（PSW.UM = 0）で、リード操作時に違反を検出した。
SW　：スーパバイザ・モード（PSW.UM = 0）で、ライト操作時に違反を検出した。
SX　：スーパバイザ・モード（PSW.UM = 0）で、命令実行時に違反を検出した。
RMW　：違反を引き起こした命令がリード・モディファイ・ライト動作（SET1, NOT1, CLR1, CAXI）を含む場合にセット（1）されます。
BL　：違反を引き起こした命令がブロック転送（PREPARE, DISPOSE, PUSHSP, POPSP）を行う場合にセット（1）されます。
MS　：違反を引き起こした命令がミスアライン・アクセスを行う場合にセット（1）されます。

5.1.7　メモリ保護設定チェック機能

本 CPU は、OS 等のサービス提供を行うプログラムの構築にあたり、依頼された操作のためのデータ領域が、サービスの呼び出し元のアクセス権限内であるかどうかを事前に確認するサービス・プロテクション機能を実現するために、メモリ保護設定チェック機能を提供します。OS はこの機能を用いることで、ユーザから提供されるシステム・サービスに対するパラメータの正当性を検証することができます。また、ソフトウェアによる領域設定の読み出し、比較作業を繰り返すよりも短時間でこの検証処理を完了することができます。

(1) 手順

MCA レジスタに対象アドレス範囲のベース・アドレス（下限）、MCS レジスタに対象アドレス範囲のサイズを設定し、MCC レジスタへ LDSR 命令（r0 指定を推奨）でアクセスを行うことによって、チェックを実行します。実行結果は、STSR 命令によって MCR レジスタを読み出すことで得ます。

注　意

1. **チェック対象の領域の指定が 0000 0000_H または 7FFF $FFFF_H$ をまたぐ場合、領域指定が誤っていると判断し、MCR.OV ビットがセット（1）されます。このため、チェック結果を参照する場合には、必ず MCR.OV ビットを確認し、結果が不正でないこと（OV = 0）を確認してから、その他のチェック結果を利用してください。**
2. **デフォルト設定（MPM.DX, DW, DR）をセット (1) している場合、正しい結果が得られない場合があります。デフォルトを許可にする場合、メモリ保護設定チェック機能は利用しないでください。**

(2) サンプル・コード

メモリ保護設定チェック機能は次のような操作によって、利用することを想定しています。

```
_service_protection:
        …
        ori     0x1000, r0, r12
        …
        mov     ADDRESS, r10        // チェックする領域の先頭アドレスを r10 に格納
        mov     SIZE, r11           // チェックする領域のサイズを r11 に格納
        di
        ldsr    r10, sr8, 5         // MCA にアドレスをセット
        ldsr    r11, sr9, 5         // MCS にサイズをセット
        ldsr    r0, sr10, 5         // MCC によりチェック開始
        stsr    sr11, r12, 5        // MCR から結果の取得
        ei
        andi    0x0100, r12, r0
        bnz     _overflow           // OV = 1 の場合は入力値不正として処理
        br      _result_check       // それ以外の場合は結果を判定
```

5.2　キャッシュ

本 CPU は、キャッシュ機能は非搭載です。

5.2.1　CACHE/PREF 命令の実行権限

CACHE 命令、PREF 命令は、ユーザ特権での実行が許可されています。

5.3　相互排除

本 CPU では、マルチ・プロセッシング環境において、複数のプログラムからの共有資源に対する相互排除制御を行うための命令を用意しています。

相互排除にあたっては、メモリ上に相互排除変数を定義し、双方のプログラムが適切な命令フローにしたがって動作することを必要としています。

注　意

シングルプロセッサ構成の組み込み CPU では、マスカブル割り込みの受け付けを禁止することによって、データの一貫性を保つプログラミング・モデルが利用されてきました。これは非常に容易かつ確実な方法でしたが、当然ながらマルチプロセッサ・システムでは、データを操作しようとするプログラムが複数同時に動作する可能性があるため、マスカブル割り込み受け付けを禁止するだけでは、データの一貫性を保つことはできません。

5.3.1　相互排除処理を必要としない共有データ

本 CPU では、次のデータ・アクセスではマルチ・プロセッシング環境でもデータ・アクセスの一貫性が保たれます。

- データ・タイプと一致するサイズで整列されたアクセス（アライン・アクセス）
  - LD, ST, SLD, SST, LDL, STC 命令
- ビット操作命令（SET1, CLR1, NOT1）によるアクセス（リード・モディファイ・ライト）
- CAXI 命令によるアクセス（リード・モディファイ・ライト）

一部の例外を除いて、一般にはこれらのデータ・アクセスでは、相互排除が達成されます。すなわち、CPU が上記のアクセスを行う命令を実行している間、他の CPU は対象データをアクセスすることがないことを保証しています。これを「命令がアトミックに実行される」、あるいは「アトミック性が保証される」と呼びます。

命令のアトミックな実行は、データ・アクセスのバス・レベルのトランザクションが分断せずに完了することを意味しますが、一連のトランザクションの完了とは必ずしも一致しないことに注意する必要があります。

注　意

ハードウェア仕様によって、一貫性保証の範囲に制限が課せられる場合があります。例えば、対象メモリによってはアライン・アクセスであっても一貫性が保たれない可能性があります。詳細は、製品のハードウェアマニュアルを参照してください。

5.3.2　LDL.W, STC.W 命令による相互排除

複数のデータ列にまたがった相互排除を行う場合に LDL.W, STC.W 命令を利用することが可能です。

LDL.W, STC.W 命令のペアによるロック獲得は、まず LDL.W 命令の実行によるリンクの生成を行い、次にリンクが生成されている状態で STC.W 命令を実行することで達成されます。

このとき、リンクの生成から STC.W 命令を実行するまでの間、リンク生成時のアドレスと同一のアドレスに対する書き込み操作があった場合には、直ちにリンクが消失し、後続の STC.W 命令が成功せず、ロックの獲得に失敗します。

(1) リンク

リンク（LLbit）は、リンク生成時のアドレス情報を含んでおり、このアドレスに依存して STC 命令の成功／失敗とリンクの消失の制御を行います。

リンクの生成は、LDL.W 命令を実行した時に行われます。リンクが生成されている状態で、更に LDL.W 命令を実行した場合、後続のリンク生成によって先行するリンクが上書きされます。つまり、後続 LDL.W 命令のアドレス情報を持ったリンクが 1 つ存在する状態となります。

リンクの消失はいくつかのイベント条件と、アドレス条件があります。**表 5.2** にリンクの消失条件を示します。**表 5.2** のいずれかの行に記載された条件が成立した場合、リンクが消失します。

表 5.2　リンクの消失条件

対象となるリンク	イベント条件	備考
システム内のすべてのリンク（他 CPU コアも含む）	リンク・アドレスを含む 32 バイト整列のアドレス範囲に対する書き込み操作が行われた場合	ST, SST, STC 命令 SET1, NOT1, CLR1, CAXI 命令 PREPARE, PUSHSP 命令
自 CPU コアのリンク	STC.W 命令の実行	成功／失敗にかかわらずリンクは消失します。
	CLL 命令の実行	関数内で明示的にリンクをクリア（アトミック操作中断）する場合、CLL 命令を使用してください。
	例外の受け付け	
	復帰命令の実行	CTRET 命令は含まない

注　意

書き込み操作によるリンクの消失は、32 バイト単位で行われます。このため、STC.W が失敗する可能性を最大限避けるためには、相互排除変数を 32 バイト範囲に 1 つだけ配置することを推奨します。32 バイト範囲に複数の相互排除変数を配置した場合に、相互排除変数へのロック獲得操作によるスラッシングが発生する可能性があります。

(2) サンプル・コード

次に LDL.W, STC.W を使った spinlock のサンプル・コードを示します。

ロックの取得

```
        mov     lock_adr, r20
Lock:   ldl.w   [r20], r21
        cmp     r0, r21
        bnz     Lock_wait
        mov     1, r21
        stc.w   r21, [r20]
        cmp     r0, r21
        bnz     Lock_success
Lock_wait:
        snooze
        br      Lock
 Lock_success:
```

ロックの解放

```
        st.w    r0, 0[r20]
```

5.3.3　SET1 命令による相互排除

複数のデータ列にまたがった相互排除を行う場合に SET1 命令を利用することが可能です。メモリ上の同一ビットに対して SET1 命令を実行し、実行結果の PSW.Z フラグによって、ロックの成功／失敗を判断します。

注　意

1. **ハードウェア仕様によっては、バス・システムを長時間占有するため頻繁に SET1 による排他制御を行うとシステム性能が低下する場合があります。このため、排他制御には極力 LDL/STC を利用してください。**
2. **SET1 命令による相互排除を行う場合、注意 1 と同様の理由でバス・システムの長時間占有を避けるため、ロック獲得に失敗した場合には、再度ロックを取得する前に snooze 命令を実行し、ロック取得ループの実行間隔を調整してください。**

(1) サンプル・コード

次に SET1 を使った spinlock のサンプル・コードを示します。

ロックの取得

```
        mov     lock_adr, r20
Lock:   set1    0, 0[r20]
        bz      Lock_succes
        snooze
        br      Lock
Lock_succes:
```

ロックの解放

```
        clr1    0, 0[r20]
```

5.3.4　CAXI 命令による相互排除

複数のデータ列にまたがった相互排除を行う場合に CAXI 命令を利用することが可能です。メモリ上の同一ワードに対して CAXI 命令を実行し、デスティネーション・レジスタの値によって、ロックの成功／失敗を判断します。

注　意

1. **ハードウェア仕様によっては、バス・システムを長時間占有するため頻繁に CAXI による排他制御を行うとシステム性能が低下する場合があります。このため、排他制御には極力 LDL/STC を利用してください。**
2. **CAXI 命令による相互排除を行う場合、注意 1 と同様の理由でバス・システムの長時間占有を避けるため、ロック獲得に失敗した場合には、再度ロックを取得する前に snooze 命令を実行し、ロック取得ループの実行間隔を調整してください。**

(1) サンプル・コード

次に CAXI を使った spinlock のサンプル・コードを示します。

ロックの取得

```
        mov     lock_adr, r20
Lock:   mov     1, r21
        caxi    [r20], r0, r21
        bz      Lock_succes
        snooze
        br      Lock
Lock_succes:
```

ロックの解放

```
        st.w     r0, 0[r20]
```

5.4　同期化機能

本 CPU は、処理性能を上げるため、先行命令と後続命令の間に依存関係がない場合には、先行命令の処理完了を待ち合わせずに後続命令を実行します。このため、ソフトウェアとして後続命令が先行命令の処理完了を確実に待ち合わせる必要がある場合には同期化の手順が必要になります。本 CPU には同期化を行うための専用命令を 4 つ用意しています。

SYNCP 命令は、先行命令の実行結果を後続命令に反映させるため、明示的に後続命令の実行開始を待ち合わせるパイプライン同期化のための特殊命令です。SYNCP 命令はロード命令の実行結果（ロード値を汎用レジスタに格納するまで）を待ち合わせますが、ストア命令の実行結果（ストア先のメモリやメモリ・マップされた制御レジスタの更新）は待ち合わせませんので、ストア命令の実行結果を後続命令に反映させたい場合には、ストア命令の後に、同メモリや制御レジスタへのダミーリードを行い、SYNCP 命令を実行します。

SYNCM 命令は、先行するすべてのロード命令の実行結果（ロード値を汎用レジスタに格納するまで）と先行するすべてのストア命令の実行結果（ストア先のメモリやメモリ・マップされた制御レジスタの更新）を待ち合わせるメモリアクセス同期化のための特殊命令です。ただし、投機的にストア完了（メモリや制御レジスタ末端までの書き込み完了が遅延）するバスシステム・周辺装置に対しては、SYNCM 命令は完全なストア命令の実行完了保証にならないことがあります。このようなメモリや制御レジスタの更新結果を確実に後続命令に反映させたい場合には、同メモリや制御レジスタへのダミーリードを行い、SYNCP 命令を実行してください。

SYNCI 命令は、すでにパイプラインに取り込まれている未実行の後続命令を捨て去り、改めて後続命令の命令フェッチを明示的に行う命令フェッチ同期化のための特殊命令です。先行命令の実行結果を後続の命令フェッチに反映させる場合に使用します。自己書き換えプログラム／ Code Flash の領域の切り替えなどで、ストア命令の実行結果を後続命令の命令フェッチに反映させる必要がある場合には、ダミーリードおよび SYNCP 命令によりストア命令の実行結果を待ち合わせた上で、SYNCI 命令を実行してください。

SYNCE 命令は、インプレサイス例外（FPI 例外）を同期化するための特殊命令です。先行命令によって発生していた FPI 例外を確実に受け付けたい場合に使用します。タスク切り替えやタスク終了前にそのタスクでの例外処理の完了を保証するために使用することができます。

表 5.3 に同期化命令の効果をまとめます。

システム・レジスタのハザード解消手続きについては、**「付録 A　システム・レジスタのハザード解消手続き」**を参照してください。

表 5.3　同期化命令の効果

同期化命令	SYNC 命令が保証する同期化（SYNC 命令に先行する各命令の実行完了待ち合わせ）					
	命令フェッチの同期化		命令実行の同期化			
	後続命令のフェッチ	キャッシュ命令・キャッシュ操作機能レジスタの変更命令注3	演算命令	ロード命令	ストア命令	FPI 例外
SYNCP	—	—	実行完了	実行完了注1	—	—
SYNCM	—	—	実行完了	実行完了注1	実行完了注2	—
SYNCI	命令実行の同期化完了後再フェッチ	実行完了	実行完了	—	—	—
SYNCE	—	—	—	—	—	例外受付

備考　“—” は待ち合わせない

注 1.　ロード値を汎用レジスタに格納するまでを意味します。

注 2.　ストア先のメモリやレジスタの更新を意味します。ただし書き込み先により保証できない場合があります。詳細は、製品のハードウェアマニュアルを参照してください。

注 3.　キャッシュ命令は NOP 命令として処理されます。キャッシュ操作機能レジスタは非搭載です。

第 6 章　コプロセッサ

6.1　浮動小数点演算

浮動小数点ユニット（FPU）は、CPU のコプロセッサとして動作し、浮動小数点演算命令を実行します。

単精度（32 ビット）データが使用できます。また、浮動小数点値と整数値の変換も可能です。

本 CPU の FPU は ANSI/IEEE 標準規格 754-2008「IEEE Standard for Floating-Point Arithmetic」に準拠しています。

(1) 浮動小数点演算命令

- 最大値、最小値を演算する命令をサポート
  MAXF.S, MINF.S
- 浮動小数点設定／状態レジスタの条件ビットを PSW レジスタの Z フラグに転送する、フラグ転送命令をサポート
  TRFSR
- 条件付き転送命令をサポート
  CMOVF.S
- 符号なし整数との型変換を効率よく実行する、符号なし変換命令をサポート
- 最近接整数への型変換を効率よく実行する、CEIL 命令、FLOOR 命令をサポート
- 高い精度で積和計算を実行する、結合積和演算命令をサポート
- 効率よいデータ格納を目的とした半精度浮動小数点形式変換命令をサポート
- 浮動小数点の比較結果を格納する 8 ビットの条件ビットをサポート
- FPU の実行モードとしてプレサイス・モードとインプレサイス・モードをサポート

(2) レジスタ・セット

- 浮動小数点レジスタ：汎用レジスタを使用
  (浮動小数点演算専用のレジスタはありません)。
- 浮動小数点システム・レジスタ：FPSR — 浮動小数点演算の設定／ステータス
  FPEPC — 浮動小数点演算例外プログラム・カウンタ
  FPST — 浮動小数点のステータス
  FPCC — 浮動小数点演算の比較結果
  FPCFG — 浮動小数点機能の設定
  FPEC — 浮動小数点演算例外の制御

6.1.1　浮動小数点演算機能の構成

(1) 非搭載

浮動小数点演算機能を搭載しない場合は、すべての浮動小数点演算命令は使用不可となり、実行しようとした場合、コプロセッサ使用不可例外が発生します。また、すべての浮動小数点システム・レジスタも動作不定となるため、LDSR/STSR による操作は行わないでください。

(2) 搭載

浮動小数点演算機能を搭載する場合は、単精度浮動小数点演算命令が使用可能です。

また、すべての浮動小数点システム・レジスタは、**「3.4　FPU 機能レジスタ」**で示される機能を提供します。

6.1.2　データ・タイプ

(1) 浮動小数点の形式

FPU は、32 ビット（単精度）の IEEE754 浮動小数点演算をサポートします。

単精度浮動小数点形式は、**図 6.1** に示すように 24 ビットの符号付き仮数部（s ＋ f）と、8 ビットの指数部（e）で構成されます。

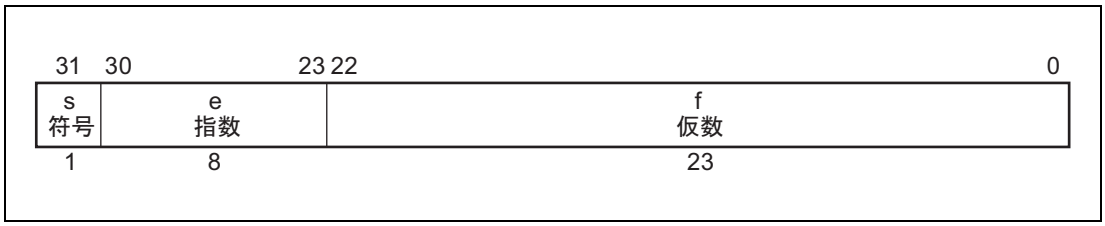


図 6.1　単精度浮動小数点形式

浮動小数点形式の数値は、次の 3 つの領域により構成されます。

- 符号ビット ：s
- 指数部　　：e = E ＋ バイアス値
- 仮数部　　：f = . $b_1b_2...b_{p-1}$（小数点第 1 位以下の値）

単精度形式の場合、バイアス値は 127 です。

バイアスしていない指数値 E の範囲は、Emin から Emax までのすべての整数値と 2 つの予約値、Emin － 1（±0、あるいはサブノーマル数）と、Emax ＋ 1（± ∞、あるいは NaN：非数）です。0 以外の数値の表現は、1 つの形式で表現されます。

この形式で表現される数値（v）は、**表 6.1** に示す式によって求められます。

表 6.1　浮動小数点値の計算式

種 類	計算式	
NaN（非数）	$E = E_{max} + 1$ かつ $f \neq 0$	ならば v は s にかかわらず NaN
± ∞（無限大数）	$E = E_{max} + 1$ かつ $f = 0$	ならば $v = (-1)^s \infty$
ノーマル数（正規化数）	$E_{min} \leqq E \leqq E_{max}$	ならば $v = (-1)^s 2^E (1.f)$
サブノーマル数（非正規化数）	$E = E_{min} - 1$ かつ $f \neq 0$	ならば $v = (-1)^s 2^{Emin} (0.f)$
±0（ゼロ）	$E = E_{min} - 1$ かつ $f = 0$	ならば $v = (-1)^s 0$

- NaN（非数）

  IEEE754 では、NaN（Not a Number）という浮動小数点値を規定しています。これは非数とも呼ばれ、数値ではないため大小関係もありません。
  すべての浮動小数点形式において、v が NaN であった場合、f の最上位ビットの値によって SignalingNaN（S-NaN）か、QuietNaN（Q-NaN）のどちらかになります。f の最上位ビットがセットされている場合は QuietNaN で、クリアされている場合は SignalingNaN です。

浮動小数点の形式で定義されている各パラメータの値を**表 6.2** に示します。

表 6.2　浮動小数点形式とパラメータ値

パラメータ	形式
	単精度
Emax	+127
Emin	–126
指数部のバイアス値	+127
指数部の長さ（ビット数）	8
整数ビット	見えない
仮数部の長さ（ビット数）	23
形式の長さ（ビット数）	32

この浮動小数点形式で表現できる最小値、最大値を**表 6.3** に示します。

表 6.3　浮動小数点の最大値、最小値

タイプ	値
単精度浮動小数点の最小値	1.40129846e – 45
単精度浮動小数点の最小値（ノーマル）	1.17549435e – 38
単精度浮動小数点の最大値	3.40282347e ＋ 38

(2) 整数の形式

整数の値は 2 の補数の形式で保持されます**図 6.2** に 32 ビット整数形式、**図 6.3** に 64 ビット整数形式を示します。符号なし整数においては、符号ビットは存在せず、全ビットが整数値を表現します。

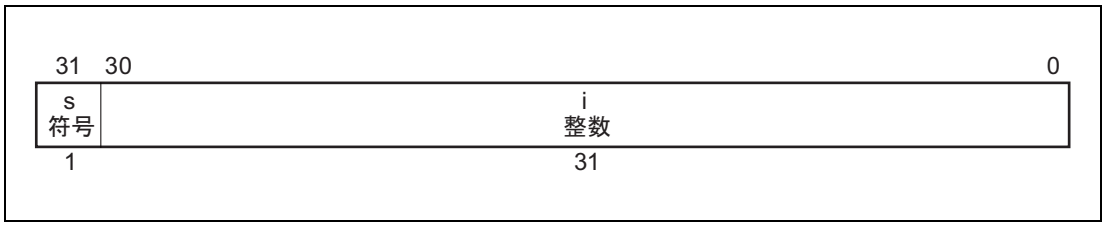


図 6.2　32 ビット整数形式

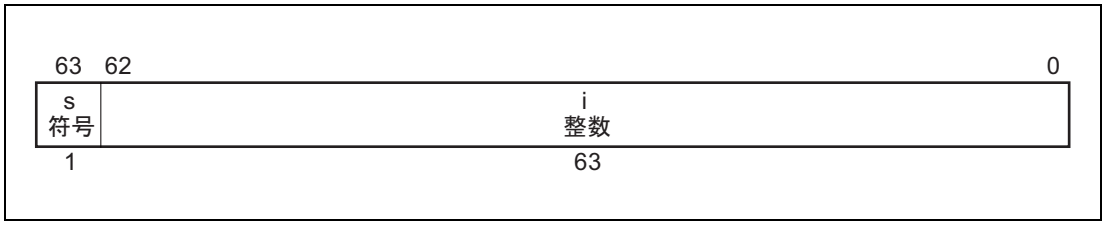


図 6.3　64 ビット整数形式

(3) 浮動小数点の拡張形式

データ格納用の浮動小数点形式として 16 ビット（半精度）の IEEE754 浮動小数点形式をサポートします。半精度浮動小数点形式はデータ量の削減を目的としており、演算はサポートされません。単精度浮動小数点形式との間で変換命令が用意されます。半精度浮動小数点形式は、**図 6.4** に示すように 11 ビットの符号付き仮数部（s ＋ f）と、5 ビットの指数部（e）で構成されます。

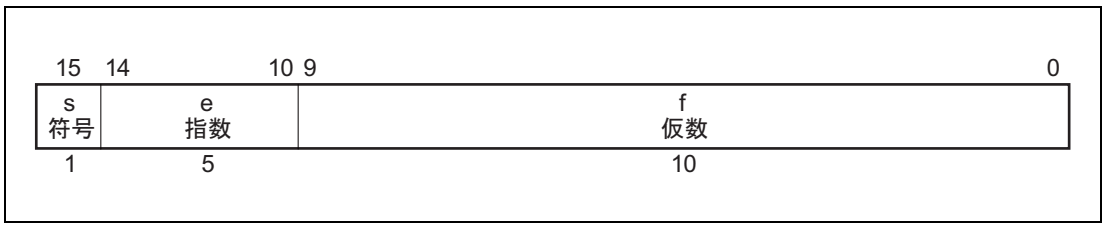


図 6.4　半精度浮動小数点形式

半精度浮動小数点形式で表現される数値は、他の浮動小数点形式と同様に**表 6.1** に示す式によって求められます。半精度浮動小数点の形式で定義されている各パラメータの値を**表 6.4** に示します。

表 6.4　半精度浮動小数点形式とパラメータ値

パラメータ	半精度
Emax	+15
Emin	–14
指数部のバイアス値	+15
指数部の長さ（ビット数）	5
整数ビット	見えない
仮数部の長さ（ビット数）	10
形式の長さ（ビット数）	16

半精度浮動小数点形式で表現できる最小値、最大値を**表 6.5** に示します。

表 6.5　半精度浮動小数点の最大値、最小値

タイプ	値
半精度浮動小数点の最小値	5.96046e – 8
半精度浮動小数点の最小値（ノーマル）	6.10352e – 5
半精度浮動小数点の最大値	65504

6.1.3　レジスタ・セット

FPU は CPU の汎用レジスタ（r0-r31）を使用します。浮動小数点演算専用のレジスタ・ファイルはありません。

- 単精度浮動小数点演算命令：
  32 個の 32 ビット・レジスタを指定できます。これは汎用レジスタの r0-r31 に相当します。

(1) 浮動小数点システム・レジスタ

FPU では 6 個のシステム・レジスタが使用できます。

- FPSR：例外の制御と監視を行います。また、比較演算の結果を保持し、FPU の動作モードを設定します。条件コード、例外モード、サブノーマル数フラッシュ許可、丸めモード制御、原因、例外許可、保存の各ビットがあります。
- FPEPC：浮動小数点演算例外が発生した命令のプログラム・カウンタが格納されます。
- FPST：FPSR レジスタのなかで演算ステータスにかかわるビットの内容を示します。
- FPCC：FPSR.CC(7:0) ビットと同一の内容を示します。
- FPCFG：FPSR レジスタのなかで演算設定にかかわるビットの内容を示します。
- FPEC：FPI 例外の保留状態の確認、取り下げ等の制御を行います。

浮動小数点システム・レジスタについては、**「3.4　FPU 機能レジスタ」**を参照してください。

6.1.4　浮動小数点演算命令

浮動小数点演算命令は、単精度 (Single) の浮動小数点演算を実行します。

浮動小数点演算命令の詳細は**「7.4　浮動小数点演算命令」**を参照してください。

6.1.5　浮動小数点演算例外

この章では、FPU が浮動小数点演算例外をどのように処理するかを説明します。

(1) 例外の種類

通常の方法で浮動小数点演算または演算結果を処理できなくなると、浮動小数点演算例外が発生します。

浮動小数点演算例外発生時の動作には、次の 2 通りがあります。

- 例外許可の場合
  浮動小数点設定／状態レジスタ FPSR の原因ビットをセットし、例外ハンドラ・ルーチンに処理（ソフトウェア処理）を移行します。
- 例外禁止の場合
  浮動小数点設定／状態レジスタ FPSR の保存ビットをセットし、FPU のデスティネーション・レジスタに適切な値（デフォルト値）を格納し、実行を継続します。

FPU は、次の 5 種類の IEEE754 例外を、原因ビット、許可ビット、保存ビット（ステータス・フラグ）によってサポートします。

- 不正確演算（I）
- オーバフロー（O）
- アンダフロー（U）
- ゼロ除算（Z）
- 無効演算（V）

また 6 番目の例外原因として、未実装演算（E）があり、浮動小数点演算を実行できないときに発生します。この例外は、ソフトウェアによる処理を必要とします。未実装演算例外（E）はその性質上、許可ビット、保存ビットはなく、常に例外が許可され、発生します。

図 6.5 に、例外をサポートするために使用する FPSR レジスタのビットを示します。

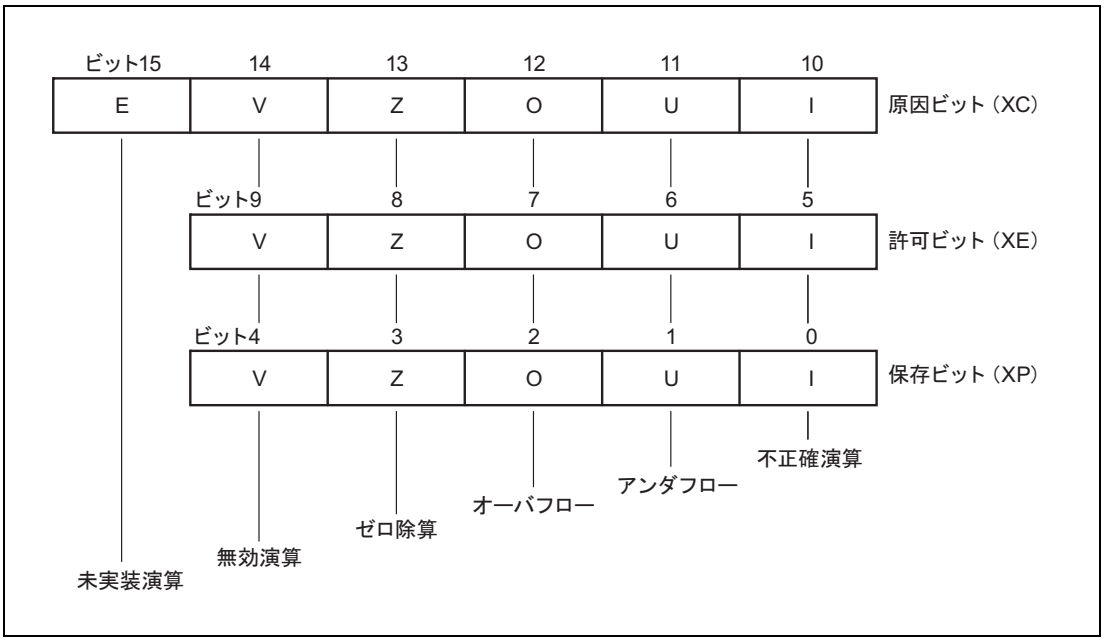


図 6.5　FPSR レジスタの原因／許可／保存ビット

IEEE754 の 5 つの例外（V, Z, O, U, I）は、許可ビットをセットすることにより許可されます。例外が発生し、対応する許可ビットがセットされていれば、FPU は対応する原因ビットをセットし、例外が受け付け可能な状態であれば、例外ハンドラ・ルーチンに移行します。例外の発生が禁止されている場合、その例外に対応する保存ビットがセットされ例外ハンドラ・ルーチンには移行しません。

(2) 例外処理

浮動小数点演算例外が発生すると、FPSR レジスタの原因ビットは、浮動小数点演算例外の発生した原因を示します。

(a) ステータス・フラグ

各 IEEE754 例外に対して、対応する保存ビットが用意されています。保存ビットは、対応する例外の発生が禁止されていて、かつ例外の条件が検出されたときにセットされます。保存ビットは、LDSR 命令で FPSR レジスタに新しい値を書き込むことによってセット／リセットできます。

許可ビットによって例外が禁止されている場合、FPU により既定の処理が行われます。この処理では、浮動小数点演算の結果の代わりに、デフォルト値を結果として与えます。このデフォルト値は例外の種類により決まっています。オーバフロー例外とアンダフロー例外の場合には、そのときの丸めモードにより異なります。**表 6.6** に、それぞれの FPU の IEEE754 例外によって与えられるデフォルト値を示します。

表 6.6　FPU の IEEE754 例外のデフォルト値

領域	説明	丸めモード	デフォルト値
V	無効演算	—	Quiet Not a Number（Q-NaN）を使用します。
Z	ゼロ除算	—	正しい符号付き∞を使用します。
O	オーバフロー	RN	中間結果の符号を付けた∞
		RZ	中間結果の符号を付けた最大ノーマル数
		RP	負のオーバフローの場合：負の最大ノーマル数 正のオーバフローの場合：＋∞
		RM	正のオーバフローの場合：正の最大ノーマル数 負のオーバフローの場合：－∞
U	アンダフロー注 1	RN 注 2	中間結果の符号を付けた 0
		RZ	中間結果の符号を付けた 0
		RP	正のアンダフローの場合：正の最小ノーマル数 負のアンダフローの場合：0
		RM	負のアンダフローの場合：負の最小ノーマル数 正のアンダフローの場合：0
I	不正確演算	—	丸められた結果を使用します。

注 1.　FPSR.FS ビットがクリアされている場合は、丸めた結果がアンダフローすると未実装演算例外（E）を発生するため、アンダフロー例外（U）は発生しません。FPSR.FS ビットがセットされている場合にはフラッシュされた結果がデフォルト値として使用されます。

注 2.　丸めモードが RN かつ FPSR.FN がセットされているときには、より精度の高い方向へフラッシュされます。詳細は**「6.1.11　近傍へのフラッシュ」**を参照してください。

6.1.6　例外の詳細

次に、FPU の各例外の発生条件と FPU での対応について説明します。

(1) 不正確演算例外（I）

次のような場合、FPU は不正確演算例外を検出します。

- 丸めた結果が精度落ちした場合
- 丸めた結果がオーバフローし、かつ、オーバフロー例外が禁止状態の場合
- 丸めた結果がアンダフローし、かつ、アンダフロー例外が禁止状態の場合
- オペランドのサブノーマル数がフラッシュされた場合で、無効演算例外（V）、ゼロ除算例外（Z）が検出されず、かつ他のオペランドが Q-NaN でない場合。

注　意

FPSR.FS ビットがクリアされている場合は、演算結果がアンダフローすると未実装演算例外（E）を発生します。この場合にはアンダフロー例外を検出しないので、不正確演算例外が検出されることはありません。

(a) 例外が許可されている場合

デスティネーション・レジスタの内容は変更せず、ソース・レジスタの内容は保存し、不正確演算例外が発生します。

(b) 例外が許可されていない場合

ほかの例外が発生しない場合、丸められた結果かアンダフロー／オーバフローした結果を、デスティネーション・レジスタに格納します。

(2) 無効演算例外（V）

オペランドの片方または両方が無効の場合、無効演算例外を検出します。

- オペランドに S-NaN を含む算術演算。条件付き転送命令（CMOV）、絶対値（ABS）、算術否定（NEG）は算術演算として扱いませんが、最小値（MIN）、最大値（MAX）は算術演算として扱います。
- 乗算：±0 × ± ∞または± ∞ × ± 0。
- 積和：（±0 × ± ∞）＋ c または（± ∞ × ±0）＋ c 。ただし c が Q-NaN の場合を除く。
- 加減算・積和[注 1]：無限大同士の異符号加算または同符号減算。
- 除算：±0 ÷ ±0 または± ∞ ÷ ± ∞。
- 平方根：オペランドが 0 より小さい。
- ソースが整数範囲外の場合の整数への変換。
- 比較 : 条件コード 8 ～ 15 で、オペランドが Unorderd の場合（**「表 7.8　条件コードのビット定義と論理反転」**参照）

注 1. 乗算の結果が無限大で、無限大同士の加減算となった場合。

(a) **例外が許可されている場合**

デスティネーション・レジスタの内容は変更せず、ソース・レジスタの内容は保持し、無効演算例外が発生します。

(b) **例外が許可されていない場合**

ほかの例外が発生しない場合、デスティネーションが浮動小数点形式であれば、Q-NaN がデスティネーション・レジスタに格納されます。デスティネーションが整数形式の場合に、デスティネーション・レジスタに格納される値については、各命令の演算結果の説明を参照してください。

(3) ゼロ除算例外（Z）

除数が 0 で被除数が 0 以外の有限数のとき、ゼロ除算例外を検出します。

(a) **例外が許可されている場合**

デスティネーション・レジスタの内容は変更せず、ソース・レジスタの内容は保存し、ゼロ除算例外が発生します。

(b) **例外が許可されていない場合**

ほかの例外が発生しない場合、正しい符号のついた無限大数（± ∞）がデスティネーション・レジスタに格納されます。

(4) オーバフロー例外（O）

オーバフロー例外は、指数範囲が無限のとき、丸められた浮動小数点の結果の大きさがデスティネーション形式の最大有限数よりも大きい場合に検出します。

(a) **例外が許可されている場合**

デスティネーション・レジスタの内容は変更せず、ソース・レジスタの内容は保存し、オーバフロー例外が発生します。

(b) **例外が許可されていない場合**

ほかの例外が発生しない場合、丸めモードと中間結果の符号によって決まるデフォルト値がデスティネーション・レジスタに格納されます（**「表 6.6　FPU の IEEE754 例外のデフォルト値」**参照）。

(5) アンダフロー例外（U）

演算結果が 0 以外で $\pm 2^{Emin}$ 内のとき、アンダフロー例外を検出します。

IEEE754 では、アンダフローを検出する方法が複数用意されていますが、どの処理の場合にも、同じ方法で検出するように規定しています。

2 進浮動小数点では、次の 2 つのアンダフロー検出方法が定められています。

- 丸め後、指数範囲を無限として計算された結果が、0 以外で $\pm 2^{Emin}$ 内のとき
- 丸め前、指数範囲と精度を無限として計算された結果が、0 以外で $\pm 2^{Emin}$ 内のとき

本 CPU では、丸めの前にアンダフローを検出します。

また、丸めた結果が以下の場合には不正確結果を検出します。

- 与えられた結果と、指数範囲と精度が無限のとき計算された結果が異なる場合

本 CPU では、不正確結果の検出に関してアンダフロー例外が許可されている場合と許可されていない場合で以下の動作を行います。

(a) 例外が許可されている場合

FPSR.FS ビットがセットされていて、アンダフロー例外が許可されている場合、アンダフロー例外（U）が発生します。FPSR.FS ビットがセットされていて、アンダフロー例外が許可されていない場合、不正確演算例外が許可されていれば、不正確演算例外（I）が発生します

(b) 例外が許可されていない場合

FPSR.FS ビットがセットされている場合、丸めモードと中間結果の値で決まるデフォルト値がデスティネーション・レジスタに格納されます（**「表 6.6　FPU の IEEE754 例外のデフォルト値」**参照）。

注　意

FPSR.FS ビットがセットされていない場合には、例外が許可されているかどうかにかかわらず、必ず未実装演算例外（E）が発生するため、アンダフロー例外（U）は発生しません。

(6) 未実装演算例外（E）

ハードウェアが適切に処理できない異常なオペランド、または異常な結果を検出した場合、E ビットをセットし、未実装演算例外（E）が発生します。オペランドとデスティネーション・レジスタの内容は変更しません。

FPSR.FS ビットがセットされている場合、未実装演算例外（E）が発生することはありません。

FPSR.FS ビットがクリアされている場合、次の条件で発生します。
（CMOVF.S, CMPF.S, ABSF.S, MAXF.S, MINF.S, NEGF.S, CVTF.HS 命令を除く）

- オペランドがサブノーマル数のとき
- 演算結果がサブノーマル数、またはアンダフローしたとき

注　意

1. **未実装演算例外（E）が発生した場合の処理については、「6.1.10　浮動小数点演算モデルの選択」を参照してください。**
2. **FPSR レジスタの FS ビットをセット（1）している場合、未実装演算例外（E）はいかなる場合においても発生しません。**

6.1.7　プレサイス例外とインプレサイス例外

浮動小数点の例外において、プレサイスに例外を発生させるか、インプレサイスに例外を発生させるかを指定することができます。

デフォルトではインプレサイスに例外を発生します。プレサイスな例外を発生するためには、例外モードを変更する必要があります。

FPSR.PEM ビットをセットすることによりプレサイス例外モードを指定します。

(1) プレサイス例外

プレサイス例外が指定されると、CPU は実行を開始した浮動小数点演算命令が完了するまで後続のすべての命令の実行を開始しません。したがって、例外が発生した場合に、ソフトウェアによるエミュレーションのあと、プログラムを続行することが可能です。

浮動小数点演算例外が発生した命令のプログラム・カウンタが EIPC レジスタおよび FPEPC レジスタに格納されます。エミュレーション処理からの復帰は、EIRET 命令により行われます。プレサイス例外モードで発生した浮動小数点演算例外は、PSW の ID ビットや NP ビットの状態にかかわらず、ただちに受け付けられます。

(2) インプレサイス例外

インプレサイス例外が指定されると、CPU は実行を開始した浮動小数点演算命令が完了する前に、後続の命令の実行を開始します。このため例外発生時には、後続の命令が投機的に実行されているので、例外が発生した場合、エミュレーションは困難となりますが、命令実行のスループットを大幅に引き上げることが可能になります。

インプレサイス例外で実行された浮動小数点演算命令が浮動小数点演算例外を発生すると、例外が受け付けられ例外ハンドラ・ルーチンへ移行するまでの間、後続の浮動小数点演算命令（TRFSR 命令を除く）の結果は汎用レジスタに反映されず、また浮動小数点演算例外も発生しません。これを命令の無効化と呼びます。

後続命令を実行する前にインプレサイス浮動小数点演算例外を受け付けたい場合、SYNCE 命令により例外を発生した命令の完了を待ち合わせることができます。

浮動小数点演算例外が発生した命令のプログラム・カウンタが FPEPC レジスタに、例外が受け付けられ中断された命令のプログラム・カウンタが EIPC レジスタに格納されます。

インプレサイス例外モードで発生した浮動小数点演算例外は、PSW の ID ビットが 1、または NP ビットが 1 のときに保留されます。この場合、LDSR 命令を使用して PSW.NP ビットと ID ビットを 0 にすると、保留していた例外が受け付けられます。

6.1.8　状態の退避と復帰

浮動小数点演算例外が発生すると、PC および PSW は EIPC, EIPSW レジスタへ退避され、例外要因コードが EIIC レジスタに格納されます。

浮動小数点演算例外の例外要因コードは、プレサイス例外では 71_H、インプレサイス例外では 72_H です。

浮動小数点演算例外の処理中に EI レベル例外が受け付けられると、EIPC などがオーバライドされて浮動小数点演算例外発生命令へ復帰できなくなるため、EI レベル例外の受け付けを許可する必要がある場合は、必ず、先に EIPC, EIPSW, EIIC レジスタの内容をスタックなどに退避させてください。

浮動小数点演算例外ハンドラ・ルーチン中で浮動小数点演算命令を使用する場合、さらに浮動小数点演算例外が発生することで FPSR, FPEPC レジスタがオーバライドされることがあります。このような場合には、浮動小数点演算例外ハンドラの先頭で FPSR, FPEPC レジスタを退避させ、ハンドラの最後に復帰させてください。

FPSR レジスタの原因ビットは許可された例外 1 回分の結果だけを保持します。いずれも次に許可された例外が発生するまで前の結果が保持されます。

6.1.9　サブノーマル数のフラッシュ

本 CPU では、最小の正規化数以下の非常に小さい値（サブノーマル数）を取り扱う際に次の 2 通りの対処が可能です。

- オペランドまたは演算結果を正規化し、演算を継続する
- 未実装演算例外（E）を発生させ、例外処理を行う

より厳密に正確な解を得るために例外処理によってソフトウェア処理を行う必要があります。しかし、厳密に正確な解を得るために必要な処理時間は演算の入力値に依存するため、リアルタイム性の確保が重要な制御系システムにおいては、多くの場合受け容れられません。このような場合に計算結果の厳密さを求めず、一定の処理時間で定められた結果を得ることが重要です。

(1) サブノーマル数を正規化し、演算を継続する場合

本 CPU では、FPSR レジスタの FS ビットをセット（1）することによって、オペランド入力や演算結果がサブノーマル数の場合でも、特定の値に正規化し演算を継続することが可能です。このとき、演算結果としては、非常に小さな値の差が現れない場合があります。

FS ビットをセット（1）した場合にサブノーマル数をどのような値にフラッシュするかを、オペランド入力と演算結果に関して**表 6.7**、**表 6.8** に示します。

表 6.7　丸めモードとオペランド入力のフラッシュ値

サブノーマル・オペランドの符号	丸めモードとオペランド入力をフラッシュする値			
	RN	RZ	RP	RM
正	+0			
負	–0			

表 6.8　丸めモードと演算結果のフラッシュ値

サブノーマル演算結果の符号	丸めモードと演算結果をフラッシュする値			
	RN 注 1	RZ	RP	RM
正	+0	+0	$+2^{Emin}$	+0
負	–0	–0	–0	-2^{Emin}

注 1.　丸めモードが RN かつ FPSR.FN がセットされているときにはより精度の高い方向へフラッシュされます。詳細は、**「6.1.11　近傍へのフラッシュ」**を参照してください。

オペランド入力のサブノーマル数フラッシュ発生は、FPSR レジスタの IF ビットを参照することによって確認可能です。演算結果のサブノーマル数フラッシュ発生は、FPSR レジスタの U ビットを参照することによって確認可能です。

注　意

1. **リアルタイム性の確保が重要な制御系システムにおいては、常に FS ビットをセット（1）して利用することを推奨します。**
2. **FPSR レジスタの FS ビットをセット（1）している場合、未実装演算例外はいかなる場合においても発生しません。**
3. **演算結果のサブノーマル数判定は丸め前の値に対して行われます。**
4. **FPSR レジスタの IF ビットは浮動小数点演算例外が発生した命令のフラッシュ発生も蓄積、表示します。**

(2) 未実装演算例外（E）を発生させ、例外処理を行う場合

FPSR レジスタの FS ビットをクリア（0）することによって、オペランド入力や演算結果がサブノーマル数になるような場合では、未実装演算例外（E）が発生します。未実装演算例外による浮動小数点演算例外処理ルーチンにおいて、ソフトウェア処理による漸進的アンダフロー処理を行うことで、より厳密な解を得ることができます。この場合、ソフトウェア処理の負荷によってはリアルタイム性を確保できなくなる可能性があることに注意してください。

注　意

ソフトウェア処理で厳密な解を得るためには、未実装演算例外による浮動小数点演算例外が正確に受け付けられる必要があります。このため、必ず FPSR レジスタの PEM ビットをセット（1）して正確に例外が受け付けられるようにしてください。

(3) サブノーマル数を扱える命令

以下の命令は FPSR.FS = 0 のときオペランド入力がサブノーマル数であっても、未実装演算例外（E）が発生せずサブノーマル数のまま命令実行が可能です。

- 条件付き転送命令（CMOV）、絶対値（ABS）、算術否定（NEG）
- 最小値（MIN）、最大値（MAX）、比較（CMPF）
- 半精度から単精度への変換（CVTF.HS）

(4) サブノーマル数フラッシュの影響を受けない命令

以下の命令は FPSR.FS = 1 のときオペランド入力がサブノーマル数であっても、フラッシュが行われません。

- 条件付き転送命令（CMOV）、絶対値（ABS）、算術否定（NEG）
- 最小値（MIN）、最大値（MAX）、比較（CMPF）
- 半精度から単精度への変換（CVTF.HS）

6.1.10　浮動小数点演算モデルの選択

本 CPU では、浮動小数点演算に要求される処理速度や演算精度によって、3 つの演算モデルを推奨しています。

処理速度が必要な場合は「例外を起こさないモデル」を選択し、浮動小数点演算実行による例外をできるだけ起こさないことで、処理性能を優先させる演算モデルでの実行を行います。リアルタイム性を必要とする用途において、厳密に正確な演算が必須でない場合には、例外発生によるエミュレーション処理のオーバヘッドを排除することができます。

また、例外が起きない限りは処理速度を追求し、例外が起きた場合には例外処理に移行する「不正確（インプレサイス）な例外モデル」が利用可能です。これは前述の処理速度を追求するモデルでの運用を想定している場合に、ソフトウェア開発段階では早期に例外の発生を検出して例外処理に移行することで、例外事象の発生点に近い内部状態を保存することで、デバッグの難易度を下げます。

演算精度をソフトウェアの要求に合わせて適切に管理したい場合は「正確（プレサイス）な例外モデル」を選択し、浮動小数点演算例外を例外事象の検出後、即座に例外処理に移行することでソフトウェアによる支援の元、より厳密な計算を行う演算モデルを選択してください。

(1) 例外を起こさないモデル

処理速度を優先し、例外を起こさないモデルで利用する場合には、次の設定を推奨します。

- FPSR レジスタの許可ビットをクリア（0）して浮動小数点演算例外発生を抑制
- FPSR レジスタの FS ビットをセット（1）してサブノーマル値をフラッシュ
- 精度を必要としない処理では単精度浮動小数点形式を使用

演算処理上、無視できる浮動小数点演算例外は例外発生を禁止することにより、デフォルトの値で演算を続行できます。サブノーマル値のフラッシュは漸進アンダフローが無視できる場合に、フラッシュした値で演算を続行できます。単精度命令は、一般に、実行クロック数（latency）も少なくてすみます。

演算過程での例外事象の発生は、別途ソフトウェアによる明示的な原因フラグの参照によって行ってください。

(2) 不正確な例外モデル

処理速度を優先しつつ、例外を発生させるモデルで利用する場合には、次の設定を推奨します。

- FPSR レジスタの許可ビットを例外処理の必要性に応じて適切に設定
- FPSR レジスタの FS ビットをセット（1）してサブノーマル値をフラッシュ
- FPSR.PEM ビットをクリア（0）して、インプレサイス例外モードを指定

演算処理上、無視できる浮動小数点演算例外は例外発生を禁止することにより、デフォルトの値で演算を続行できます。サブノーマル値のフラッシュは漸進アンダフローが無視できる場合に、フラッシュした値で演算を続行できます。単精度命令は、一般に、実行クロック数（latency）も少なくてすみます。インプレサイス例外モードは演算のスループットがプレサイス例外モードと比べて高くなります。

(3) 正確な例外モデル

演算精度をソフトウェアによって管理するために、正確に例外を発生させるモデルで利用する場合には、次の設定を推奨します。

- FPSR レジスタの許可ビットを例外処理の必要性に応じて適切に設定
- FPSR レジスタの FS ビットをクリア（0）してサブノーマル値で例外を発生させる
- FPSR.PEM ビットをセット（1）して、プレサイス例外モードを指定

この設定によって、例外発生時には正確にその命令で例外が受け付けられます。後続の命令が実行されることはなく、命令実行前のプロセッサ状態が保存されているため、厳密に正確な演算を必要とする場合に、ソフトウェアによってエミュレーションすることが可能です。エミュレートしたオペレーションによってさらに IEEE754 例外が発生した場合、今度はそれらの例外もエミュレートしてください。

FPEPC レジスタを使って命令を検索することによって、例外ハンドラは次のことを判別できます。

- 実行中の命令
- デスティネーションの形式

オーバフロー例外、アンダフロー例外（変換命令を除く）、不正確演算例外が発生したときに正しく丸められた結果を得るには、ソース・レジスタを調べたり、命令をエミュレートするソフトウェアを例外ハンドラ中に用意したりしてください。

無効演算例外やゼロ除算例外が発生した場合や、オーバフロー例外またはアンダフロー例外が浮動小数点変換時に発生した場合、例外ハンドラ中に命令のソース・レジスタを調べることによってオペランドの値を得るようなソフトウェアを用意してください。

IEEE754 においては、可能ならオーバフロー例外およびアンダフロー例外を不正確演算例外に対して優先させることを推奨しています。この優先順位は、ソフトウェアによって設定します。ハードウェアは、オーバフロー例外、アンダフロー例外と不正確演算例外の両方のビットをセットします。

また、FPU では、無効なデータ形式の命令を実行しようとしたとき、または FPSR.FS ビットがクリア（0）された状態でオペランド入力や演算結果がサブノーマル数になるような場合では、未実装演算例外（E）が発生します（一部命令を除く）。この時、オペランドとデスティネーション・レジスタの内容は変更しません。

6.1.11　近傍へのフラッシュ

本 CPU では演算結果のサブノーマル数フラッシュに、より精度の高い近傍へのフラッシュを備えています。近傍へのフラッシュは丸めモードが RN かつ FPSR.FN がセット（1）されている時に有効です。この時、FPU は演算結果の符号だけでなく、演算結果の値を元にフラッシュする値を決定します。ただし、SUBF, FMSF, FNMSF 命令による減算、および ADDF, FMAF, FNMAF 命令による負数の加算を行った演算結果が特定のサブノーマル数 ($\pm 2^{(Emin-2)}$) のときに、**表 6.9** に示した 近傍とは異なる $\pm 2^{Emin}$ にフラッシュします。丸めモードが RN 以外の時、またオペランド入力のフラッシュ結果には影響を受けません。

表 6.9　丸めモードと演算結果をフラッシュする値

サブノーマル演算結果の値	丸めモードと演算結果をフラッシュする値				
	RN		RZ	RP	RM
	FN = 1	FN = 0			
$+2^{Emin-1} \leqq$ 演算結果 $< +2^{Emin}$	$+2^{Emin}$	+ 0	+ 0	$+2^{Emin}$	+ 0
+ 0 < 演算結果 $< +2^{Emin-1}$	+ 0				
$-2^{Emin-1} <$ 演算結果 < –0	–0	–0	–0	–0	-2^{Emin}
$-2^{Emin} <$ 演算結果 $\leqq -2^{Emin-1}$	-2^{Emin}				

注　意

演算結果のサブノーマル数判定は丸め前の値に対して行われます。

第 7 章　命令

7.1　オペコードと命令フォーマット

本 CPU の命令には、基本命令として定義される「CPU 命令」と、用途ごとに定義される「コプロセッサ命令」の 2 種類があります。

7.1.1　CPU 命令

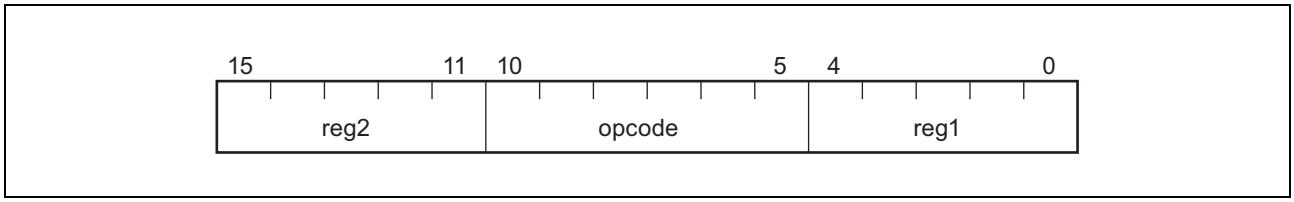
「7.1.2　コプロセッサ命令」で示されるコプロセッサ命令フォーマット以外のオペコード領域は、CPU 命令に分類される命令が配置されます。

CPU 命令は、基本的に 16 ビット長／32 ビット長のフォーマットにしたがって表現されます。また、いくつかの命令は、これらのフォーマットに追加する形で、さらにオプション・データを利用し、48 ビット長／64 ビット長の命令を構成します。詳細は**「7.2.2　基本命令セット」**の各命令のオペコードを参照してください。

このオペコード領域中で、有意な CPU 命令が定義されていないオペコードは、予約命令として将来の機能拡張のために予約されています。詳細は**「7.1.3　予約命令」**を参照してください。

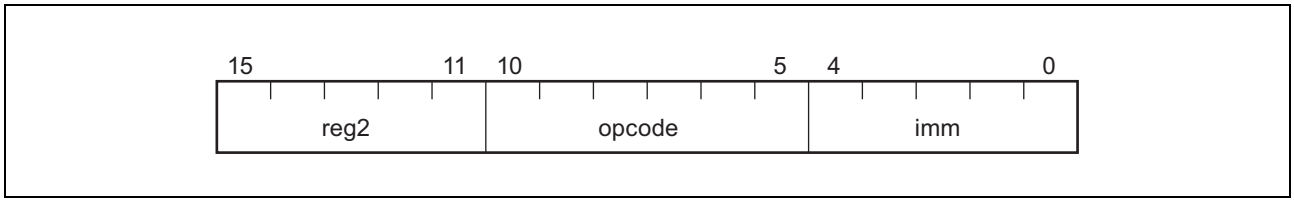
(1) reg-reg 命令形式（Format I）

6 ビットのオペコード・フィールド、2 つの汎用レジスタ指定フィールドを持つ 16 ビット長命令形式。



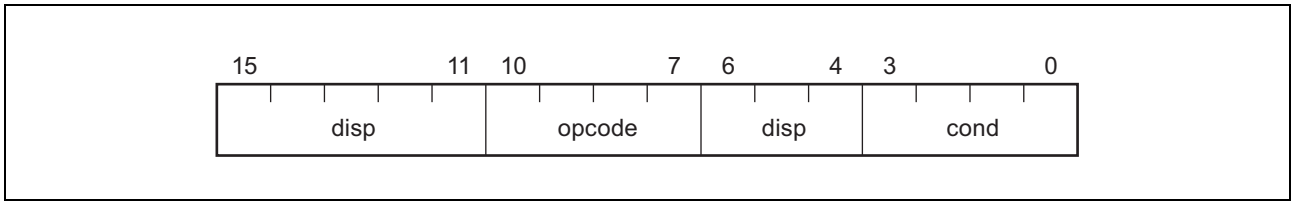
(2) imm-reg 命令形式（Format II）

6 ビットのオペコード・フィールド、5 ビットのイミディエイト・フィールド、1 つの汎用レジスタ・フィールドを持つ 16 ビット長命令形式。



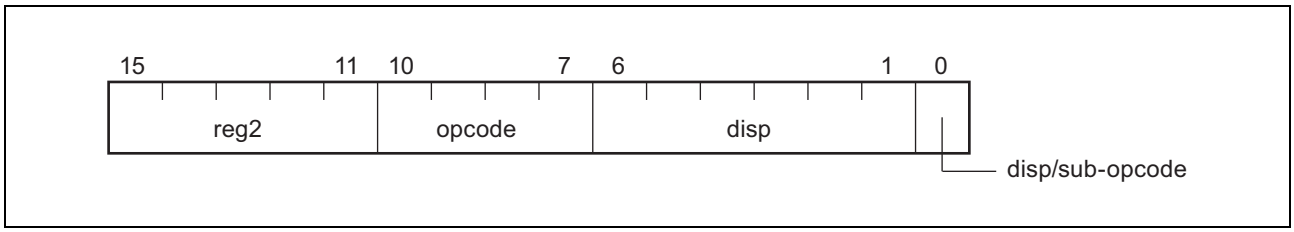
(3) 条件分岐命令形式（Format III）

4 ビットのオペコード・フィールド、4 ビットの条件コード・フィールド、8 ビットのディスプレースメント・フィールドを持つ 16 ビット長命令形式。

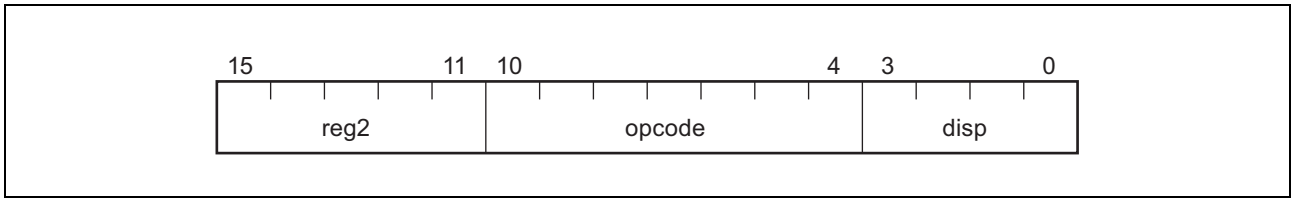


(4) ロード／ストア命令 16 ビット形式（Format IV）

4 ビットのオペコード・フィールド、1 つの汎用レジスタ指定フィールド、7 ビットのディスプレースメント・フィールド（または 6 ビット・ディスプレースメント・フィールドと 1 ビット・サブオペコード・フィールド）を持つ 16 ビット長命令形式。

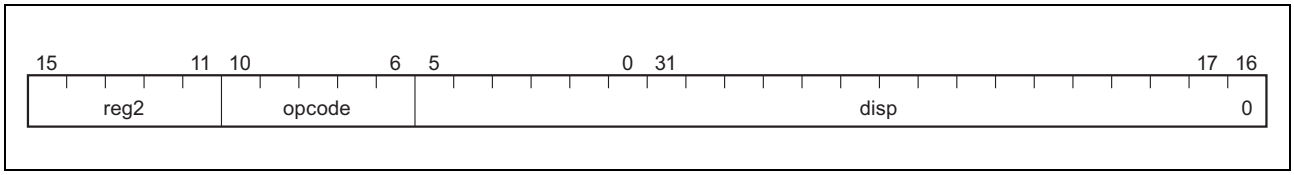


または、7 ビットのオペコード・フィールドと 1 つの汎用レジスタ指定フィールド、4 ビットのディスプレースメント・フィールドを持つ 16 ビット長命令形式。



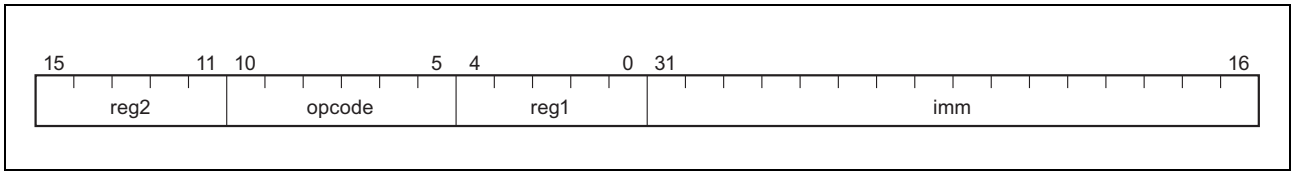
(5) ジャンプ命令形式（Format V）

5 ビットのオペコード・フィールド、1 つの汎用レジスタ指定フィールド、22 ビットのディスプレースメント・フィールドを持つ 32 ビット長命令形式。



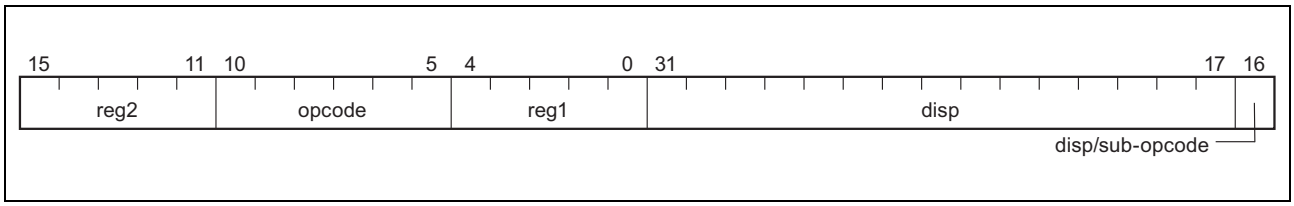
(6) 3 オペランド命令形式（Format VI）

6 ビットのオペコード・フィールド、2 つの汎用レジスタ指定フィールド、16 ビットのイミディエイト・フィールドを持つ 32 ビット長命令形式。



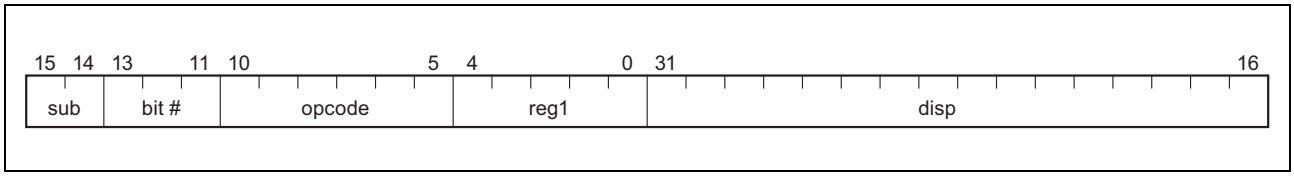
(7) ロード／ストア命令 32 ビット形式（Format VII）

6 ビットのオペコード・フィールド、2 つの汎用レジスタ指定フィールド、16 ビットのディスプレースメント・フィールド（または 15 ビットのディスプレースメント・フィールドと 1 ビット・サブオペコード・フィールド）を持つ 32 ビット長命令形式。



(8) ビット操作命令形式（Format VIII）

6 ビットのオペコード・フィールドと 2 ビットのサブオペコード・フィールド、3 ビットのビット指定フィールド、1 つの汎用レジスタ指定フィールド、16 ビットのディスプレースメント・フィールドを持つ 32 ビット長命令形式。

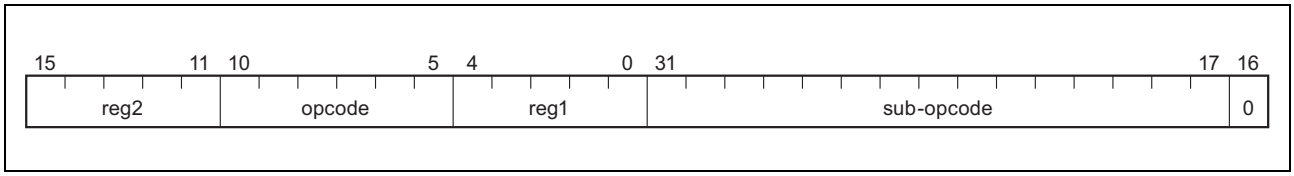


(9) 拡張命令形式 1（Format IX）

6 ビットのオペコード・フィールドと 2 つの汎用レジスタ指定フィールドを持ち、それ以外のビットはサブオペコード・フィールドとして取り扱う 32 ビット長命令形式。

注　意

拡張命令形式 1 では、汎用レジスタ指定フィールド、またはサブオペコード・フィールドの一部をシステム・レジスタ番号フィールド、条件コード・フィールド、イミディエイト・フィールド、ディスプレースメント・フィールドとして取り扱う場合があります。詳細は「7.2.2　基本命令セット」の各命令の説明を参照してください。

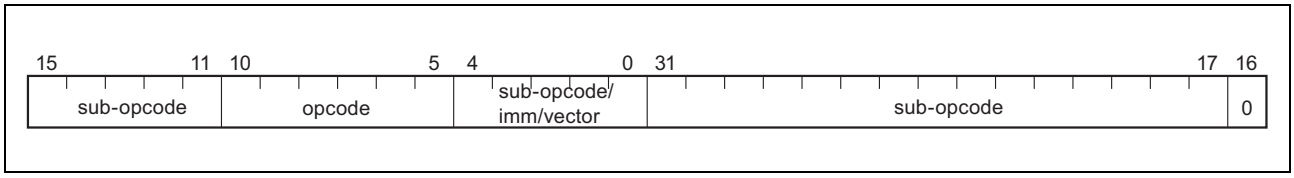


(10) 拡張命令形式 2（Format X）

6 ビットのオペコード・フィールドを持ち、それ以外のビットをサブオペコード・フィールドとして取り扱う 32 ビット長命令形式。

注　意

拡張命令形式 2 では、汎用レジスタ指定フィールド、またはサブオペコード・フィールドの一部をシステム・レジスタ番号フィールド、条件コード・フィールド、イミディエイト・フィールド、ディスプレースメント・フィールドとして取り扱う場合があります。詳細は**「7.2.2　基本命令セット」**の各命令の説明を参照してください。

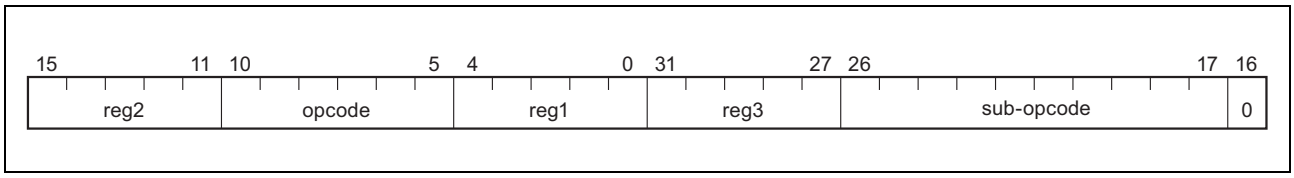


(11) 拡張命令形式 3（Format XI）

6 ビットのオペコード・フィールドと、3 つの汎用レジスタ指定フィールドを持ち、それ以外のビットをサブオペコード・フィールドとして取り扱う 32 ビット長命令形式。

注　意

拡張命令形式 3 では、汎用レジスタ指定フィールド、またはサブオペコード・フィールドの一部をシステム・レジスタ番号フィールド、条件コード・フィールド、イミディエイト・フィールド、ディスプレースメント・フィールドとして取り扱う場合があります。詳細は**「7.2.2　基本命令セット」**の各命令の説明を参照してください。

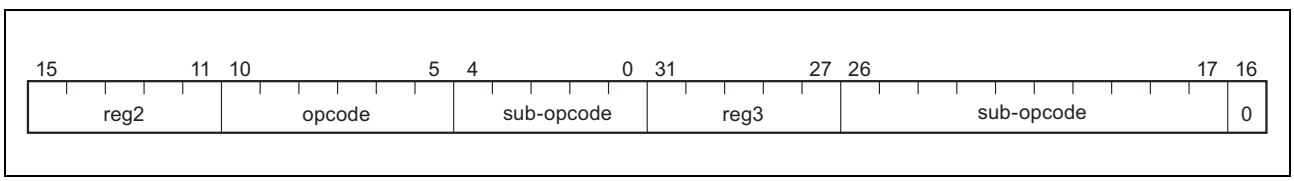


(12) 拡張命令形式 4（Format XII）

6 ビットのオペコード・フィールド、2 つの汎用レジスタ指定フィールドを持ち、それ以外のビットをサブオペコード・フィールドとして取り扱う 32 ビット長命令形式。

注　意

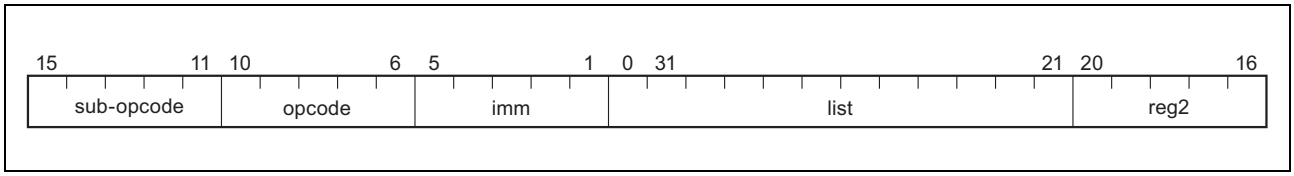
拡張命令形式 4 では、汎用レジスタ指定フィールド、またはサブオペコード・フィールドの一部をシステム・レジスタ番号フィールド、条件コード・フィールド、イミディエイト・フィールド、ディスプレースメント・フィールドとして取り扱う場合があります。詳細は**「7.2.2　基本命令セット」**の各命令の説明を参照してください。



(13) スタック操作命令形式（Format XIII）

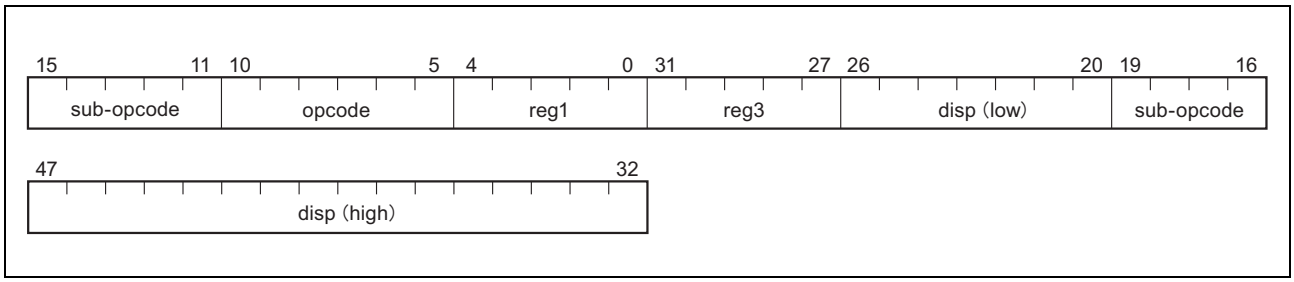
5 ビットのオペコード・フィールドと 5 ビットのイミディエイト・フィールド、12 ビットのレジスタ・リスト・フィールド、5 ビットのサブオペコード・フィールド、1 つの汎用レジスタ指定フィールド（または 5 ビットのサブオペコード・フィールド）を持つ 32 ビット長命令形式。

汎用レジスタ指定フィールドは、命令の形式によっては、サブオペコード・フィールドとして取り扱います。



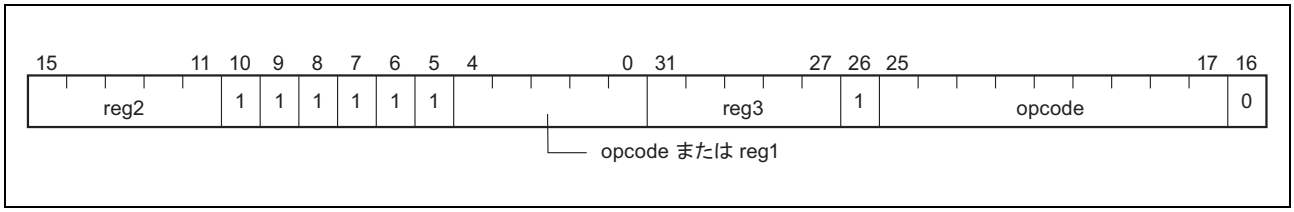
(14) ロード／ストア命令 48 ビット形式（Format XIV）

6 ビットのオペコード・フィールドと、2 つの汎用レジスタ指定フィールド、23 ビットのディスプレースメント・フィールドを持ち、それ以外のビットをサブオペコード・フィールドとして取り扱う 48 ビット長命令形式。



7.1.2　コプロセッサ命令

次のフォーマットにしたがう命令は、コプロセッサ命令として定義されます。



コプロセッサ命令は、それぞれのコプロセッサ機能で定義されます。

(1) コプロセッサ使用不可例外

コプロセッサ命令と定義されたオペコードに対して、製品上で搭載されていない、あるいは動作状態によって、使用が許可されていない場合に、これらのコプロセッサ命令を実行しようとした場合、ただちにコプロセッサ使用不可例外（UCPOP）が発生します。

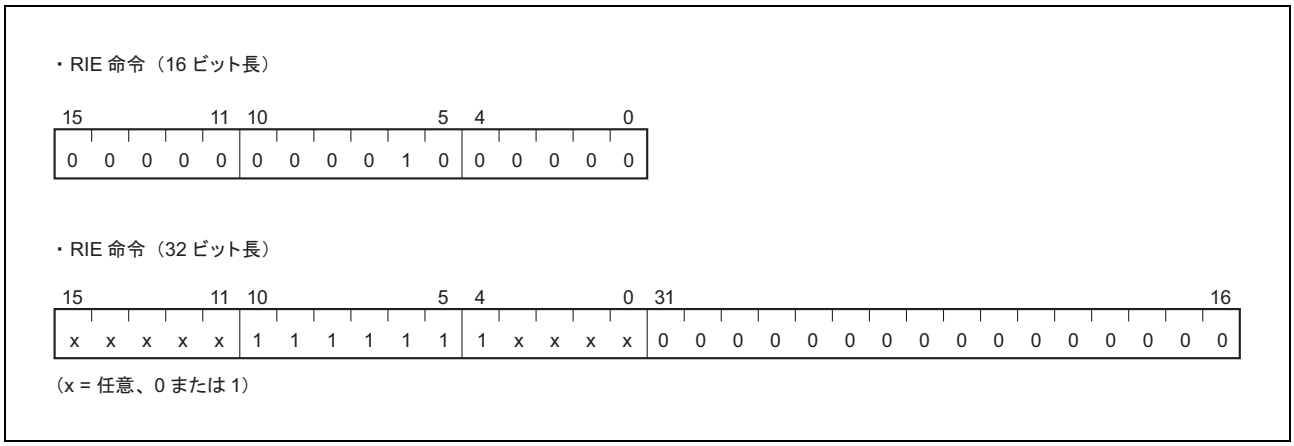
詳細は、**「2.4.3　コプロセッサ使用不可例外」**を参照してください。

7.1.3　予約命令

将来の機能拡張のため予約され、命令が定義されていないオペコードは予約命令として定義されています。予約命令のオペコードに対しては、次の 2 種類の動作のいずれを行うことがハードウェア仕様によって定義されます。

- 予約命令例外が発生する。
- いずれかの命令として動作する。

また、次のオペコードは本 CPU において、常に予約命令例外が発生する RIE 命令として定義されています。



7.2　基本命令

7.2.1　基本命令の概要

(1) ロード命令

メモリからレジスタへのデータ転送を行います。次の命令（ニーモニック）があります。

(a) LD 命令

- LD.B:　Load byte
- LD.BU:　Load byte unsigned
- LD.DW:　Load Double word
- LD.H:　Load half-word
- LD.HU:　Load half-word unsigned
- LD.W:　Load word

(b) SLD 命令

- SLD.B:　Short format load byte
- SLD.BU:　Short format load byte unsigned
- SLD.H:　Short format load half-word
- SLD.HU:　Short format load half-word unsigned
- SLD.W:　Short format load word

(2) ストア命令

レジスタからメモリへのデータ転送を行います。次の命令（ニーモニック）があります。

(a) ST 命令

- ST.B:　Store byte
- ST.DW:　Store double word
- ST.H:　Store half-word
- ST.W:　Store word

(b) SST 命令

- SST.B:　Short format store byte
- SST.H:　Short format store half-word
- SST.W:　Short format store word

(3) 乗算命令

内蔵のハードウェア乗算器により、1 クロックでの乗算処理を行います。次の命令（ニーモニック）があります。

- MUL:　Multiply word
- MULH:　Multiply half-word
- MULHI:　Multiply half-word immediate
- MULU:　Multiply word unsigned

(4) 加算付き乗算命令

乗算後、その結果に対する加算を行います。次の命令（ニーモニック）があります。

- MAC: Multiply and add word
- MACU: Multiply and add word unsigned

(5) 算術演算命令

加減算、レジスタ間のデータ転送、データ比較を行います。次の命令（ニーモニック）があります。

- ADD: Add
- ADDI: Add immediate
- CMP: Compare
- MOV: Move
- MOVEA: Move effective address
- MOVHI: Move high half-word
- SUB: Subtract
- SUBR: Subtract reverse

(6) 条件付き演算命令

指定された条件に応じた加減算を行います。次の命令（ニーモニック）があります。

- ADF: Add on condition flag
- SBF: Subtract on condition flag

(7) 飽和演算命令

飽和加減算を行います。なお、演算の結果が正の最大値（7FFF FFFF$_H$）を越えたときは 7FFF FFFF$_H$ を、負の最大値（8000 0000$_H$）を越えたときは 8000 0000$_H$ を返します。次の命令（ニーモニック）があります。

- SATADD: Saturated add
- SATSUB: Saturated subtract
- SATSUBI: Saturated subtract immediate
- SATSUBR: Saturated subtract reverse

(8) 論理演算命令

論理演算を行います。次の命令（ニーモニック）があります。

- AND:　AND
- ANDI:　AND immediate
- NOT:　NOT
- OR:　OR
- ORI:　OR immediate
- TST:　Test
- XOR:　Exclusive OR
- XORI:　Exclusive OR immediate

(9) データ操作命令

データ操作とシフト命令があります。シフト命令には、算術シフトと論理シフトがあります。内蔵のバレル・シフタにより、1 クロックで複数ビットのシフトを行います。次の命令（ニーモニック）があります。

- BINS:　Bitfield Insert
- BSH:　Byte swap half-word
- BSW:　Byte swap word
- CMOV:　Conditional move
- HSH:　Half-word swap half-word
- HSW:　Half-word swap word
- ROTL:　Rotate left
- SAR:　Shift arithmetic right
- SASF:　Shift and set flag condition
- SETF:　Set flag condition
- SHL:　Shift logical left
- SHR:　Shift logical right
- SXB:　Sign extend byte
- SXH:　Sign extend half-word
- ZXB:　Zero extend byte
- ZXH:　Zero extend half-word

(10) ビット・サーチ命令

レジスタに格納されたデータから指定のビットを検索します。

- SCH0L:　Search zero from left
- SCH0R:　Search zero from right
- SCH1L:　Search one from left
- SCH1R:　Search one from right

(11) 除算命令

除算を行います。レジスタに格納された値にかかわらず、常に一定のステップ数で演算を実行します。次の命令（ニーモニック）があります。

- DIV:　Divide word
- DIVH:　Divide half-word
- DIVHU:　Divide half-word unsigned
- DIVU:　Divide word unsigned

(12) 高速除算命令

除算を行います。レジスタに格納された値から、あらかじめ商の有効桁数を判断し、必要最小なステップで演算を実行します。次の命令（ニーモニック）があります。

- DIVQ:　Divide word quickly
- DIVQU:　Divide word unsigned quickly

(13) 分岐命令

無条件分岐命令（JARL, JMP, JR）とフラグの状態により制御を変更する条件分岐命令（Bcond）があります。分岐命令により指定されたアドレスにプログラムの制御を移します。次の命令（ニーモニック）があります。

- Bcond（BC, BE, BGE, BGT, BH, BL, BLE, BLT, BN, BNC, BNE, BNH, BNL, BNV, BNZ, BP, BR, BSA, BV, BZ）: Branch on condition code
- JARL:　Jump and register link
- JMP:　Jump register
- JR:　Jump relative

(14) ループ命令

- LOOP:　Loop

(15) ビット操作命令

メモリのビット・データに対して、論理演算を行います。指定されたビット以外は影響を受けません。次の命令（ニーモニック）があります。

- CLR1:　Clear bit
- NOT1:　Not bit
- SET1:　Set bit
- TST1:　Test bit

(16) 特殊命令

前項までのカテゴリに含まれない命令です。次の命令（ニーモニック）があります。

- CALLT:　Call with table look up
- CAXI:　Compare and exchange for interlock
- CLL:　Clear Load Link
- CTRET:　Return from CALLT
- DI:　Disable interrupt
- DISPOSE:　Function dispose
- EI:　Enable interrupt
- EIRET:　Return from trap or interrupt
- FERET:　Return from trap or interrupt
- FETRAP:　Software Trap
- HALT:　Halt
- LDSR:　Load system register
- LDL.W:　Load linked word
- NOP:　No operation
- POPSP:　Pop registers from Stack
- PREPARE:　Function prepare
- PUSHSP:　Push registers from Stack
- RIE:　Reserved instruction exception
- SNOOZE:　Snooze
- STSR:　Store system register
- STC.W:　Store conditional word
- SWITCH:　Jump with table look up
- SYNCE:　Synchronize exceptions
- SYNCI:　Synchronize memory for Instruction Writers
- SYNCM:　Synchronize memory
- SYNCP:　Synchronize pipeline
- SYSCALL:　System call
- TRAP:　Trap

7.2.2　基本命令セット

この節では、各命令のニーモニックごとに（アルファベット順）、次の項目に分けて説明します。

- 命令形式 :
  命令の記述方法、オペランドを示します（略号については、**表 7.1** 参照）。
- オペレーション :
  命令の機能を示します（略号については、**表 7.2** 参照）。
- フォーマット :
  命令形式を命令フォーマットで示します（**「7.1　オペコードと命令フォーマット」**参照）。
- オペコード :
  命令のオペコードをビット・フィールドで示します（略号については、**表 7.3** 参照）。
- フラグ :
  命令実行により変化する PSW（プログラム・ステータス・ワード）の各フラグの動作を示します。「0」はクリア（リセット）を、「1」はセットを、「—」は変化しないことを示します。
- 説明 : 命令の動作説明をします。
- 補足 : 命令の補足説明をします。
- 注意 : 注意事項を示します。

表 7.1　命令形式の凡例

略号	意味
reg1	汎用レジスタ（ソース・レジスタとして使用）
reg2	汎用レジスタ（主にデスティネーション・レジスタとして使用。一部の命令で、ソース・レジスタとしても使用）
reg3	汎用レジスタ（主に除算結果の余り、乗算結果の上位 32 ビットを格納）
bit#3	ビット・ナンバ指定用 3 ビット・データ
imm ×	× ビット・イミディエイト・データ
disp ×	× ビット・ディスプレースメント・データ
regID	システム・レジスタ番号
selID	システム・レジスタのグループ番号
vector ×	ベクタを指定するデータ（× はビット・サイズをあらわします）
cond	条件名を示します（**「表 7.4　条件コード一覧」**参照）
cccc	条件コードを示す 4 ビット・データ（**「表 7.4　条件コード一覧」**参照）
sp	スタック・ポインタ（r3）
ep	エレメント・ポインタ（r30）
list12	レジスタ・リスト
rh-rt	rh で示される汎用レジスタから、rt で示される汎用レジスタまでの複数の汎用レジスタ

表 7.2　オペレーションの凡例

略号	意味
←	代入
GR[a]	汎用レジスタ「a」の格納値
SR[a, b]	システム・レジスタ（RegID =「a」, SelID =「b」）の格納値
(n:m)	ビット選択。ビット「n」からビット「m」までを選択する。
zero-extend (n)	n を、ワード長までゼロ拡張する。
sign-extend (n)	n を、ワード長まで符号拡張する。
load-memory (a, b)	アドレス「a」から、サイズ「b」のデータを読み出す。
store-memory (a, b, c)	アドレス「a」にデータ「b」をサイズ「c」で書き込む。
extract-bit (a, b)	データ「a」のビット・ナンバ「b」の値を取り出す。
set-bit (a, b)	データ「a」のビット・ナンバ「b」の値をセットする。
not-bit (a, b)	データ「a」のビット・ナンバ「b」の値を反転する。
clear-bit (a, b)	データ「a」のビット・ナンバ「b」の値をクリアする。
saturated (n)	n の飽和処理を行う。 計算の結果、n ≧ 7FFF FFFF$_H$ となった場合、n = 7FFF FFFF$_H$ とする。 計算の結果、n ≦ 8000 0000$_H$ となった場合、n = 8000 0000$_H$ とする。
result	結果をフラグに反映する。
Byte	バイト（8 ビット）
Half-word	ハーフワード（16 ビット）
Word	ワード（32 ビット）
==	比較（一致で真）
!=	比較（不一致で真）
+	加算
–	減算
||	ビット連結
×	乗算
÷	除算
%	除算結果の余り
AND	論理積
OR	論理和
XOR	排他的論理和
NOT	論理否定
logically shift left by	論理左シフト
logically shift right by	論理右シフト
arithmetically shift right by	算術右シフト

表 7.3　オペコードの凡例

略号	意味
R	reg1 または regID を指定するコードの 1 ビット分データ
r	reg2 を指定するコードの 1 ビット分データ
w	reg3 を指定するコードの 1 ビット分データ
D	ディスプレースメントの 1 ビット分データ（ディスプレースメントの上位ビットを示す）
d	ディスプレースメントの 1 ビット分データ
I	イミディエイトの 1 ビット分データ（イミディエイトの上位ビットを示す）
i	イミディエイトの 1 ビット分データ
V	vector を指定するコードの 1 ビット分データ（vector の上位ビットを示す）
v	vector を指定するコードの 1 ビット分データ
cccc	条件コードを示す 4 ビット・データ（**「表 7.4　条件コード一覧」**参照）
bbb	ビット・ナンバ指定用 3 ビット・データ
L	レジスタ・リスト中の汎用レジスタを指定する 1 ビット分データ
S	レジスタ・リスト中の EIPC/FEPC, EIPSW/FEPSW を指定する 1 ビット分データ
P	レジスタ・リスト中の PSW を指定する 1 ビット分データ

表 7.4　条件コード一覧

条件コード（cccc）	条件名	条件式
0000	V	OV = 1
1000	NV	OV = 0
0001	C/L	CY = 1
1001	NC/NL	CY = 0
0010	Z	Z = 1
1010	NZ	Z = 0
0011	NH	(CY or Z) = 1
1011	H	(CY or Z) = 0
0100	S/N	S = 1
1100	NS/P	S = 0
0101	T	always（無条件）
1101	SA	SAT = 1
0110	LT	(S xor OV) = 1
1110	GE	(S xor OV) = 0
0111	LE	((S xor OV) or Z) = 1
1111	GT	((S xor OV) or Z) = 0

＜算術演算命令＞

ADD

Add register/immediate

加算

【命令形式】　（1）ADD　reg1, reg2

（2）ADD　imm5, reg2

【オペレーション】　（1）GR[reg2] ← GR[reg2] + GR[reg1]

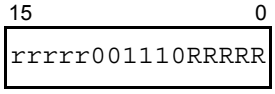
（2）GR[reg2] ← GR[reg2] + sign-extend (imm5)

【フォーマット】　（1）Format I

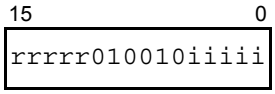
（2）Format II

【オペコード】

(1)



(2)



【フラグ】

CY	MSB からのキャリーがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　（1）汎用レジスタ reg2 のワード・データに汎用レジスタ reg1 のワード・データを加算し、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

（2）汎用レジスタ reg2 のワード・データにワード長まで符号拡張した 5 ビット・イミディエイトを加算し、その結果を汎用レジスタ reg2 に格納します。

＜算術演算命令＞

ADDI

Add immediate

加算

【命令形式】　ADDI imm16, reg1, reg2

【オペレーション】　GR[reg2] ← GR[reg1] + sign-extend (imm16)

【フォーマット】　Format VI

【オペコード】

15	0	31	16
rrrrr110000RRRRR		iiiiiiiiiiiiiiii	

【フラグ】

CY	MSB からのキャリーがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg1 のワード・データにワード長まで符号拡張した 16 ビット・イミディエイトを加算し、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

＜条件付き演算命令＞

ADF

Add on condition flag

条件付き加算

【命令形式】　ADF　cccc, reg1, reg2, reg3

【オペレーション】　if conditions are satisfied
then GR[reg3] ← GR[reg1] + GR[reg2] +1
else GR[reg3] ← GR[reg1] + GR[reg2] +0

【フォーマット】　Format XI

【オペコード】

15	031	16
rrrrr111111RRRRR	wwwww011101cccc0	

【フラグ】

CY	MSB からのキャリーがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　条件コード「cccc」で指定された条件が満たされた場合は、汎用レジスタ reg2 のワード・データに汎用レジスタ reg1 のワード・データを加算した結果に、1 を加算し、その結果を汎用レジスタ reg3 に格納します。

条件コード「cccc」で指定された条件が満たされなかった場合は、汎用レジスタ reg2 のワード・データに汎用レジスタ reg1 のワード・データを加算し、その結果を汎用レジスタ reg3 に格納します。

汎用レジスタ reg1, reg2 は影響を受けません。

次の表で示されている条件コードのうちの 1 つを「cccc」として指定してください（ただし、cccc ≠ 1101）。

条件コード	条件名	条件式
0000	V	OV = 1
1000	NV	OV = 0
0001	C/L	CY = 1
1001	NC/NL	CY = 0
0010	Z	Z = 1
1010	NZ	Z = 0
0011	NH	(CY or Z) = 1
1011	H	(CY or Z) = 0

条件コード	条件名	条件式
0100	S/N	S = 1
1100	NS/P	S = 0
0101	T	always（無条件）
0110	LT	(S xor OV) = 1
1110	GE	(S xor OV) = 0
0111	LE	((S xor OV) or Z) = 1
1111	GT	((S xor OV) or Z) = 0
（1101）	設定禁止	

＜論理演算命令＞

AND

AND

論理積

【命令形式】　AND reg1, reg2

【オペレーション】　GR[reg2] ← GR[reg2] AND GR[reg1]

【フォーマット】　Format I

【オペコード】

15	0
rrrrr001010RRRRR	

【フラグ】

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データと汎用レジスタ reg1 のワード・データの論理積をとり、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

＜論理演算命令＞

ANDI

AND immediate

論理積

【命令形式】　ANDI imm16, reg1, reg2

【オペレーション】　GR[reg2] ← GR[reg1] AND zero-extend (imm16)

【フォーマット】　Format VI

【オペコード】

15	0	31	16
rrrrr110110RRRRR		iiiiiiiiiiiiiiii	

【フラグ】

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg1 のワード・データと 16 ビット・イミディエイトをワード長までゼロ拡張した値の論理積をとり、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

＜分岐命令＞

Bcond

Branch on condition code with 9-bit displacement

条件分岐

【命令形式】　（1）Bcond　disp9

（2）Bcond　disp17

【オペレーション】　（1）if conditions are satisfied
then PC ← PC + sign-extend (disp9)

（2）if conditions are satisfied
then PC ← PC + sign-extend (disp17)

【フォーマット】　（1）Format III

（2）Format VII

【オペコード】

(1)

15	0
ddddd1011dddcccc	

ただし、ddddddddは disp9 の上位 8 ビットです。cccc は、cond で示される条件の条件コードです（**「表 7.5　Bcond 命令一覧」**参照）。

(2)

15	0	31	16
00000111111Dcccc		ddddddddddddddd1	

ただし、Dddddddddddddddd は disp17 の上位 16 ビットです。
cccc は、cond で示される条件の条件コードです（**「表 7.5　Bcond 命令一覧」**参照）。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　（1）命令が指定する PSW の各フラグをテストし、条件を満たしているときは分岐し、そうでないときは次の命令に進みます。分岐先 PC は、現在の PC と 8 ビット・イミディエイトを 1 ビット・シフトしてワード長まで符号拡張した 9 ビット・ディスプレースメントを加算した値です。

（2）命令が指定する PSW の各フラグをテストし、条件を満たしているときは 16 ビット・イミディエイトを 1 ビット論理左シフトしてワード長まで符号拡張した結果に現在の PC を加算した値を PC に設定し、制御を移します。そうでないときは次の命令に進みます。条件コードとして BR (0101) を指定することはできません。

【補　足】　9 ビット・ディスプレースメントのビット 0 は 0 にマスクされます。なお、計算に使用される現在の PC とは、この命令自身の先頭バイトのアドレスであるためディスプレースメント値が 0 のときは、分岐先はこの命令自身になります。

表 7.5　Bcond 命令一覧

命令		条件コード（cccc）	フラグの状態	分岐条件
符号付き整数	BGE	1110	(S xor OV) = 0	Greater than or equal signed
	BGT	1111	((S xor OV) or Z) = 0	Greater than signed
	BLE	0111	((S xor OV) or Z) = 1	Less than or equal signed
	BLT	0110	(S xor OV) = 1	Less than signed
整数符号なし整数	BH	1011	(CY or Z) = 0	Higher (Greater than)
	BL	0001	CY = 1	Lower (Less than)
	BNH	0011	(CY or Z) = 1	Not higher (Less than or equal)
	BNL	1001	CY = 0	Not lower (Greater than or equal)
共通	BE	0010	Z = 1	Equal
	BNE	1010	Z = 0	Not equal
その他	BC	0001	CY = 1	Carry
	BF	1010	Z = 0	False
	BN	0100	S = 1	Negative
	BNC	1001	CY = 0	No carry
	BNV	1000	OV = 0	No overflow
	BNZ	1010	Z = 0	Not zero
	BP	1100	S = 0	Positive
	BR	0101	—	Always（無条件） 形式 (2) では指定できません
	BSA	1101	SAT = 1	Saturated
	BT	0010	Z = 1	True
	BV	0000	OV = 1	Overflow
	BZ	0010	Z = 1	Zero

注　意

1. 飽和演算命令の実行結果で SAT フラグがセット（1）された場合、符号付き整数の条件分岐（BGE, BGT, BLE, BLT）は、分岐条件に意味がなくなります。これは、次の理由によるものです。通常の演算では、結果が正の最大値を越えると負の値になり、負の最大値を越えたときは正の値になります。つまり、オーバフローが生じると、S フラグが反転（0 → 1, 1 → 0）します。一方、飽和演算命令では、結果が正の最大値を越えたときは正の値で、負の最大値を越えたときは負の値で飽和します。通常の演算とは異なり、オーバフローが生じても S フラグは反転しません。このように、演算結果が飽和したときの S フラグは通常の演算とは異なるので、OV フラグとの排他的論理和（XOR）をとる分岐条件に意味がなくなります。
2. 形式（2）Bcond disp17 では、条件コードとして BR (0101) を指定することはできません。

＜データ操作命令＞

BINS

Bitfield Insert

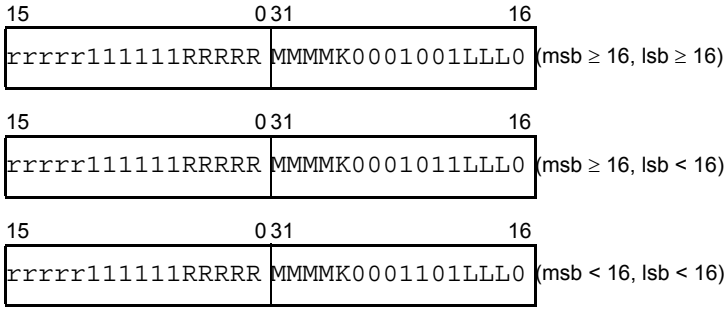
レジスタ上のビット挿入

【命令形式】　BINS　reg1, pos,width, reg2

【オペレーション】　GR[reg2] ← GR[reg2] (31:width+pos) // GR[reg1] (width-1:0) // GR[reg2] (pos-1:0)

【フォーマット】　Format IX

【オペコード】



更新されるフィールドの最上位ビット：msb = pos + width – 1
更新されるフィールドの最下位ビット：lsb = pos
MMMM = msb の下位 4 ビット、KLLL = lsb の下位 4 ビット

【フラグ】

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg1 の下位 width ビットを取り出し、汎用レジスタ reg2 のフィールド、ビット pos ＋ width – 1 からビット pos に格納する命令です。汎用レジスタ reg2 のうち、指定されたフィールド以外の値は影響を受けません。汎用レジスタ reg1 は影響を受けません。

【補　足】　汎用レジスタ reg2 のうち、更新されるフィールドの最上位ビット msb（ビット pos ＋ widhth – 1）と最下位ビット lsb（ビット pos）は、それぞれその下位 4 ビットを BINS 命令の MMMM フィールド、KLLL フィールドで指定します。
msb, lsb の値により、サブオペコード･フィールドの下位 3 ビット（ビット 23 ～ 21）が異なります。
msb < lsb のときの動作は不定です。

＜データ操作命令＞

BSH

Byte swap half-word

ハーフワード・データのバイト・スワップ

【命令形式】　BSH reg2, reg3

【オペレーション】　GR[reg3] ← GR[reg2] (23:16) || GR[reg2] (31:24) || GR[reg2] (7:0) || GR[reg2] (15:8)

【フォーマット】　Format XII

【オペコード】

15	0 31	16
rrrrr11111100000	wwwww01101000010	

【フラグ】

CY	演算結果の下位ハーフワード・データ中に、0 のバイトが 1 つ以上含まれるとき 1、そうでないとき 0
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果の下位ハーフワード・データが 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　エンディアン変換します。

＜データ操作命令＞

BSW

Byte swap word

ワード・データのバイト・スワップ

【命令形式】　BSW　reg2, reg3

【オペレーション】　GR[reg3] ← GR[reg2] (7:0) || GR[reg2] (15:8) || GR[reg2] (23:16) || GR[reg2] (31:24)

【フォーマット】　Format XII

【オペコード】

15	0 31	16
rrrrr11111100000	wwwww01101000000	

【フラグ】

CY	演算結果のワード・データ中に、0 のバイトが 1 つ以上含まれるとき 1、そうでないとき 0
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果のワード・データが 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　エンディアン変換します。

＜特殊命令＞

CALLT

Call with table look up

テーブル参照によるサブルーチン・コール

【命令形式】　CALLT　imm6

【オペレーション】
CTPC ← PC + 2 (return PC)
CTPSW(4:0) ← PSW(4:0)
adr ← CTBP + zero-extend (imm6 logically shift left by 1) 注 1
PC ← CTBP + zero-extend (Load-memory (adr, Half-word))

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　Format II

【オペコード】

15															0
0	0	0	0	0	0	1	0	0	0	i	i	i	i	i	i

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　次の順に処理を行います。

<1> 復帰 PC と PSW の内容を CTPC と CTPSW に転送

<2> CTBP の値と、1 ビット論理左シフトしワード長までゼロ拡張した 6 ビット・イミディエイト・データを加算して 32 ビット・テーブル・エントリ・アドレスを生成

<3> <2> で生成されたアドレスのハーフワードをロードし、ワード長までゼロ拡張

<4> <3> のデータに CTBP の値を加算して 32 ビット・ターゲット・アドレスを生成

<5> <4> で生成されたターゲット・アドレスへ分岐

注　意

1. 命令実行中に例外が発生すると、リード・サイクルが終了したあとに命令の実行を中止する場合があります。
2. CALLT 命令のテーブル読み出しのためのメモリからの読み出し操作では、メモリ保護が行われます。
   メモリ保護が有効である場合に、ユーザ・プログラムからのアクセスが禁止されている領域に配置されているテーブルからターゲット・アドレスを生成するためのデータをロードすることはできません。

＜特殊命令＞

CAXI

Compare and exchange for interlock

比較と交換

【命令形式】　CAXI [reg1], reg2, reg3

【オペレーション】

adr ← GR[reg1] 注 1

token ← Load-memory(adr, Word)

result ← GR[reg2] – token

If result == 0

then Store-memory(adr, GR[reg3],Word)
GR[reg3] ← token

else Store-memory(adr, token,Word)
GR[reg3] ← token

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　Format XI

【オペコード】

15	0 31	16
rrrrr111111RRRRR	wwwww00011101110	

【フラグ】

CY	result の演算時に MSB へのボローがあれば 1、そうでないとき 0
OV	result の演算時にオーバフローが起こったとき 1、そうでないとき 0
S	result が負のとき 1、そうでないとき 0
Z	result が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】

指定したアドレスからワード・データを読み出し、汎用レジスタ reg2 のワード・データと比較し、結果を PSW の各フラグに示します。比較は汎用レジスタ reg2 のワード・データから、読み出したワード・データを減算することで行います。比較の結果が 0 であれば、汎用レジスタ reg3 のワード・データを、そうでなければ、読み出したワード・データを、生成したアドレスに格納します。

その後、読み出したワード・データを汎用レジスタ reg3 へ格納します。汎用レジスタ reg1, reg2 は影響を受けません。

注　意

1. この命令は排他制御を目的としたアトミック性保証のため、読み出しから書き込みまでの間、対象のアドレスが他の要因によるアクセスによって操作されることはありません。
2. CAXI 命令は後方互換のために維持されています。マルチコア・システムでのアトミック性保証が必要な場合には、LDL.W 命令と STC.W 命令を使用してください。

＜特殊命令＞

Clear Load Link

CLL

アトミック操作のリンク解除

【命令形式】　CLL

【オペレーション】　LLbit ← 0

【フォーマット】　Format X

【オペコード】

15	0 31	16
1111111111111111	1111000101100000	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　LDL.W 命令により生成した自 CPU コアのリンクを消去します。

コア間のリンクに対する動作は、**「5.3.2　LDL.W, STC.W 命令による相互排除」**を参照してください。

注　意

マルチコア・システム等において CLL 命令がどのような動作を行うかは製品のシステム構成に依存します。詳細は製品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＞

CLR1

Clear bit

ビット・クリア

【命令形式】　（1）CLR1　bit#3, disp16 [reg1]

（2）CLR1　reg2, [reg1]

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1
token ← Load-memory (adr, Byte)
Z フラグ ← Not (extract-bit (token, bit#3))
token ← clear-bit (token, bit#3)
Store-memory (adr, token, Byte)

（2）adr ← GR[reg1] 注 1

token ← Load-memory (adr, Byte)

Z フラグ ← Not (extract-bit (token, reg2))

token ← clear-bit (token, reg2)

Store-memory (adr, token, Byte)…

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VIII

（2）Format IX

【オペコード】

（1）

15	0	31	16
10bbb111110RRRRR		ddddddddddddddddd	

（2）

15	0	31	16
rrrrr111111RRRRR		0000000011100100	

【フ ラ グ】

CY	—
OV	—
S	—
Z	指定したビットが 0 のとき 1、指定したビットが 1 のとき 0
SAT	—

【説　明】

(1) まず、汎用レジスタ reg1 のワード・データと、ワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスのバイト・データを読み出し、3 ビットのビット・ナンバで指定されるビットをクリア（0）し、元のアドレスに書き戻します。読み出したバイト・データの指定ビットが 0 のとき Z フラグをセット（1）し、指定ビットが 1 のとき Z フラグをクリア（0）します。

(2) まず、汎用レジスタ reg1 のワード・データを読み出して 32 ビット・アドレスを生成します。生成したアドレスのバイト・データを読み出し、汎用レジスタ reg2 の下位 3 ビットで指定されるビットをクリア（0）し、元のアドレスに書き戻します。読み出したバイト・データの指定ビットが 0 のとき Z フラグをセット（1）し、指定ビットが 1 のとき Z フラグをクリア（0）します。

【補　足】

PSW の Z フラグはこの命令を実行する前に該当ビットが 0 か 1 だったかを示します。この命令実行後の該当ビットの内容を示すものではありません。

注　意

この命令は排他制御を目的としたアトミック性保証のため、読み出しから書き込みまでの間、対象のアドレスが他の要因によるアクセスによって操作されることはありません。

＜データ操作命令＞

CMOV

Conditional move

条件付き転送

【命令形式】　（1）CMOV　cccc, reg1, reg2, reg3

（2）CMOV　cccc, imm5, reg2, reg3

【オペレーション】　（1）if conditions are satisfied

then GR[reg3] ← GR[reg1]

else GR[reg3] ← GR[reg2]

（2）if conditions are satisfied

then GR[reg3] ← sign-extended (imm5)

else GR[reg3] ← GR[reg2]

【フォーマット】　（1）Format XI

（2）Format XII

【オペコード】

(1)

15 0	31 16
rrrrr111111RRRRR	wwwww011001cccc0

(2)

15 0	31 16
rrrrr111111iiiii	wwwww011000cccc0

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

(1) 条件コード「cccc」で指定された条件が満たされた場合は汎用レジスタ reg1 のデータを、満たされなかった場合は汎用レジスタ reg2 のデータを、汎用レジスタ reg3 に転送します。次の表で示されている条件コードのうちの 1 つを「cccc」として指定してください。

条件コード	条件名	条件式
0000	V	OV = 1
1000	NV	OV = 0
0001	C/L	CY = 1
1001	NC/NL	CY = 0
0010	Z	Z = 1
1010	NZ	Z = 0
0011	NH	(CY or Z) = 1
1011	H	(CY or Z) = 0

条件コード	条件名	条件式
0100	S/N	S = 1
1100	NS/P	S = 0
0101	T	always（無条件）
1101	SA	SAT = 1
0110	LT	(S xor OV) = 1
1110	GE	(S xor OV) = 0
0111	LE	((S xor OV) or Z) = 1
1111	GT	((S xor OV) or Z) = 0

(2) 条件コード「cccc」で指定された条件が満たされた場合はワード長まで符号拡張した 5 ビット・イミディエイト・データを、満たされなかった場合は汎用レジスタ reg2 のデータを、汎用レジスタ reg3 に転送します。次の表で示されている条件コードのうちの 1 つを「cccc」として指定してください。

条件コード	条件名	条件式
0000	V	OV = 1
1000	NV	OV = 0
0001	C/L	CY = 1
1001	NC/NL	CY = 0
0010	Z	Z = 1
1010	NZ	Z = 0
0011	NH	(CY or Z) = 1
1011	H	(CY or Z) = 0

条件コード	条件名	条件式
0100	S/N	S = 1
1100	NS/P	S = 0
0101	T	always（無条件）
1101	SA	SAT = 1
0110	LT	(S xor OV) = 1
1110	GE	(S xor OV) = 0
0111	LE	((S xor OV) or Z) = 1
1111	GT	((S xor OV) or Z) = 0

【補　足】

SETF 命令を参照してください。

＜算術演算命令＞

CMP

Compare register/immediate (5-bit)

比較

【命令形式】　（1）CMP　reg1, reg2

（2）CMP　imm5, reg2

【オペレーション】　（1）result ← GR[reg2] – GR[reg1]

（2）result ← GR[reg2] – sign-extend (imm5)

【フォーマット】　（1）Format I

（2）Format II

【オペコード】

(1)

15															0
r	r	r	r	r	0	0	1	1	1	1	R	R	R	R	R

(2)

15															0
r	r	r	r	r	0	1	0	0	1	1	i	i	i	i	i

【フラグ】

CY	MSB へのボローがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　（1）汎用レジスタ reg2 のワード・データと汎用レジスタ reg1 のワード・データを比較し、結果を PSW の各フラグに示します。比較は汎用レジスタ reg2 のワード・データから汎用レジスタ reg1 の内容を減算することで行います。汎用レジスタ reg1, reg2 は影響を受けません。

（2）汎用レジスタ reg2 のワード・データとワード長まで符号拡張した 5 ビット・イミディエイトを比較し、結果を PSW の各フラグに示します。比較は汎用レジスタ reg2 のワード・データから符号拡張したイミディエイトの内容を減算することで行います。汎用レジスタ reg2 は影響を受けません。

＜特殊命令＞

CTRET

Return from CALLT

サブルーチン・コールからの復帰

【命令形式】　CTRET

【オペレーション】　PC ← CTPC

PSW (4:0) ← CTPSW (4:0)

【フォーマット】　Format X

【オペコード】

15	0 31	16
0000011111100000	0000000101000100	

【フラグ】

CY	CTPSW から読み出した値が設定される
OV	CTPSW から読み出した値が設定される
S	CTPSW から読み出した値が設定される
Z	CTPSW から読み出した値が設定される
SAT	CTPSW から読み出した値が設定される

【説　明】　システム・レジスタから復帰 PC と PSW（下位 5 ビット）を取り出し、CALLT 命令により呼び出されたルーチンから復帰します。この命令の動作は次のとおりです。

<1> 復帰 PC と復帰 PSW（下位 5 ビット）を、CTPC と CTPSW（下位 5 ビット）から取り出します。

<2> 取り出した復帰 PC と復帰 PSW（下位 5 ビット）を、PC と PSW（下位 5 ビット）に設定します。

注　意

CTRET 命令では PSW レジスタの下位 5 ビットのみが更新され、上位 27 ビットは保持されます。

＜特殊命令＞

DI

Disable interrupt

EI レベル・マスカブル例外の禁止

【命令形式】　DI

【オペレーション】　PSW.ID ← 1（EI レベル・マスカブル例外の禁止）

【フォーマット】　Format X

【オペコード】

15	0	31	16
0000011111100000		0000000101100000	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—
ID	1

【説　明】　PSW の ID ビットをセット（1）し、この命令実行後から EI レベル・マスカブル例外の受け付けを禁止します。

【補　足】　この命令による PSW のフラグの書き換えが有効になるのは次の命令からとなります。

MCTL.UIC がクリア（0）されている場合はスーパバイザ特権命令です。

MCTL.UIC がセット（1）されている場合は常に実行可能です。

＜特殊命令＞

DISPOSE

Function dispose

スタック・フレームの削除

【命令形式】　（1）DISPOSE　imm5, list12

（2）DISPOSE　imm5, list12, [reg1]

【オペレーション】　（1）tmp ← sp ＋ zero-extend (imm5 logically shift by 2)
foreach (all regs in list12) {
adr ← tmp 注 1, 注 2
GR[reg in list12] ← Load-memory (adr, Word)
tmp ← tmp ＋ 4
}
sp ← tmp

（2）tmp ← sp ＋ zero-extend (imm5 logically shift by 2)
foreach (all regs in list12) {
adr ← tmp 注 1, 注 2
GR[reg in list12] ← Load-memory (adr, Word)
tmp ← tmp ＋ 4
}
PC ← GR[reg1]
sp ← tmp

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

注 2. adr の下位 2 ビットは 0 にマスクされます。

【フォーマット】　Format XIII

【オペコード】

(1)

15 … 0	31 … 16
0000011001iiiiiL	LLLLLLLLLLL00000

(2)

15 … 0	31 … 16
0000011001iiiiiL	LLLLLLLLLLLRRRRR

RRRRR ≠ 00000（reg1 には r0 を設定しないでください）

また、LLLLLLLLLLL は、レジスタ・リスト「list12」の中の対応するビットの値を示します（たとえば、オペコード中のビット 21 の「L」は list12 のビット 21 の値を示します）。

list12 は、次のように定義される 32 ビットのレジスタ・リストです。

31	30	29	28	27	26	25	24	23	22	21	20 … 1	0
r24	r25	r26	r27	r20	r21	r22	r23	r28	r29	r31	—	r30

ビット 31 ～ 21 とビット 0 の各ビットに汎用レジスタ（r20 ～ r31）が対応しており、セット（1）されたビットに対応するレジスタが操作の対象として指定されます。たとえば、r20, r30 を指定する場合、list12 の値は次のようになります（レジスタが対応付けられていないビット 20 ～ 1 への設定値は任意です）。

- レジスタが対応付けられていないビットの値をすべて 0 とした場合：$0800\ 0001_H$
- レジスタが対応付けられていないビットの値をすべて 1 とした場合：$081F\ FFFF_H$

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

（1）5 ビット・イミディエイト・データを、2 ビット論理左シフトし、ワード長までゼロ拡張したワード・データを、sp に加算します。そして、list12 で指定されている汎用レジスタに復帰（sp で指定するアドレスからデータをロードし、sp に 4 を加算）します。

（2）5 ビット・イミディエイト・データを、2 ビット論理左シフトし、ワード長までゼロ拡張したワード・データを、sp に加算します。そして、list12 で指定されている汎用レジスタに復帰（sp で指定するアドレスからデータをロードし、sp に 4 を加算）し、汎用レジスタ reg1 で指定されたアドレスに制御を移します。

【補　足】

list12 の汎用レジスタは、降順にロードされます（r31, r30, ... , r20）。
imm5 は、自動変数と一時データのためのスタック・フレームを復元します。
sp で指定された下位 2 ビットのアドレスは、0 でマスクされ、ワード境界にアラインされます。

注　意

1. 命令実行中に例外が発生すると、リード・サイクルとレジスタ値の書き換えが終了したあとに、命令の実行を中止する場合がありますが、sp は実行開始前の元の値を保持します。そのあと、例外から復帰すると、命令が再実行されます。
2. 命令形式（2）の DISPOSE imm5, list12, [reg1] では、reg1 には r0 を指定しないでください。

＜除算命令＞

Divide word

DIV

（符号付き）ワード・データの除算

【命令形式】　DIV　reg1, reg2, reg3

【オペレーション】　GR[reg2] ← GR[reg2] ÷ GR[reg1]

GR[reg3] ← GR[reg2] % GR[reg1]

【フォーマット】　Format XI

【オペコード】

15	0 31	16
rrrrr111111RRRRR	wwwww01011000000	

【フラグ】

CY	—
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果の商が負のとき 1、そうでないとき 0
Z	演算結果の商が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 のワード・データで除算し、その商を汎用レジスタ reg2 に、余りを汎用レジスタ reg3 に格納します。汎用レジスタ reg1 は影響を受けません。ゼロで割ったときは、オーバフローを生じ、OV フラグ以外の演算結果は不定となります。

【補　足】　オーバフローは負の最大値（$8000\ 0000_H$）を –1 で割ったとき（商が $8000\ 0000_H$）と、ゼロによる除算のとき（演算結果は不定）に生じます。
汎用レジスタ reg2 と汎用レジスタ reg3 が同じレジスタの場合、そのレジスタには余りが格納されます。

この命令実行中に例外が発生すると、実行を中止し、戻り番地をこの命令の先頭アドレスとして例外を処理してから、例外処理完了後に再実行します。この場合、汎用レジスタ reg1 と汎用レジスタ reg2 はこの命令実行前の値を保持します。

注　意

汎用レジスタ reg2 と汎用レジスタ reg3 に同じレジスタを指定した場合、汎用レジスタ reg2 に演算結果の商が格納されないので、フラグは不定となります。

＜除算命令＞

Divide half-word

DIVH

（符号付き）ハーフワード・データの除算

【命令形式】　（1）DIVH　reg1, reg2

（2）DIVH　reg1, reg2, reg3

【オペレーション】　（1）GR[reg2] ← GR[reg2] ÷ sign-extend (GR[reg1] (15:0))

（2）GR[reg2] ← GR[reg2] ÷ sign-extend (GR[reg1] (15:0))
GR[reg3] ← GR[reg2] % sign-extend (GR[reg1] (15:0))

【フォーマット】　（1）Format I

（2）Format XI

【オペコード】

(1)

15	0
rrrrr000010RRRRR	

RRRRR ≠ 00000（reg1 には r0 を設定しないでください）

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

(2)

15	0	31	16
rrrrr111111RRRRR		wwwww01010000000	

【フラグ】

CY	—
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果の商が負のとき 1、そうでないとき 0
Z	演算結果の商が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　（1）汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位ハーフワード・データで除算し、その商を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。ゼロで割ったときは、オーバフローを生じ、OV フラグ以外の演算結果は不定となります。

（2）汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位ハーフワード・データで除算し、その商を汎用レジスタ reg2 に、余りを汎用レジスタ reg3 に格納します。汎用レジスタ reg1 は影響を受けません。ゼロで割ったときは、オーバフローを生じ、OV フラグ以外の演算結果は不定となります。

【補　足】

(1) 除算結果の余りは格納されません。オーバフローは負の最大値（$8000\ 0000_H$）を –1 で割ったとき（商が $8000\ 0000_H$）と、ゼロによる除算のとき（演算結果は不定）に生じます。この命令実行中に例外が発生すると、実行を中止し、戻り番地をこの命令の先頭アドレスとして例外を処理してから、例外処理完了後に再実行します。この場合、汎用レジスタ reg1 と汎用レジスタ reg2 はこの命令実行前の値を保持します。

(2) オーバフローは負の最大値（$8000\ 0000_H$）を – 1 で割ったとき（商が $8000\ 0000_H$）と、ゼロによる除算のとき（演算結果は不定）に生じます。
汎用レジスタ reg2 と汎用レジスタ reg3 が同じレジスタの場合、そのレジスタには余りが格納されます。
この命令実行中に例外が発生すると、実行を中止し、戻り番地をこの命令の先頭アドレスとして例外を処理してから、例外処理完了後に再実行します。この場合、汎用レジスタ reg1 と汎用レジスタ reg2 はこの命令実行前の値を保持します。

注　意

1. 汎用レジスタ reg2 と汎用レジスタ reg3 に同じレジスタを指定した場合、汎用レジスタ reg2 に演算結果の商が格納されないので、フラグは不定となります。
2. 命令形式（1）の DIVH　reg1, reg2 では、reg1, reg2 には r0 を指定しないでください。

＜除算命令＞

Divide half-word unsigned

DIVHU

（符号なし）ハーフワード・データの除算

【命令形式】　DIVHU　reg1, reg2, reg3

【オペレーション】　GR[reg2] ← GR[reg2] ÷ zero-extend (GR[reg1] (15:0))

GR[reg3] ← GR[reg2] % zero-extend (GR[reg1] (15:0))

【フォーマット】　Format XI

【オペコード】

15	0	31	16
rrrrr111111RRRRR		wwwww01010000010	

【フラグ】

CY	—
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果の商のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果の商が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位ハーフワード・データで除算し、その商を汎用レジスタ reg2 に、余りを汎用レジスタ reg3 に格納します。汎用レジスタ reg1 は影響を受けません。ゼロで割ったときは、オーバフローを生じ、OV フラグ以外の演算結果は不定となります。

【補　足】　オーバフローはゼロによる除算のとき（演算結果は不定）に生じます。
汎用レジスタ reg2 と汎用レジスタ reg3 が同じレジスタの場合、そのレジスタには余りが格納されます。
この命令実行中に例外が発生すると、実行を中止し、戻り番地をこの命令の先頭アドレスとして例外を処理してから、例外処理完了後に再実行します。この場合、汎用レジスタ reg1 と汎用レジスタ reg2 はこの命令実行前の値を保持します。

注　意

汎用レジスタ reg2 と汎用レジスタ reg3 に同じレジスタを指定した場合、汎用レジスタ reg2 に演算結果の商が格納されないので、フラグは不定となります。

＜高速除算命令＞

Divide word quickly

DIVQ

（符号付き）ワード・データの除算（可変ステップ）

【命令形式】　DIVQ　reg1, reg2, reg3

【オペレーション】　GR[reg2] ← GR[reg2] ÷ GR[reg1]

GR[reg3] ← GR[reg2] % GR[reg1]

【フォーマット】　Format XI

【オペコード】

15	0 31	16
rrrrr111111RRRRR	wwwww01011111100	

【フラグ】

CY	—
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果の商が負のとき 1、そうでないとき 0
Z	演算結果の商が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 のワード・データで除算し、その商を汎用レジスタ reg2 に、余りを汎用レジスタ reg3 に格納します。汎用レジスタ reg1 は影響を受けません。

reg1, reg2 の値から除算に必要となる最小なステップ数を判断して、演算を実行します。

ゼロで割ったときは、オーバフローを生じ、OV フラグ以外の演算結果は不定となります。

【補　足】　(1) オーバフローは負の最大値（$8000\ 0000_H$）を –1 で割ったとき（商が $8000\ 0000_H$）と、ゼロによる除算のとき（演算結果は不定）に生じます。
汎用レジスタ reg2 と汎用レジスタ reg3 が同じレジスタの場合、そのレジスタには余りが格納されます。
この命令実行中に例外が発生すると、実行を中止し、戻り番地をこの命令の先頭アドレスとして例外を処理してから、例外処理完了後に再実行します。この場合、汎用レジスタ reg1 と汎用レジスタ reg2 はこの命令実行前の値を保持します。

(2) 実行サイクル数は reg1, reg2 の有効ビット数の差が小さいほど少なく、ほとんどの場合に通常の除算命令より実行サイクル数が少なくなります。16 ビット整数型のデータ同士の除算の場合、有効ビット数の差は 15 ビット以下であり、20 サイクル以内で演算を完了します。

注　意

1. 汎用レジスタ reg2 と汎用レジスタ reg3 に同じレジスタを指定した場合、汎用レジスタ reg2 に演算結果の商が格納されないので、フラグは不定となります。
2. 正確な実行サイクル数は、付録を参照してください。
3. リアルタイム性の保証などのために、常に実行サイクル数が一定であることが必要な場合は、通常の除算命令を使用してください。

＜高速除算命令＞

Divide word unsigned quickly

DIVQU

（符号なし）ワード・データの除算（可変ステップ）

【命令形式】　DIVQU　reg1, reg2, reg3

【オペレーション】　GR[reg2] ← GR[reg2] ÷ GR[reg1]

GR[reg3] ← GR[reg2] % GR[reg1]

【フォーマット】　Format XI

【オペコード】

15	0	31	16
rrrrr111111RRRRR		wwwww01011111110	

【フラグ】

CY	—
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果の商のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果の商が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 のワード・データで除算し、その商を汎用レジスタ reg2 に、余りを汎用レジスタ reg3 に格納します。汎用レジスタ reg1 は影響を受けません。

reg1, reg2 の値から除算に必要となる最小なステップ数を判断して、演算を実行します。

ゼロで割ったときは、オーバフローを生じ、OV フラグ以外の演算結果は不定となります。

【補　足】　(1) オーバフローはゼロによる除算のとき（演算結果は不定）に生じます。
汎用レジスタ reg2 と汎用レジスタ reg3 が同じレジスタの場合、そのレジスタには余りが格納されます。
この命令実行中に例外が発生すると、実行を中止し、戻り番地をこの命令の先頭アドレスとして例外を処理してから、例外処理完了後に再実行します。この場合、汎用レジスタ reg1 と汎用レジスタ reg2 はこの命令実行前の値を保持します。

(2) 実行サイクル数は reg1, reg2 の有効ビット数の差が小さいほど少なく、ほとんどの場合に通常の除算命令より実行サイクル数が少なくなります。16 ビット整数型のデータ同士の除算の場合、有効ビット数の差は 15 ビット以下であり、20 サイクル以内で演算を完了します。

注　意

1. 汎用レジスタ reg2 と汎用レジスタ reg3 に同じレジスタを指定した場合、汎用レジスタ reg2 に演算結果の商が格納されないので、フラグは不定となります。
2. 正確な実行サイクル数は、付録を参照してください。
3. リアルタイム性の保証などのために、常に実行サイクル数が一定であることが必要な場合は、通常の除算命令を使用してください。

＜除算命令＞

DIVU

Divide word unsigned

（符号なし）ワード・データの除算

【命令形式】　DIVU　reg1, reg2, reg3

【オペレーション】　GR[reg2] ← GR[reg2] ÷ GR[reg1]

GR[reg3] ← GR[reg2] % GR[reg1]

【フォーマット】　Format XI

【オペコード】

15	0 31	16
rrrrr111111RRRRR	wwwww01011000010	

【フラグ】

CY	—
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果の商のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果の商が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 のワード・データで除算し、その商を汎用レジスタ reg2 に、余りを汎用レジスタ reg3 に格納します。汎用レジスタ reg1 は影響を受けません。

ゼロで割ったときは、オーバフローを生じ、OV フラグ以外の演算結果は不定となります。

【補　足】　オーバフローはゼロによる除算のとき（演算結果は不定）に生じます。

汎用レジスタ reg2 と汎用レジスタ reg3 が同じレジスタの場合、そのレジスタには余りが格納されます。

この命令実行中に例外が発生すると、実行を中止し、戻り番地をこの命令の先頭アドレスとして例外を処理してから、例外処理完了後に再実行します。この場合、汎用レジスタ reg1 と汎用レジスタ reg2 はこの命令実行前の値を保持します。

注　意

汎用レジスタ reg2 と汎用レジスタ reg3 に同じレジスタを指定した場合、汎用レジスタ reg2 に演算結果の商が格納されないので、フラグは不定となります。

＜特殊命令＞

EI

Enable interrupt

EI レベル・マスカブル例外の許可

【命令形式】　EI

【オペレーション】　PSW.ID ← 0（EI レベル・マスカブル例外の許可）

【フォーマット】　Format X

【オペコード】

15	0	31	16
1000011111100000		0000000101100000	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—
ID	0

【説　明】　PSW の ID ビットをクリア（0）し、次の命令より EI レベル・マスカブル例外の受け付けを許可します。

【補　足】　MCTL.UIC がクリア（0）されている場合はスーパバイザ特権命令です。

MCTL.UIC がセット（1）されている場合は常に実行可能です。

＜特殊命令＞

EIRET

Return from trap or interrupt

EI レベル例外からの復帰

【命令形式】　EIRET

【オペレーション】　PC ← EIPC

PSW ← EIPSW

【フォーマット】　Format X

【オペコード】

15	0 31	16
0000011111100000	0000000101001000	

【フラグ】

CY	EIPSW から読み出した値が設定される
OV	EIPSW から読み出した値が設定される
S	EIPSW から読み出した値が設定される
Z	EIPSW から読み出した値が設定される
SAT	EIPSW から読み出した値が設定される

【説　明】　EI レベル例外から復帰する命令です。EIPC, EIPSW から復帰 PC と PSW を取り出し、PC, PSW に設定し制御を移します。

また、EP = 0 の場合、割り込み（EIINT*n*）処理を終了したため、ISPR レジスタの所定のビットをクリアします。

【補　足】　この命令はスーパバイザ特権命令です。

＜特殊命令＞

FERET

Return from trap or interrupt

FE レベル例外からの復帰

【命令形式】　FERET

【オペレーション】　PC ← FEPC

PSW ← FEPSW

【フォーマット】　Format X

【オペコード】

15	0 31	16
0000011111100000	0000000101001010	

【フラグ】

CY	FEPSW から読み出した値が設定される
OV	FEPSW から読み出した値が設定される
S	FEPSW から読み出した値が設定される
Z	FEPSW から読み出した値が設定される
SAT	FEPSW から読み出した値が設定される

【説　明】　FE レベル例外から復帰する命令です。FEPC, FEPSW から復帰 PC と PSW を取り出し、PC, PSW に設定し制御を移します。

【補　足】　この命令はスーパバイザ特権命令です。

注　意

FERET 命令は、ハザード・バリア命令として OS などの管理プログラムによって、CPU 動作状態（PSW）を変更する際にも利用します。搭載された CPU によっては、PSW 上の UM ビットなどを変更する際に、そのビットに関連するハードウェア機能（主にメモリ管理機能）が効力を発揮するプログラム・ブロックを明確にするために FERET 命令を使用します。FERET 命令によって更新された PSW の値にしたがって動作するハードウェア機能は、FERET 命令の復帰アドレスが示す命令から効力を発揮することを保証します。

＜特殊命令＞

FETRAP

FE-level Trap

FE レベル・ソフトウェア例外

【命令形式】　FETRAP　vector4

【オペレーション】　FEPC ← PC + 2 （復帰 PC）

FEPSW ← PSW

FEIC ← 例外要因コード注 1

PSW.UM ← 0

PSW.NP ← 1

PSW.EP ← 1

PSW.ID ← 1

PC ← 例外ハンドラ・アドレス注 2

注 1.「表 4.1　例外要因一覧」を参照してください。
注 2.「4.5　例外ハンドラ・アドレス」を参照してください。

【フォーマット】　Format I

【オペコード】

15	0
0vvvv00001000000	

ただし、vvvv は vector4 です。

また、vector4 には 0_H を設定しないでください（vvvv ≠ 0000）。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

復帰 PC（FETRAP 命令の次の命令のアドレス）と現在の PSW の内容を、それぞれ FEPC と FEPSW に退避し、例外要因コードを FEIC レジスタに格納、PSW を **「表 4.1　例外要因一覧」**にしたがって更新します。続いて、例外ハンドラ・アドレスに分岐し、例外処理を開始します。

vector4 と例外要因コード、例外ハンドラ・アドレスのオフセットの対応は**表 7.6** で示されます。例外ハンドラ・アドレスは、**表 7.6** が示すオフセット・アドレスを元に計算されます。詳細は、**「4.5　例外ハンドラ・アドレス」**を参照してください。

表 7.6　vector4 と例外要因コード、例外ハンドラ・アドレスのオフセットの対応

vector4	例外要因コード	オフセット・アドレス
0_H	指定不可	
1_H	0000 0031_H	30_H
2_H	0000 0032_H	
（中略）		
F_H	0000 $003F_H$	

＜特殊命令＞

HALT

Halt

停止

【命令形式】　HALT

【オペレーション】　CPU コアが停止状態に遷移します

【フォーマット】　Format X

【オペコード】

15	0 31	16
0000011111100000	0000000100100000	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　HALT 命令を実行した CPU コアが停止状態に遷移します。

停止状態から通常の実行状態への復帰は、特定の例外要求の発生によって行われます。

なお、HALT 状態で例外を受け付けた場合、その例外の復帰 PC は、HALT 命令の次の命令の PC となります。

停止状態の解除条件は次のとおりです。

- すべての中断型例外の発生

また、上記の例外の受け付け条件（ID および NP の値）を満たしていない場合であっても、要求が存在する場合には停止状態の解除が行われます（例：PSW.ID = 1 であっても、INT0 が発生した段階で停止状態が解除されます）。

機能ごとに定義された次のマスク機能によって、中断型例外の発生がマスクされている場合は、停止状態は解除されません。

- 割り込みコントローラによる割り込みチャネルのマスク注 1
- 浮動小数点演算の例外許可ビットによるマスク
- 上記以外のハードウェア機能で定義されたマスク

注 1. ISPR レジスタ、PMR レジスタによるマスクは含まれません。

【補　足】　この命令はスーパバイザ特権命令です。

＜データ操作命令＞

HSH

Half-word swap half-word

ハーフワード・データのハーフワード・スワップ

【命令形式】　HSH　reg2, reg3

【オペレーション】　GR[reg3] ← GR[reg2]

【フォーマット】　Format XII

【オペコード】

15	0	31	16
rrrrr11111100000		wwwww01101000110	

【フラグ】

CY	演算結果の下位ハーフワードが 0 のとき 1、そうでないとき 0
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果の下位ハーフワードが 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 の内容を汎用レジスタ reg3 に格納し、フラグの判定結果を PSW に格納します。

＜データ操作命令＞

HSW

Half-word swap word

ワード・データのハーフワード・スワップ

【命令形式】　　HSW　reg2, reg3

【オペレーション】　　GR[reg3] ← GR[reg2] (15:0) || GR[reg2] (31:16)

【フォーマット】　　Format XII

【オペコード】

15	0 31	16
rrrrr11111100000	wwwww01101000100	

【フラグ】

CY	演算結果のワード・データ中に、0 のハーフワードが 1 つ以上含まれるとき 1、そうでないとき 0
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果のワード・データが 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　　エンディアン変換します。

＜分岐命令＞

JARL

Jump and register link

分岐とレジスタ・リンク

【命令形式】　（1）JARL　disp22, reg2

（2）JARL　disp32, reg1

（3）JARL　[reg1], reg3

【オペレーション】　（1）GR[reg2] ← PC + 4
PC ← PC + sign-extend (disp22)

（2）GR[reg1] ← PC + 6
PC ← PC + disp32

（3）GR[reg3] ← PC + 4
PC ← GR[reg1]

【フォーマット】　（1）Format V

（2）Format VI

（3）Format XI

【オペコード】

(1)

15	0	31	16
rrrrr11110dddddd		ddddddddddddddd0	

ただし、ddddddddddddddddddddd は disp22 の上位 21 ビットです。

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

(2)

15	0	31	16	47	32
00000010111RRRRR		ddddddddddddddd0		DDDDDDDDDDDDDDDD	

ただし、DDDDDDDDDDDDDDDDddddddddddddddddは disp32 の上位 31 ビットです。

RRRRR ≠ 00000（reg1 には r0 を設定しないでください）

(3)

15	0	31	16
11000111111RRRRR		WWWWW00101100000	

WWWWW ≠ 00000（reg3 には r0 を設定しないでください）

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

(1) 現在の PC に 4 を加算した値を汎用レジスタ reg2 に退避し、現在の PC とワード長まで符号拡張した 22 ビット・ディスプレースメントを加算した値を PC に設定し、制御を移します。22 ビット・ディスプレースメントのビット 0 は 0 にマスクされます。

(2) 現在の PC に 6 を加算した値を汎用レジスタ reg1 に退避し、現在の PC と 32 ビット・ディスプレースメントを加算した値を PC に設定し、制御を移します。32 ビット・ディスプレースメントのビット 0 は 0 にマスクされます。

(3) 現在の PC に 4 を加算した値を reg3 に格納し、reg1 の内容を PC に設定し、制御を移します。

【補　足】

計算に使用される現在の PC とは、この命令自身の先頭バイトのアドレスであるためディスプレースメント値が 0 のときは、分岐先はこの命令自身になります。

この命令は、サブルーチン制御命令のコールに相当し、復帰 PC を汎用レジスタ reg1 または reg2 に格納します。一方、リターンに相当する JMP 命令では、復帰 PC を格納している汎用レジスタを汎用レジスタ reg1 として指定して、使用できます。

注　意

命令形式（1）JARL disp22, reg2 では、汎用レジスタ reg2 には r0 を指定しないでください。
命令形式（2）JARL disp32, reg1 では、汎用レジスタ reg1 には r0 を指定しないでください。
命令形式（3）JARL [reg1], reg3 では、汎用レジスタ reg3 には r0 を指定しないでください。

＜分岐命令＞

JMP

Jump register

無条件分岐（レジスタ間接）

【命令形式】　（1）JMP [reg1]

（2）JMP disp32 [reg1]

【オペレーション】　（1）PC ← GR[reg1]

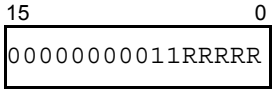
（2）PC ← GR[reg1] + disp32

【フォーマット】　（1）Format I

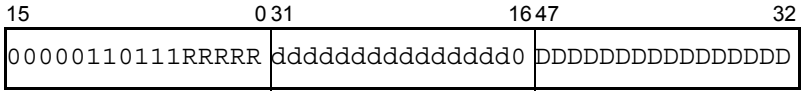
（2）Format VI

【オペコード】

(1)



(2)



ただし、DDDDDDDDDDDDDDDDddddddddddddddd は disp32 の上位 31 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　（1）汎用レジスタ reg1 で指定されるアドレスに制御を移します。アドレスのビット 0 は 0 にマスクされます。

（2）汎用レジスタ reg1 に 32 ビット・ディスプレースメントを加算したアドレスに制御を移します。アドレスのビット 0 は 0 にマスクされます。

【補　足】　この命令をサブルーチン制御命令のリターンとして使用する場合は、復帰 PC を汎用レジスタ reg1 で指定します。

＜分岐命令＞

JR

Jump relative

無条件分岐（PC 相対）

【命令形式】　（1）JR　disp22

（2）JR　disp32

【オペレーション】　（1）PC ← PC + sign-extend (disp22)

（2）PC ← PC + disp32

【フォーマット】　（1）Format V

（2）Format VI

【オペコード】

(1)

15	0	31	16
0000011110dddddd		ddddddddddddddd0	

ただし、ddddddddddddddddddddd は disp22 の上位 21 ビットです。

(2)

15	0	31	16	47	32
0000001011100000		ddddddddddddddd0		DDDDDDDDDDDDDDDD	

ただし、DDDDDDDDDDDDDDDDddddddddddddddd は disp32 の上位 31 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　（1）現在の PC とワード長まで符号拡張した 22 ビット・ディスプレースメントを加算した値を PC に設定し、制御を移します。22 ビット・ディスプレースメントのビット 0 は 0 にマスクされます。

（2）現在の PC と 32 ビット・ディスプレースメントを加算した値を PC に設定し、制御を移します。32 ビット・ディスプレースメントのビット 0 は 0 にマスクされます。

【補　足】　計算に使用される現在の PC とは、この命令自身の先頭バイトのアドレスであるため、ディスプレースメント値が 0 の場合の分岐先は、この命令自身になります。

＜ロード命令＞

Load byte

LD.B

（符号付き）バイト・データのロード

【命令形式】　（1）LD.B　disp16 [reg1] , reg2

（2）LD.B　disp23 [reg1] , reg3

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1
GR[reg2] ← sign-extend (Load-memory (adr, Byte))

（2）adr ← GR[reg1] + sign-extend (disp23) 注 1
GR[reg3] ← sign-extend (Load-memory (adr, Byte))

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VII

（2）Format XIV

【オペコード】

(1)

15	0	31	16
rrrrr111000RRRRR		ddddddddddddddddd	

(2)

15	0	31	16	47	32
00000111100RRRRR		wwwwwddddddd0101		DDDDDDDDDDDDDDDD	

ただし、RRRRR = reg1, wwwww = reg3 です。

ddddddd は、disp23 の下位 7 ビットです。

DDDDDDDDDDDDDDDD は、disp23 の上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　（1）汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスからバイト・データを読み出し、ワード長まで符号拡張し、汎用レジスタ reg2 に格納します。

（2）汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 23 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスからバイト・データを読み出し、ワード長まで符号拡張し、汎用レジスタ reg3 に格納します。

＜ロード命令＞

LD.BU

Load byte unsigned

（符号なし）バイト・データのロード

【命令形式】　（1）LD.BU　disp16 [reg1] , reg2

（2）LD.BU　disp23 [reg1] , reg3

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1
GR[reg2] ← zero-extend (Load-memory (adr, Byte))

（2）adr ← GR[reg1] + sign-extend (disp23) 注 1
GR[reg3] ← zero-extend (Load-memory (adr, Byte))

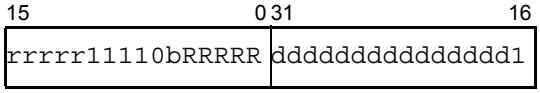
注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VII

（2）Format XIV

【オペコード】

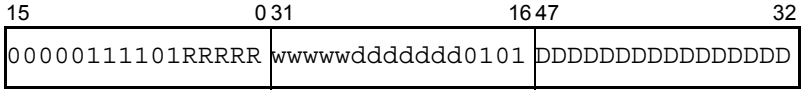
(1)



ただし、ddddddddddddddd は disp16 の上位 15 ビット、b は disp16 のビット 0 です。

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

(2)



ただし、RRRRR = reg1, wwwww = reg3 です。

ddddddd は、disp23 の下位 7 ビットです。

DDDDDDDDDDDDDDDD は、disp23 の上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

(1) 汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスからバイト・データを読み出し、ワード長までゼロ拡張し、汎用レジスタ reg2 に格納します。

(2) 汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 23 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスからバイト・データを読み出し、ワード長までゼロ拡張し、汎用レジスタ reg3 に格納します。

注　意

reg2 には、r0 を指定しないでください。

＜ロード命令＞

LD.DW

Load Double Word

ダブルワード・データのロード

【命令形式】　LD.DW　disp23[reg1], reg3

【オペレーション】　adr ← GR[reg1] + sign-extend (disp23) 注 1

data ← Load-memory (adr, Double-word)

GR[reg3 + 1] || GR[reg3] ← data

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　Format XIV

【オペコード】

15	0 31	16 47	32
00000111101RRRRR	wwwwwddddd01001	DDDDDDDDDDDDDDDD	

ただし、RRRRR = reg1, wwwww = reg3 です。

ddddd は、disp23 の下位側 6-1 ビットです。

DDDDDDDDDDDDDDDD は、disp23 の上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 23 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成した 32 ビット・アドレスからダブルワード・データを読み出し、汎用レジスタ reg3 に下位 32 ビット、reg3 ＋ 1 に上位 32 ビットを格納します。

【補　足】　reg3 は偶数番号のレジスタである必要があります。

注　意

アドレス計算の結果がワード境界の場合は、ミスアライン例外が発生することはありません。

＜ロード命令＞

Load half-word

LD.H

（符号付き）ハーフワード・データのロード

【命令形式】　（1）LD.H　disp16 [reg1] , reg2

（2）LD.H　disp23 [reg1] , reg3

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1
GR[reg2] ← sign-extend (Load-memory (adr, Half-word))

（2）adr ← GR[reg1] + sign-extend (disp23) 注 1
GR[reg3] ← sign-extend (Load-memory (adr, Half-word))

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VII

（2）Format XIV

【オペコード】

(1)

15 … 0	31 … 16
rrrrr111001RRRRR	ddddddddddddddd0

ただし、ddddddddddddddd は disp16 の上位 15 ビットです。

(2)

15 … 0	31 … 16	47 … 32
00000111100RRRRR	wwwwwddddddd00111	DDDDDDDDDDDDDDDD

ただし、RRRRR = reg1, wwwww = reg3 です。

dddddd は、disp23 の下位側ビット 6-1 です。

DDDDDDDDDDDDDDDD は、disp23 の上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

(1) 汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスからハーフワード・データを読み出し、ワード長まで符号拡張し、汎用レジスタ reg2 に格納します。

(2) 汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 23 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスからハーフワード・データを読み出し、ワード長まで符号拡張し、汎用レジスタ reg3 に格納します。

＜ロード命令＞

LD.HU

Load half-word unsigned

（符号なし）ハーフワード・データのロード

【命令形式】　（1）LD.HU　disp16 [reg1] , reg2

（2）LD.HU　disp23 [reg1] , reg3

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1

GR[reg2] ← zero-extend (Load-memory (adr, Half-word))

（2）adr ← GR[reg1] + sign-extend (disp23) 注 1

GR[reg3] ← zero-extend (Load-memory (adr, Half-word))

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VII

（2）Format XIV

【オペコード】

(1)

15 0	31 16
rrrrr111111RRRRR	ddddddddddddddd1

ただし、ddddddddddddddd は disp16 の上位 15 ビットです。

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

(2)

15 0	31 16	47 32
00000111101RRRRR	wwwwwddddddd00111	DDDDDDDDDDDDDDDD

ただし、RRRRR = reg1, wwwww = reg3 です。

dddddd は、disp23 の下位側ビット 6-1 です。

DDDDDDDDDDDDDDDD は、disp23 の上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

(1) 汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成した 32 ビット・アドレスからハーフワード・データを読み出し、ワード長までゼロ拡張し、汎用レジスタ reg2 に格納します。

(2) 汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 23 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスからハーフワード・データを読み出し、ワード長までゼロ拡張し、汎用レジスタ reg3 に格納します。

注　意

reg2 には、r0 を指定しないでください。

＜ロード命令＞

LD.W

Load word

ワード・データのロード

【命令形式】　（1）LD.W　disp16 [reg1] , reg2

（2）LD.W　disp23 [reg1] , reg3

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1

GR[reg2] ← Load-memory (adr, Word)

（2）adr ← GR[reg1] + sign-extend (disp23) 注 1

GR[reg3] ← Load-memory (adr, Word)

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VII

（2）Format XIV

【オペコード】

(1)

15	0	31	16
rrrrr111001RRRRR		ddddddddddddddd1	

ただし、ddddddddddddddd は disp16 の上位 15 ビットです。

(2)

15	0	31	16	47	32
00000111100RRRRR		wwwwwddddddd01001		DDDDDDDDDDDDDDDD	

ただし、RRRRR = reg1, wwwww = reg3 です。

dddddd は、disp23 の下位側ビット 6-1 です。

DDDDDDDDDDDDDDDD は、disp23 の上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

(1) 汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成した 32 ビット・アドレスからワード・データを読み出し、汎用レジスタ reg2 に格納します。

(2) 汎用レジスタ reg1 のワード・データとワード長まで符号拡張した 23 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスからワード・データを読み出し、汎用レジスタ reg3 に格納します。

＜特殊命令＞

Load Linked

LDL.W

アトミックなワード・データ操作を開始するロード

【命令形式】　LDL.W [reg1], reg3

【オペレーション】　adr ← GR[reg1] 注 1

GR[reg3] ← Load-memory (adr, Word)

LLbit ← 1 注 2

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

注 2. 割り込み／例外発生時、および CLL/EIRET/FERET 実行で LLbit ← 0 になります。

【フォーマット】　Format VII

【オペコード】

15	0 31	16
00000111111RRRRR	wwwww01101111000	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　アトミック・リード・モディファイ・ライトのためにメモリからワード・データを読み出し、汎用レジスタ reg3 に格納します。さらに、指定したアドレスを含むアドレス範囲に対応するリンクを生成します。

以降、LDL.W 命令に対応する STC.W 命令が実行される以前に、特定の条件が成立した場合、リンクが消失します。リンクが消失した状態で、STC.W を実行した場合、STC.W の結果は、失敗を示します。

リンクが維持された状態で、STC.W を実行すると、STC.W の結果は成功となり、この場合も、リンクは消失します。

LDL.W 命令と STC.W 命令を使い、マルチコア・システムでのメモリ更新を正確に処理できます。

【補　足】　マルチコア・システムでのアトミック性保証が必要なメモリ更新では、CAXI 命令の代わりに LDL.W 命令と STC.W 命令を使用してください。

＜特殊命令＞

LDSR

Load to system register

システム・レジスタへのロード

【命令形式】　LDSR　reg2, regID, selID

LDSR　reg2, regID

【オペレーション】　SR[regID, selID] ← GR[reg2] 注 1

注 1. アクセス権限によって例外が発生する場合があります。詳細は**「2.5.3　レジスタの更新」**を参照してください。

【フォーマット】　Format IX

【オペコード】

15	0	31	16
rrrrr111111RRRRR		sssss00000100000	

rrrrr：regID，sssss：selID，RRRRR：reg2

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データをシステム・レジスタ番号とグループ番号（regID, selID）で指定されるシステム・レジスタに設定します。汎用レジスタ reg2 は影響を受けません。selID が省略された場合は selID = 0 が指定されたものとします。

【補　足】　命令実行の結果、CPU 動作モードの状態とアクセス対象のシステム・レジスタの組み合わせによっては、PIE 例外、または UCPOP 例外が発生する場合があります。詳細は、**「2.5.3　レジスタの更新」**を参照してください。

注　意

1. この命令では、ニーモニック記述の都合上、ソース・レジスタを汎用レジスタ reg2 としていますが、オペコード上は汎用レジスタ reg1 のフィールドを使用しています。したがって、ニーモニック記述とオペコードにおいて、レジスタ指定の意味がほかの命令と異なります。
2. システム・レジスタ番号、グループ番号は、システム・レジスタを一意に識別するための番号です。未定義レジスタに対する動作は、**「2.5.4　未定義レジスタへの操作」**で定められていますが、推奨しません。

＜ループ命令＞

LOOP

Loop

ループ

【命令形式】　LOOP　reg1,disp16

【オペレーション】　GR[reg1] ← GR[reg1] + (–1) 注 1

if (GR[reg1] != 0)

then

PC ← PC – zero-extend (disp16)

注 1. –1 (0xFFFFFFFF) を加算します。キャリー・フラグの更新は ADD 命令と同一です。

【フォーマット】　Format VII

【オペコード】

15	0	31	16
00000110111RRRRR		ddddddddddddddd1	

ただし、ddddddddddddddd は disp16 の上位 15 ビットです。

【フラグ】

CY	reg1 の演算時に MSB からのキャリーがあれば 1、そうでないとき 0
OV	reg1 の演算時にオーバフローが起こったとき 1、そうでないとき 0
S	reg1 が負のとき 1、そうでないとき 0
Z	reg1 が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg1 の内容から –1 を加算した値で汎用レジスタ reg1 を更新します。更新後の汎用レジスタ reg1 の内容が 0 でなかった場合、次の処理を行います。そうでないときは次の命令に進みます。

- 15 ビット・イミディエイトを 1 ビット論理左シフトしてワード長までゼロ拡張した結果を現在の PC から減算した値を PC に設定し、制御を移します。
- 汎用レジスタ reg1 の内容に対しては、–1（0xFFFFFFFF）を加算します。キャリー・フラグの更新は SUB 命令ではなく、ADD 命令と同一の動作となります。

【補　足】　16 ビット・ディスプレースメントのビット 0 は暗黙的に 0 が使用されます。なお、計算に使用される現在の PC とは、この命令自身の先頭バイトのアドレスであるためディスプレースメント値が 0 のときは、分岐先はこの命令自身になります。

注　意

reg1 には、r0 を指定しないでください。

＜加算付き乗算命令＞

Multiply and add word

MAC

（符号付き）ワード・データの加算付き乗算

【命令形式】　MAC　reg1, reg2, reg3, reg4

【オペレーション】　GR[reg4+1] || GR[reg4] ← GR[reg2] × GR[reg1] + GR[reg3+1] || GR[reg3]

【フォーマット】　Format XI

【オペコード】

15	0 31	16
rrrrr111111RRRRR	wwww0011110mmmm0	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データに汎用レジスタ reg1 のワード・データを乗算した結果（64 ビット・データ）と、汎用レジスタ reg3 を下位 32 ビットとして、汎用レジスタ reg3+1（たとえば、reg3 が r6 の場合、「reg3+1」は r7 となります）を上位 32 ビットとして結合した 64 ビット・データを加算し、その結果（64 ビット・データ）の上位 32 ビットを汎用レジスタ reg4+1 に、下位 32 ビットを汎用レジスタ reg4 に格納します。

汎用レジスタ reg1, reg2 の内容を 32 ビットの符号付き整数として扱います。

汎用レジスタ reg1, reg2, reg3, reg3+1 は影響を受けません。

注　意

reg3、または reg4 に指定できる汎用レジスタは、偶数番号の付いたレジスタ（r0, r2, r4, …, r30）だけです。奇数番号の付いたレジスタ（r1, r3, …, r31）を指定した場合の結果は不定です。

＜加算付き乗算命令＞

MACU

Multiply and add word unsigned

（符号なし）ワード・データの加算付き乗算

【命令形式】　MACU　reg1, reg2, reg3, reg4

【オペレーション】　GR[reg4+1] || GR[reg4] ← GR[reg2] × GR[reg1] + GR[reg3+1] || GR[reg3]

【フォーマット】　Format XI

【オペコード】

15	0	31	16
rrrrr111111RRRRR		wwww0011111mmmm0	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データに汎用レジスタ reg1 のワード・データを乗算した結果（64 ビット・データ）と、汎用レジスタ reg3 を下位 32 ビットとして、汎用レジスタ reg3+1（たとえば、reg3 が r6 の場合、「reg3+1」は r7 となります）を上位 32 ビットとして結合した 64 ビット・データを加算し、その結果（64 ビット・データ）の上位 32 ビットを汎用レジスタ reg4+1 に、下位 32 ビットを汎用レジスタ reg4 に格納します。

汎用レジスタ reg1, reg2 の内容を 32 ビットの符号なし整数として扱います。

汎用レジスタ reg1, reg2, reg3, reg3+1 は影響を受けません。

注　意

reg3、または reg4 に指定できる汎用レジスタは、偶数番号の付いたレジスタ（r0, r2, r4, …, r30）だけです。奇数番号の付いたレジスタ（r1, r3, …, r31）を指定した場合の結果は不定です。

＜算術演算命令＞

MOV

Move register/immediate (5-bit) /immediate (32-bit)

データの転送

【命令形式】　（1）MOV　reg1, reg2

（2）MOV　imm5, reg2

（3）MOV　imm32, reg1

【オペレーション】　（1）GR[reg2] ← GR[reg1]

（2）GR[reg2] ← sign-extend (imm5)

（3）GR[reg1] ← imm32

【フォーマット】　（1）Format I

（2）Format II

（3）Format VI

【オペコード】

(1)

15	0
rrrrr000000RRRRR	

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

(2)

15	0
rrrrr010000iiiii	

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

(3)

15	0	31	16	47	32
00000110001RRRRR		iiiiiiiiiiiiiiii		IIIIIIIIIIIIIIII	

i（ビット 31 ～ 16）は 32 ビット・イミディエイト・データの下位 16 ビットです。

I（ビット 47 ～ 32）は 32 ビット・イミディエイト・データの上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

(1) 汎用レジスタ reg1 のワード・データを、汎用レジスタ reg2 にコピーし転送します。汎用レジスタ reg1 は影響を受けません。

(2) 5 ビット・イミディエイトをワード長まで符号拡張した値を、汎用レジスタ reg2 にコピーし転送します。

(3) 32 ビット・イミディエイトを、汎用レジスタ reg1 にコピーし転送します。

注　意

命令形式（1）の MOV　reg1, reg2 と命令形式（2）の MOV　imm5, reg2 では、reg2 には r0 を指定しないでください。

＜算術演算命令＞

MOVEA

Move effective address

実行アドレスの転送

【命令形式】　MOVEA　imm16, reg1, reg2

【オペレーション】　GR[reg2] ← GR[reg1] + sign-extend (imm16)

【フォーマット】　Format VI

【オペコード】

15	0	31	16
rrrrr110001RRRRR		iiiiiiiiiiiiiiii	

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　汎用レジスタ reg1 のワード・データにワード長まで符号拡張した 16 ビット・イミディエイトを加算し、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。加算によってもフラグは変化しません。

【補　足】　32 ビット・アドレスを計算する際、フラグを変化させたくない場合に、この命令を使用します。

注　意

reg2 には、r0 を指定しないでください。

＜算術演算命令＞

MOVHI

Move high half-word

上位ハーフワードの転送

【命令形式】　MOVHI imm16, reg1, reg2

【オペレーション】　GR[reg2] ← GR[reg1] + (imm16 || 0^{16})

【フォーマット】　Format VI

【オペコード】

15	0	31	16
rrrrr110010RRRRR		iiiiiiiiiiiiiiii	

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　汎用レジスタ reg1 のワード・データに、上位 16 ビットが 16 ビット・イミディエイト、下位 16 ビットが 0 であるワード・データを加算し、その結果を汎用レジスタ reg2 に格納します。

汎用レジスタ reg1 は影響を受けません。加算によってもフラグは変化しません。

【補　足】　32 ビット・アドレスの上位 16 ビットの生成にこの命令を使用します。

注　意

reg2 には、r0 を指定しないでください。

＜乗算命令＞

MUL

Multiply word by register/immediate (9-bit)

（符号付き）ワード・データの乗算

【命令形式】　（1）MUL　reg1, reg2, reg3

（2）MUL　imm9, reg2, reg3

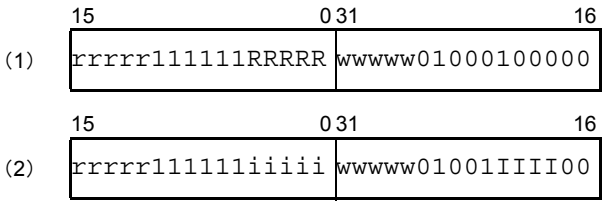
【オペレーション】　（1）GR[reg3] || GR[reg2] ← GR[reg2] × GR[reg1]

（2）GR[reg3] || GR[reg2] ← GR[reg2] × sign-extend (imm9)

【フォーマット】　（1）Format XI

（2）Format XII

【オペコード】



iiiii は、9 ビット・イミディエイト・データの下位 5 ビットです。

IIII は、9 ビット・イミディエイト・データの上位 4 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　（1）汎用レジスタ reg2 のワード・データに汎用レジスタ reg1 のワード・データを乗算し、その結果（64 ビット・データ）の上位 32 ビットを汎用レジスタ reg3 に、下位 32 ビットを汎用レジスタ reg2 に格納します。
reg1, reg2 の内容を 32 ビットの符号付き整数として扱います。汎用レジスタ reg1 は影響を受けません。

（2）汎用レジスタ reg2 のワード・データにワード長まで符号拡張した 9 ビット・イミディエイト・データを乗算し、その結果（64 ビット・データ）の上位 32 ビットを汎用レジスタ reg3 に、下位 32 ビットを汎用レジスタ reg2 に格納します。

【補　足】　汎用レジスタ reg2 と汎用レジスタ reg3 が同じレジスタの場合、そのレジスタには乗算結果の上位 32 ビットが格納されます。

＜乗算命令＞

MULH

Multiply half-word by register/immediate (5-bit)

（符号付き）ハーフワード・データの乗算

【命令形式】　（1）MULH　reg1, reg2

（2）MULH　imm5, reg2

【オペレーション】　（1）GR[reg2] ← GR[reg2] (15:0) × GR[reg1] (15:0)

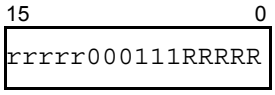
（2）GR[reg2] ← GR[reg2] × sign-extend (imm5)

【フォーマット】　（1）Format I

（2）Format II

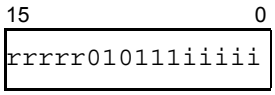
【オペコード】

(1)



rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

(2)



rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　（1）汎用レジスタ reg2 の下位ハーフワード・データに汎用レジスタ reg1 の下位ハーフワード・データを乗算し、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

（2）汎用レジスタ reg2 の下位ハーフワード・データにハーフワード長まで符号拡張した 5 ビット・イミディエイトを乗算し、その結果を汎用レジスタ reg2 に格納します。

【補　足】　乗数、被乗数の場合、汎用レジスタ reg1, reg2 の上位 16 ビットを無視します。

注　意

reg2 には、r0 を指定しないでください。

＜乗算命令＞

MULHI

Multiply half-word by immediate (16-bit)

（符号付き）ハーフワード・イミディエイトの乗算

【命令形式】　MULHI　imm16, reg1, reg2

【オペレーション】　GR[reg2] ← GR[reg1] (15:0) × imm16

【フォーマット】　Format VI

【オペコード】

15	0 31	16
rrrrr110111RRRRR	iiiiiiiiiiiiiiii	

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　汎用レジスタ reg1 の下位ハーフワード・データに、16 ビット・イミディエイトを乗算し、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

【補　足】　被乗数の場合、汎用レジスタ reg1 の上位 16 ビットを無視します。

注　意

reg2 には、r0 を指定しないでください。

＜乗算命令＞

MULU

Multiply word unsigned by register/immediate (9-bit)

（符号なし）ワード・データの乗算

【命令形式】　（1）MULU　reg1, reg2, reg3

（2）MULU　imm9, reg2, reg3

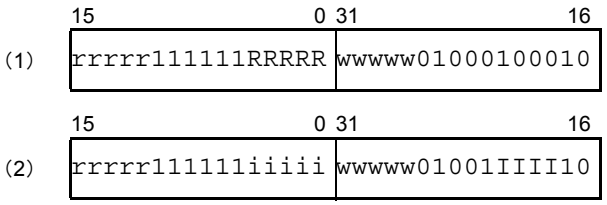
【オペレーション】　（1）GR[reg3] || GR[reg2] ← GR[reg2] × GR[reg1]

（2）GR[reg3] || GR[reg2] ← GR[reg2] × zero-extend (imm9)

【フォーマット】　（1）Format XI

（2）Format XII

【オペコード】



iiiii は、9 ビット・イミディエイト・データの下位 5 ビットです。

IIII は、9 ビット・イミディエイト・データの上位 4 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　（1）汎用レジスタ reg2 のワード・データに汎用レジスタ reg1 のワード・データを乗算し、その結果（64 ビット・データ）の上位 32 ビットを汎用レジスタ reg3 に、下位 32 ビットを汎用レジスタ reg2 に格納します。
汎用レジスタ reg1 は影響を受けません。

（2）汎用レジスタ reg2 のワード・データにワード長までゼロ拡張した 9 ビット・イミディエイト・データを乗算し、その結果（64 ビット・データ）の上位 32 ビットを汎用レジスタ reg3 に、下位 32 ビットを汎用レジスタ reg2 に格納します。

【補　足】　汎用レジスタ reg2 と汎用レジスタ reg3 が同じレジスタの場合、そのレジスタには乗算結果の上位 32 ビットが格納されます。

＜特殊命令＞

NOP

No operation

オペレーションなし

【命令形式】　　NOP

【オペレーション】　　何も行いません。

【フォーマット】　　Format I

【オペコード】

15	0
0000000000000000	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　　何も行わず、次の命令に進みます。

【補　足】　　オペコードは「MOV　r0, r0」と同一になります。

＜論理演算命令＞

NOT

NOT

論理否定（1 の補数をとる）

【命令形式】　NOT reg1, reg2

【オペレーション】　GR[reg2] ← NOT (GR[reg1])

【フォーマット】　Format I

【オペコード】

15	0
rrrrr000001RRRRR	

【フラグ】

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg1 のワード・データの論理否定（1 の補数）をとり、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

＜ビット操作命令＞

NOT1

NOT bit

ビット・ノット

【命令形式】　（1）NOT1　bit#3, disp16 [reg1]

（2）NOT1　reg2, [reg1]

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1
token ← Load-memory (adr, Byte)
Z フラグ ← Not (extract-bit (token, bit#3))
token ← not-bit (token, bit#3)
Store-memory (adr, token, Byte)

（2）adr ← GR[reg1] 注 1
token ← Load-memory (adr, Byte)
Z フラグ ← Not (extract-bit (token, reg2))
token ← not-bit (token, reg2)
Store-memory (adr, token, Byte)

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VIII

（2）Format IX

【オペコード】

(1)

15	0	31	16
01bbb111110RRRRR		ddddddddddddddddd	

(2)

15	0	31	16
rrrrr111111RRRRR		0000000011100010	

【フラグ】

CY	—
OV	—
S	—
Z	指定したビットが 0 のとき 1、指定したビットが 1 のとき 0
SAT	—

【説　明】

(1) まず、汎用レジスタ reg1 のワード・データと、ワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスのバイト・データを読み出し、3 ビットのビット・ナンバで指定されるビットを反転（0 → 1, 1 → 0）し、元のアドレスに書き戻します。
読み出したバイト・データの指定ビットが 0 のとき Z フラグをセット（1）し、指定ビットが 1 のとき Z フラグをクリア（0）します。

(2) まず、汎用レジスタ reg1 のワード・データを読み出して 32 ビット・アドレスを生成します。生成したアドレスのバイト・データを読み出し、汎用レジスタ reg2 の下位 3 ビットで指定されるビットを反転（0 → 1, 1 → 0）し、元のアドレスに書き戻します。
読み出したバイト・データの指定ビットが 0 のとき Z フラグをセット（1）し、指定ビットが 1 のとき Z フラグをクリア（0）します。

【補　足】

PSW の Z フラグはこの命令を実行する前に該当ビットが 0 か 1 だったかを示します。この命令実行後の該当ビットの内容を示すものではありません。

注　意

この命令は排他制御を目的としたアトミック性保証のため、読み出しから書き込みまでの間、対象のアドレスが他の要因によるアクセスによって操作されることはありません。

＜論理演算命令＞

OR

OR

論理和

【命令形式】　　OR　reg1, reg2

【オペレーション】　　GR[reg2] ← GR[reg2] OR GR[reg1]

【フォーマット】　　Format I

【オペコード】

15	0
rrrrr001000RRRRR	

【フラグ】

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　　汎用レジスタ reg2 のワード・データと汎用レジスタ reg1 のワード・データの論理和をとり、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

＜論理演算命令＞

ORI

OR immediate (16-bit)

論理和

【命令形式】　ORI　imm16, reg1, reg2

【オペレーション】　GR[reg2] ← GR[reg1] OR zero-extend (imm16)

【フォーマット】　Format VI

【オペコード】

15	0	31	16
rrrrr110100RRRRR		iiiiiiiiiiiiiiii	

【フラグ】

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg1 のワード・データと 16 ビット・イミディエイトをワード長までゼロ拡張した値の論理和をとり、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

＜特殊命令＞

POPSP

Pop registers from Stack

スタックからのポップ

【命令形式】　　POPSP　rh-rt

【オペレーション】

```
if rh ≦ rt
then cur ← rt
     end ← rh
     tmp ← sp
     while (cur ≧ end) {
     adr ← tmp 注 1, 注 2
     GR[cur] ← Load-memory (adr, Word)
     cur ← cur – 1
     tmp ← tmp + 4
     }
     sp ← tmp
```

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

注 2.adr の下位 2 ビットは 0 にマスクされます。

【フォーマット】　　Format XI

【オペコード】

15	0 31	16
01100111111RRRRR	wwwww00101100000	

RRRRR は、rh です。

wwwww は、rt です。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　　汎用レジスタ rt から rh へ降順（rt, rt － 1, rt － 2,…,rh）にスタックからロードします。指定されたレジスタへすべてロードしたあと、sp を更新（加算）します。

【補　足】

sp で指定された下位 2 ビットのアドレスは、0 にマスクされます。

また、sp の更新前に例外を受け付けると、実行を中止し、戻り番地をこの命令の先頭アドレスとして例外を処理してから、例外処理完了後に再実行します（sp は例外処理実行開始前の元の値を保持します)。

注　意

復帰するレジスタに sp(r3) を含む指定をした場合（rh が 3-31)、sp(r3) にはメモリから読み出してきた値は格納されません。このため、途中で中断した場合にも正しく再実行することが可能です。

＜特殊命令＞

PREPARE

Function prepare

スタック・フレームの生成

【命令形式】　（1）PREPARE　list12, imm5

（2）PREPARE　list12, imm5, sp/imm 注 1

注 1. sp/imm の値は、サブオペコードのビット 19、ビット 20 で指定します。

【オペレーション】　（1）tmp ← sp
foreach (all regs in list12) {
tmp ← tmp － 4
adr ← tmp 注 1, 注 2
Store-memory (adr, GR[reg in list12], Word)
}
sp ← tmp － zero-extend (imm5 logically shift left by 2)

（2）tmp ← sp
foreach (all regs in list12) {
tmp ← tmp － 4
adr ← tmp 注 1, 注 2
Store-memory (adr, GR[reg in list12], Word)
}
sp ← tmp － zero-extend (imm5 logically shift left by 2)

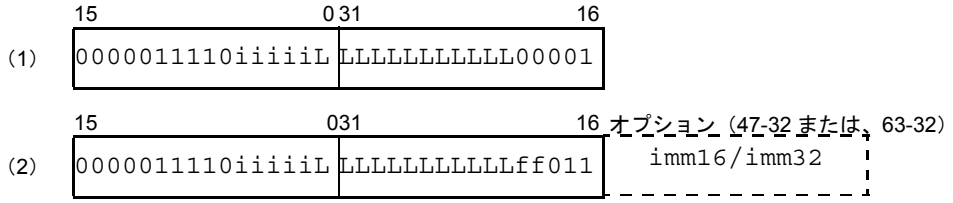
case
ff = 00:　ep ← sp
ff = 01:　ep ← sign-extend (imm16)
ff = 10:　ep ← imm16 logically shift left by 16
ff = 11:　ep ← imm32

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

注 2. adr の下位 2 ビットは 0 にマスクされます。

【フォーマット】　Format XIII

【オペコード】



32 ビット・イミディエイト・データ（imm32）の場合、ビット 47 ～ 32 が imm32 の下位 16 ビット、ビット 63 ～ 48 が imm32 の上位 16 ビットです。

ff = 00 : sp を ep にロード

ff = 01 : 符号拡張した 16 ビット・イミディエイト・データ（ビット 47 ～ 32）を ep にロード

ff = 10 : 16 ビット論理左シフトした 16 ビット・イミディエイト・データ（ビット 47 ～ 32）を ep にロード

ff = 11 : 32 ビット・イミディエイト・データ（ビット 63 ～ 32）を ep にロード

また、LLLLLLLLLLLL は、レジスタ・リスト「list12」の中の対応するビットの値を示します（たとえば、オペコード中のビット 21 の「L」は list12 のビット 21 の値を示します）。

list12 は、次のように定義される 32 ビットのレジスタ・リストです。

31	30	29	28	27	26	25	24	23	22	21	20 ... 1	0
r24	r25	r26	r27	r20	r21	r22	r23	r28	r29	r31	—	r30

ビット 31 ～ 21 とビット 0 の各ビットに汎用レジスタ（r20 ～ r31）が対応しており、セット（1）されたビットに対応するレジスタが操作の対象として指定されます。たとえば、r20, r30 を指定する場合、list12 の値は次のようになります（レジスタが対応付けられていないビット 20 ～ 1 への設定値は任意です）。

- レジスタが対応付けられていないビットの値をすべて 0 とした場合：$0800\ 0001_H$
- レジスタが対応付けられていないビットの値をすべて 1 とした場合：$081F\ FFFF_H$

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

(1) list12 で指定されている汎用レジスタを退避（sp から 4 を減算し、データをそのアドレスに格納）します。次に、2 ビット論理左シフトしワード長までゼロ拡張した 5 ビット・イミディエイトを sp から減算します。

(2) list12 で指定されている汎用レジスタを退避（sp から 4 を減算し、データをそのアドレスに格納）します。次に、2 ビット論理左シフトしワード長までゼロ拡張した 5 ビット・イミディエイトを sp から減算します。
続いて、第 3 オペランド（sp/imm）で指定されるデータを ep にロードします。

【補　足】

list12 の汎用レジスタは、昇順に格納されます（r20, r21, ..., r31）。

imm5 は、自動変数と一時データ用のスタック・フレームを作るために使用されます。

sp で指定された下位 2 ビットのアドレスは、0 でマスクされワード境界にアラインされます。

注　意

命令実行中に例外が発生すると、ライト・サイクルとレジスタ値の書き換えが終了したあとに、命令の実行を中止する場合がありますが、sp は実行開始前の元の値を保持します。そのあと、例外から復帰すると、命令が再実行されます。

＜特殊命令＞

PUSHSP

Push registers to Stack

スタックへのプッシュ

【命令形式】　PUSHSP　rh-rt

【オペレーション】

```
if rh ≦ rt
then cur ← rh
     end ← rt
     tmp ← sp
     while (cur ≦ end) {
     tmp ← tmp – 4
     adr ← tmp 注 1, 注 2
     Store-memory (adr, GR[cur], Word)
     cur ← cur + 1
     }
     sp ← tmp
```

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

注 2. adr の下位 2 ビットは 0 にマスクされます。

【フォーマット】　Format XI

【オペコード】

15	0 31	16
01000111111RRRRR	wwwww00101100000	

RRRRR は、rh です。

wwwww は、rt です。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　汎用レジスタ rh から rt を昇順（rh, rh+1, rh+2,…,rt）にスタックに退避します。指定されたレジスタをすべて退避したあと、sp を更新（減算）します。

【補　足】　sp で指定された下位 2 ビットのアドレスは、0 にマスクされます。
また、sp の更新前に例外を受け付けると、実行を中止し、戻り番地をこの命令の先頭アドレスとして例外を処理してから、例外処理完了後に再実行します（sp は例外処理実行開始前の元の値を保持します)。

＜特殊命令＞

RIE

Reserved Instruction Exception

予約命令例外

【命令形式】　（1）RIE

（2）RIE imm5, imm4

【オペレーション】　FEPC ← PC（復帰 PC）
FEPSW ← PSW
FEIC ← 例外要因コード（$0000\ 0060_H$）
PSW.UM ← 0
PSW.NP ← 1
PSW.EP ← 1
PSW.ID ← 1
PC ← 例外ハンドラ・アドレス（オフセット・アドレス 60_H）

【フォーマット】　（1）Format I

（2）Format X

【オペコード】

(1)

15	0
0000000001000000	

(2)

15	0	31	16
iiiii1111111IIII		0000000000000000	

ただし、iiiii は imm5 です。

IIII は imm4 です。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　復帰 PC（RIE 命令のアドレス）と現在の PSW の内容を、それぞれ FEPC と FEPSW に退避し、例外要因コードを FEIC レジスタに格納、PSW を**「表 4.1　例外要因一覧」**にしたがって更新します。続いて、例外ハンドラ・アドレスに分岐し、例外処理を開始します。

例外ハンドラ・アドレスは、オフセット・アドレス 60_H を元に計算されます。詳細は、**「4.5　例外ハンドラ・アドレス」**を参照してください。

＜データ操作命令＞

ROTL

Rotate Left

ローテート

【命令形式】　（1）ROTL　imm5, reg2, reg3

（2）ROTL　reg1,reg2,reg3

【オペレーション】　（1）GR[reg3] ← GR[reg2] rotate left by zero-extend (imm5)

（2）GR[reg3] ← GR[reg2] rotate left by GR[reg1]

【フォーマット】　Format VII

【オペコード】

(1)

15	0	31	16
rrrrr111111iiiii		wwwww00011000100	

(2)

15	0	31	16
rrrrr111111RRRRR		wwwww00011000110	

【フラグ】

CY	演算結果のビット 0 が 1 のとき 1、そうでないとき 0、ただしローテート量が 0 のときは 0
OV	0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　（1）汎用レジスタ reg2 のワード・データを、ワード長までゼロ拡張した 5 ビット・イミディエイトで示されるシフト数分、左へローテート（回転）し、汎用レジスタ reg3 に書き込みます。汎用レジスタ reg2 は影響を受けません。

（2）汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位 5 ビットで示されるシフト数分、左へローテート（回転）し、汎用レジスタ reg3 に書き込みます。汎用レジスタ reg1, reg2 は影響を受けません。

＜データ操作命令＞

SAR

Shift arithmetic right by register/immediate (5-bit)

算術右シフト

【命令形式】　（1）SAR　reg1, reg2

（2）SAR　imm5, reg2

（3）SAR　reg1, reg2, reg3

【オペレーション】　（1）GR[reg2] ← GR[reg2] arithmetically shift right by GR[reg1]

（2）GR[reg2] ← GR[reg2] arithmetically shift right by zero-extend (imm5)

（3）GR[reg3] ← GR[reg2] arithmetically shift right by GR[reg1]

【フォーマット】　（1）Format IX

（2）Format II

（3）Format XI

【オペコード】

(1)

15	0	31	16
rrrrr111111RRRRR		0000000010100000	

(2)

15	0
rrrrr010101iiiii	

(3)

15	0	31	16
rrrrr111111RRRRR		wwwww00010100010	

【フラグ】

CY	最後にシフト・アウトしたビットが 1 のとき 1、そうでないとき 0、ただしシフト数が 0 のときは 0
OV	0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】

(1) 汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位 5 ビットで示されるシフト数分、0 から +31 までを算術右シフトし（シフト以前の MSB の値を、シフトを実行したあとの MSB にコピーする）、汎用レジスタ reg2 に書き込みます。汎用レジスタ reg1 は影響を受けません。

(2) 汎用レジスタ reg2 のワード・データを、ワード長までゼロ拡張した 5 ビット・イミディエイトで示されるシフト数分、0 から +31 までを算術右シフトし（シフト以前の MSB の値を、シフトを実行したあとの MSB にコピーする）、汎用レジスタ reg2 に書き込みます。

(3) 汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位 5 ビットで示されるシフト数分、0 から +31 までを算術右シフトし（シフト以前の MSB の値を、シフトを実行したあとの MSB にコピーする）、汎用レジスタ reg3 に書き込みます。汎用レジスタ reg1, reg2 は影響を受けません。

＜データ操作命令＞

SASF

Shift and set flag condition

シフトとフラグ条件の設定

【命令形式】　SASF　cccc, reg2

【オペレーション】　if conditions are satisfied

then GR[reg2] ← (GR[reg2] Logically shift left by 1) OR 0000 0001_H

else GR[reg2] ← (GR[reg2] Logically shift left by 1) OR 0000 0000_H

【フォーマット】　Format IX

【オペコード】

15	0 31	16
rrrrr1111110cccc	0000001000000000	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　条件コード「cccc」で指定された条件が満たされた場合は、汎用レジスタ reg2 のデータを 1 ビット論理左シフトし、LSB がセット（1）されます。満たされなかった場合は、汎用レジスタ reg2 のデータを 1 ビット論理左シフトし、LSB がクリア（0）されます。

次の表で示されている条件コードのうちの 1 つを「cccc」として指定してください。

条件コード	条件名	条件式	条件コード	条件名	条件式
0000	V	OV = 1	0100	S/N	S = 1
1000	NV	OV = 0	1100	NS/P	S = 0
0001	C/L	CY = 1	0101	T	always（無条件）
1001	NC/NL	CY = 0	1101	SA	SAT = 1
0010	Z	Z = 1	0110	LT	(S xor OV) = 1
1010	NZ	Z = 0	1110	GE	(S xor OV) = 0
0011	NH	(CY or Z) = 1	0111	LE	((S xor OV) or Z) = 1
1011	H	(CY or Z) = 0	1111	GT	((S xor OV) or Z) = 0

【補　足】　SETF 命令を参照してください。

＜飽和演算命令＞

SATADD

Saturated add register/immediate (5-bit)

飽和加算

【命令形式】　（1）SATADD　reg1, reg2

（2）SATADD　imm5, reg2

（3）SATADD　reg1, reg2, reg3

【オペレーション】　（1）GR[reg2] ← saturated (GR[reg2] + GR[reg1])

（2）GR[reg2] ← saturated (GR[reg2] + sign-extend (imm5))

（3）GR[reg3] ← saturated (GR[reg2] + GR[reg1])

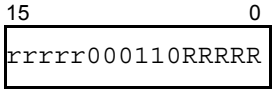
【フォーマット】　（1）Format I

（2）Format II

（3）Format XI

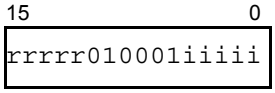
【オペコード】

(1)



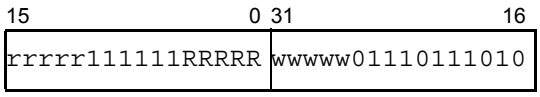
rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

(2)



rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

(3)



【フラグ】

CY	MSB からのキャリーがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	飽和演算結果が負のとき 1、そうでないとき 0
Z	飽和演算結果が 0 のとき 1、そうでないとき 0
SAT	OV = 1 であるとき 1、そうでないとき変化しない

【説　明】

(1) 汎用レジスタ reg2 のワード・データに汎用レジスタ reg1 のワード・データを加算し、その結果を汎用レジスタ reg2 に格納します。ただし、結果が正の最大値 7FFF FFFF$_H$ を越えたときは 7FFF FFFF$_H$ を、負の最大値 8000 0000$_H$ を越えたときは 8000 0000$_H$ を reg2 に格納し、SAT フラグをセット（1）します。汎用レジスタ reg1 は影響を受けません。

(2) 汎用レジスタ reg2 のワード・データにワード長まで符号拡張した 5 ビット・イミディエイトを加算し、その結果を汎用レジスタ reg2 に格納します。ただし、結果が正の最大値 7FFF FFFF$_H$ を越えたときは 7FFF FFFF$_H$ を、負の最大値 8000 0000$_H$ を越えたときは 8000 0000$_H$ を reg2 に格納し、SAT フラグをセット（1）します。

(3) 汎用レジスタ reg2 のワード・データに汎用レジスタ reg1 のワード・データを加算し、その結果を汎用レジスタ reg3 に格納します。ただし、結果が正の最大値 7FFF FFFF$_H$ を越えたときは 7FFF FFFF$_H$ を、負の最大値 8000 0000$_H$ を越えたときは 8000 0000$_H$ を reg3 に格納し、SAT フラグをセット（1）します。汎用レジスタ reg1 と reg2 は影響を受けません。

【補　足】

SAT フラグは累積フラグであり、飽和演算命令で演算結果が飽和するとセット（1）され、以降の命令の演算結果が飽和しなくてもクリア（0）されません。

SAT フラグがセット（1）されていても、飽和演算命令は正常に実行します。

注　意

1. SAT フラグをクリア（0）するときは、LDSR 命令によって PSW にデータをロードしてください
2. 命令形式（1）SATADD　reg1, reg2 と命令形式（2）の SATADD　imm5, reg2 では、reg2 には r0 を指定しないでください。

＜飽和演算命令＞

SATSUB

Saturated subtract

飽和減算

【命令形式】　（1）SATSUB　reg1, reg2

（2）SATSUB　reg1, reg2, reg3

【オペレーション】　（1）GR[reg2] ← saturated (GR[reg2] – GR[reg1])

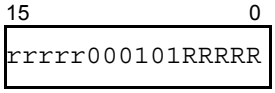
（2）GR[reg3] ← saturated (GR[reg2] – GR[reg1])

【フォーマット】　（1）Format I

（2）Format XI

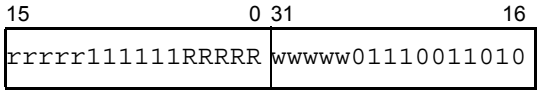
【オペコード】

(1)



rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

(2)



【フラグ】

CY	MSB へのボローがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	飽和演算結果が負のとき 1、そうでないとき 0
Z	飽和演算結果が 0 のとき 1、そうでないとき 0
SAT	OV = 1 であるとき 1、そうでないとき変化しない

【説　明】　（1）汎用レジスタ reg2 のワード・データから汎用レジスタ reg1 のワード・データを減算し、その結果を汎用レジスタ reg2 に格納します。ただし、結果が正の最大値 7FFF FFFF$_H$ を越えたときは 7FFF FFFF$_H$ を、負の最大値 8000 0000$_H$ を越えたときは 8000 0000$_H$ を reg2 に格納し、SAT フラグをセット（1）します。汎用レジスタ reg1 は影響を受けません。

（2）汎用レジスタ reg2 のワード・データから汎用レジスタ reg1 のワード・データを減算し、その結果を汎用レジスタ reg3 に格納します。ただし、結果が正の最大値 7FFF FFFF$_H$ を越えたときは 7FFF FFFF$_H$ を、負の最大値 8000 0000$_H$ を越えたときは 8000 0000$_H$ を reg3 に格納し、SAT フラグをセット（1）します。汎用レジスタ reg1 と reg2 は影響を受けません。

【補　足】

SAT フラグは累積フラグであり、飽和演算命令で演算結果が飽和するとセット（1）され、以降の命令の演算結果が飽和しなくてもクリア（0）されません。

SAT フラグがセット（1）されていても、飽和演算命令は正常に実行します。

注　意

1. SAT フラグをクリア（0）するときは、LDSR 命令によって PSW にデータをロードしてください。
2. 命令形式（1）の SATSUB　reg1, reg2 では、reg2 には r0 を指定しないでください。

＜飽和演算命令＞

SATSUBI

Saturated subtract immediate

飽和減算

【命令形式】　SATSUBI　imm16, reg1, reg2

【オペレーション】　GR[reg2] ← saturated (GR[reg1] － sign-extend (imm16))

【フォーマット】　Format VI

【オペコード】

15	0	31	16
rrrrr110011RRRRR		iiiiiiiiiiiiiiii	

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

【フラグ】

CY	MSB へのボローがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	飽和演算結果が負のとき 1、そうでないとき 0
Z	飽和演算結果が 0 のとき 1、そうでないとき 0
SAT	OV = 1 であるとき 1、そうでないとき変化しない

【説　明】　汎用レジスタ reg1 のワード・データからワード長まで符号拡張した 16 ビット・イミディエイトを減算し、その結果を汎用レジスタ reg2 に格納します。ただし、結果が正の最大値 7FFF FFFF$_H$ を越えたときは 7FFF FFFF$_H$ を、負の最大値 8000 0000$_H$ を越えたときは 8000 0000$_H$ を reg2 に格納し、SAT フラグをセット（1）します。汎用レジスタ reg1 は影響を受けません。

【補　足】　SAT フラグは累積フラグであり、飽和演算命令で演算結果が飽和するとセット（1）され、以降の命令の演算結果が飽和しなくてもクリア（0）されません。

SAT フラグがセット（1）されていても、飽和演算命令は正常に実行します。

注　意

1. SAT フラグをクリア（0）するときは、LDSR 命令によって PSW にデータをロードしてください。
2. reg2 には、r0 を指定しないでください。

＜飽和演算命令＞

SATSUBR

Saturated subtract reverse

飽和逆減算

【命令形式】　SATSUBR　reg1, reg2

【オペレーション】　GR[reg2] ← saturated (GR[reg1] － GR[reg2])

【フォーマット】　Format I

【オペコード】

15	0
rrrrr000100RRRRR	

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

【フラグ】

CY	MSB へのボローがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	飽和演算結果が負のとき 1、そうでないとき 0
Z	飽和演算結果が 0 のとき 1、そうでないとき 0
SAT	OV = 1 であるとき 1、そうでないとき変化しない

【説　明】　汎用レジスタ reg1 のワード・データから汎用レジスタ reg2 のワード・データを減算し、その結果を汎用レジスタ reg2 に格納します。ただし、結果が正の最大値 7FFF FFFF$_H$ を越えたときは 7FFF FFFF$_H$ を、負の最大値 8000 0000$_H$ を越えたときは 8000 0000$_H$ を reg2 に格納し、SAT フラグをセット（1）します。汎用レジスタ reg1 は影響を受けません。

【補　足】　SAT フラグは累積フラグであり、飽和演算命令で演算結果が飽和するとセット（1）され、以降の命令の演算結果が飽和しなくてもクリア（0）されません。

SAT フラグがセット（1）されていても、飽和演算命令は正常に実行します。

注　意

1. SAT フラグをクリア（0）するときは、LDSR 命令によって PSW にデータをロードしてください。
2. reg2 には、r0 を指定しないでください。

＜条件付き演算命令＞

Subtract on condition flag

SBF

条件付き減算

【命令形式】　SBF　cccc, reg1, reg2, reg3

【オペレーション】　if conditions are satisfied
then GR[reg3] ← GR[reg2] － GR[reg1] –1
else GR[reg3] ← GR[reg2] － GR[reg1] –0

【フォーマット】　Format XI

【オペコード】

15	0	31	16
rrrrr111111RRRRR		wwwww011100cccc0	

【フラグ】

CY	MSB へのボローがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　条件コード「cccc」で指定された条件が満たされた場合は、汎用レジスタ reg2 のワード・データから汎用レジスタ reg1 のワード・データを減算した結果から、1 を減算し、その結果を汎用レジスタ reg3 に格納します。

条件コード「cccc」で指定された条件が満たされなかった場合は、汎用レジスタ reg2 のワード・データから汎用レジスタ reg1 のワード・データを減算し、その結果を汎用レジスタ reg3 に格納します。

汎用レジスタ reg1, reg2 は影響を受けません。

次の表で示されている条件コードのうちの 1 つを「cccc」として指定してください（ただし、cccc ≠ 1101）。

条件コード	条件名	条件式
0000	V	OV = 1
1000	NV	OV = 0
0001	C/L	CY = 1
1001	NC/NL	CY = 0
0010	Z	Z = 1
1010	NZ	Z = 0
0011	NH	(CY or Z) = 1
1011	H	(CY or Z) = 0

条件コード	条件名	条件式
0100	S/N	S = 1
1100	NS/P	S = 0
0101	T	always（無条件）
0110	LT	(S xor OV) = 1
1110	GE	(S xor OV) = 0
0111	LE	((S xor OV) or Z) = 1
1111	GT	((S xor OV) or Z) = 0
（1101）	設定禁止	

＜ビット・サーチ命令＞

SCH0L

Search zero from left

MSB 側からのビット（0）検索

【命令形式】　SCH0L　reg2, reg3

【オペレーション】　GR[reg3] ← search zero from left of GR[reg2]

【フォーマット】　Format IX

【オペコード】

15	0	31	16
rrrrr11111100000		wwwww01101100100	

【フラグ】

CY	最後にビット（0）が見つかったとき 1、そうでないとき 0
OV	0
S	0
Z	ビット（0）が見つからなかったとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データを左側（MSB 側）から検索し、最初に 0 が見つかったビット位置（0 ～ 31）までの 1 の個数 +1 を汎用レジスタ reg3 に書き込みます（たとえば、reg2 のビット 31 が 0 の場合は、reg3 に 01_H を書き込みます）。

ビット（0）が見つからなかった場合は、reg3 に 0 を書き込み、同時に Z フラグをセット（1）します。見つかったビット（0）が LSB だった場合は CY フラグをセット（1）します。

＜ビット・サーチ命令＞

SCH0R

Search zero from right

LSB 側からのビット（0）検索

【命令形式】　SCH0R　reg2, reg3

【オペレーション】　GR[reg3] ← search zero from right of GR[reg2]

【フォーマット】　Format IX

【オペコード】

15	0	31	16
rrrrr11111100000		wwwww01101100000	

【フラグ】

CY	最後にビット（0）が見つかったとき 1、そうでないとき 0
OV	0
S	0
Z	ビット（0）が見つからなかったとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データを右側（LSB 側）から検索し、最初に 0 が見つかったビット位置（0 ～ 31）までの 1 の個数 +1 を汎用レジスタ reg3 に書き込みます（たとえば、reg2 のビット 0 が 0 の場合は、reg3 に 01_H を書き込みます）。

ビット（0）が見つからなかった場合は、reg3 に 0 を書き込み、同時に Z フラグをセット（1）します。見つかったビット（0）が MSB だった場合は CY フラグをセット（1）します。

＜ビット・サーチ命令＞

SCH1L

Search one from left

MSB 側からのビット（1）検索

【命令形式】　SCH1L　reg2, reg3

【オペレーション】　GR[reg3] ← search one from left of GR[reg2]

【フォーマット】　Format IX

【オペコード】

15	0	31	16
rrrrr11111100000		wwwww01101100110	

【フラグ】

CY	最後にビット（1）が見つかったとき 1、そうでないとき 0
OV	0
S	0
Z	ビット（1）が見つからなかったとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データを左側（MSB 側）から検索し、最初に 1 が見つかったビット位置（0 ～ 31）までの 0 の個数 +1 を汎用レジスタ reg3 に書き込みます（たとえば、reg2 のビット 31 が 1 の場合は、reg3 に 01_H を書き込みます）。

ビット（1）が見つからなかった場合は、reg3 に 0 を書き込み、同時に Z フラグをセット（1）します。見つかったビット（1）が LSB だった場合は CY フラグをセット（1）します。

＜ビット・サーチ命令＞

SCH1R

Search one from right

LSB 側からのビット（1）検索

【命令形式】　SCH1R　reg2, reg3

【オペレーション】　GR[reg3] ← search one from right of GR[reg2]

【フォーマット】　Format IX

【オペコード】

15	0 31	16
rrrrr11111100000	wwwww01101100010	

【フラグ】

CY	最後にビット（1）が見つかったとき 1、そうでないとき 0
OV	0
S	0
Z	ビット（1）が見つからなかったとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データを右側（LSB 側）から検索し、最初に 1 が見つかったビット位置（0 ～ 31）までの 0 の個数 +1 を汎用レジスタ reg3 に書き込みます（たとえば、reg2 のビット 0 が 1 の場合は、reg3 に 01_H を書き込みます）。

ビット（1）が見つからなかった場合は、reg3 に 0 を書き込み、同時に Z フラグをセット（1）します。見つかったビット（1）が MSB だった場合は CY フラグをセット（1）します。

＜ビット操作命令＞

SET1

Set bit

ビット・セット

【命令形式】　（1）SET1　bit#3, disp16 [reg1]

（2）SET1　reg2, [reg1]

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1
token ← Load-memory (adr, Byte)
Z フラグ ← Not (extract-bit (token, bit#3))
token ← set-bit (token, bit#3)
Store-memory (adr, token, Byte)

（2）adr ← GR[reg1] 注 1
token ← Load-memory (adr, Byte)
Z フラグ ← Not (extract-bit (token, reg2))
token ← set-bit (token, reg2)
Store-memory (adr, token, Byte)

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VIII

（2）Format IX

【オペコード】

(1)

15 … 0	31 … 16
00bbb111110RRRRR	ddddddddddddddddd

(2)

15 … 0	31 … 16
rrrrr111111RRRRR	0000000011100000

【フラグ】

CY	—
OV	—
S	—
Z	指定したビットが 0 のとき 1、指定したビットが 1 のとき 0
SAT	—

【説　明】

(1) まず、汎用レジスタ reg1 のワード・データと、ワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスのバイト・データを読み出し、3 ビットのビット・ナンバで指定されるビットをセット（1）し、元のアドレスに書き戻します。
読み出したバイト・データの指定ビットが 0 のとき Z フラグをセット（1）し、指定ビットが 1 のとき Z フラグをクリア（0）します。

(2) まず、汎用レジスタ reg1 のワード・データを読み出して 32 ビット・アドレスを生成します。生成したアドレスのバイト・データを読み出し、汎用レジスタ reg2 の下位 3 ビットで指定されるビットをセット（1）し、元のアドレスに書き戻します。
読み出したバイト・データの指定ビットが 0 のとき Z フラグをセット（1）し、指定ビットが 1 のとき Z フラグをクリア（0）します。

【補　足】

PSW の Z フラグはこの命令を実行する前に該当ビットが 0 か 1 だったかを示します。この命令実行後の該当ビットの内容を示すものではありません。

注　意

この命令は排他制御を目的としたアトミック性保証のため、読み出しから書き込みまでの間、対象のアドレスが他の要因によるアクセスによって操作されることはありません。

＜データ操作命令＞

SETF

Set flag condition

フラグ条件の設定

【命令形式】　SETF　cccc, reg2

【オペレーション】　if conditions are satisfied
then GR[reg2] ← 0000 0001$_H$
else GR[reg2] ← 0000 0000$_H$

【フォーマット】　Format IX

【オペコード】

15	031	16
rrrrr1111110cccc	0000000000000000	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　条件コード「cccc」の示す条件が満たされた場合、汎用レジスタ reg2 に 1 を、そうでない場合は 0 を格納します。

次の表で示されている条件コードのうちの 1 つを「cccc」として指定してください。

条件コード	条件名	条件式	条件コード	条件名	条件式
0000	V	OV = 1	0100	S/N	S = 1
1000	NV	OV = 0	1100	NS/P	S = 0
0001	C/L	CY = 1	0101	T	always（無条件）
1001	NC/NL	CY = 0	1101	SA	SAT = 1
0010	Z	Z = 1	0110	LT	(S xor OV) = 1
1010	NZ	Z = 0	1110	GE	(S xor OV) = 0
0011	NH	(CY or Z) = 1	0111	LE	((S xor OV) or Z) = 1
1011	H	(CY or Z) = 0	1111	GT	((S xor OV) or Z) = 0

【補　足】　この命令の利用方法の例を示します。

(1) 複数の条件節の翻訳

C 言語での if（A）という文において、A が複数の条件節（a1, a2, a3, …）から成り立つとき、通常は if（a1）then, if（a2）then というシーケンスに翻訳します。オブジェクト・コードでは an に相当する評価の結果を見て「条件分岐」をします。パイプライン・プロセッサでは「条件判断 + 分岐」は通常の演算に比べて遅いので、おのおのの条件節を評価した結果 if（an）の結果をレジスタ Ra に覚えておきます。すべての条件節を評価し終わったあとに Ran をまとめて論理演算することで、パイプラインによる遅れを回避できます。

(2) 倍長演算

Add with Carry のような倍長演算をするときに、CY フラグの結果を汎用レジスタ reg2 に格納できるため、下位からの桁上がりを数値として表現できます。

＜データ操作命令＞

SHL

Shift logical left by register/immediate (5-bit)

論理左シフト

【命令形式】　（1）SHL　reg1, reg2

（2）SHL　imm5, reg2

（3）SHL　reg1, reg2, reg3

【オペレーション】　（1）GR[reg2] ← GR[reg2] logically shift left by GR[reg1]

（2）GR[reg2] ← GR[reg2] logically shift left by zero-extend (imm5)

（3）GR[reg3] ← GR[reg2] logically shift left by GR[reg1]

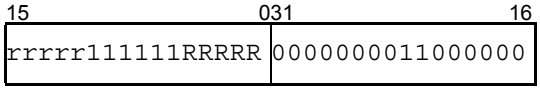
【フォーマット】　（1）Format IX

（2）Format II

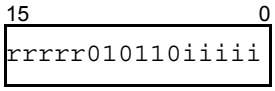
（3）Format XI

【オペコード】

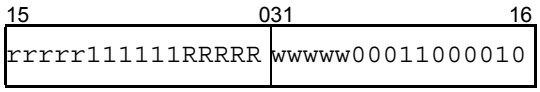
(1)



(2)



(3)



【フラグ】

CY	最後にシフト・アウトしたビットが 1 のとき 1、そうでないとき 0、ただしシフト数が 0 のときは 0
OV	0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】

(1) 汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位 5 ビットで示されるシフト数分、0 から +31 までを論理左シフトし（LSB 側に 0 を送り込む)、汎用レジ スタ reg2 に書き込みます。汎用レジスタ reg1 は影響を受けません。

(2) 汎用レジスタ reg2 のワード・データを、ワード長までゼロ拡張した 5 ビット・イミディエイトで示されるシフト数分、0 から +31 までを論理左シフトし（LSB 側に 0 を送り込む)、汎用レジスタ reg2 に書き込みます。

(3) 汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位 5 ビットで示されるシフト数分、0 から +31 までを論理左シフトし（LSB 側に 0 を送り込む)、汎用レジスタ reg3 に書き込みます。汎用レジスタ reg1, reg2 は影響を受けません。

＜データ操作命令＞

SHR

Shift logical right by register/immediate (5-bit)

論理右シフト

【命令形式】　（1）SHR　reg1, reg2

（2）SHR　imm5, reg2

（3）SHR　reg1, reg2, reg3

【オペレーション】　（1）GR[reg2] ← GR[reg2] logically shift right by GR[reg1]

（2）GR[reg2] ← GR[reg2] logically shift right by zero-extend (imm5)

（3）GR[reg3] ← GR[reg2] logically shift right by GR[reg1]

【フォーマット】　（1）Format IX

（2）Format II

（3）Format XI

【オペコード】

(1)

15	0	31	16
rrrrr111111RRRRR		0000000010000000	

(2)

15	0
rrrrr010100iiiii	

(3)

15	0	31	16
rrrrr111111RRRRR		wwwww00010000010	

【フラグ】

CY	最後にシフト・アウトしたビットが 1 のとき 1、そうでないとき 0、ただしシフト数が 0 のときは 0
OV	0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】

(1) 汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位 5 ビットで示されるシフト数分、0 から +31 までを論理右シフトし（MSB 側に 0 を送り込む）、汎用レジスタ reg2 に書き込みます。汎用レジスタ reg1 は影響を受けません。

(2) 汎用レジスタ reg2 のワード・データを、ワード長までゼロ拡張した 5 ビット・イミディエイトで示されるシフト数分、0 から +31 までを論理右シフトし（MSB 側に 0 を送り込む）、汎用レジスタ reg2 に書き込みます。

(3) 汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位 5 ビットで示されるシフト数分、0 から +31 までを論理右シフトし（MSB 側に 0 を送り込む）、汎用レジスタ reg3 に書き込みます。汎用レジスタ reg1, reg2 は影響を受けません。

＜ロード命令＞

SLD.B

Short format load byte

（符号付き）バイト・データのロード

【命令形式】　SLD.B　disp7 [ep] , reg2

【オペレーション】　adr ← ep + zero-extend (disp7) 注 1

GR[reg2] ← sign-extend (Load-memory (adr, Byte))

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　Format IV

【オペコード】

15	0
rrrrr0110ddddddd	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　エレメント・ポインタと、ワード長までゼロ拡張した 7 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスからバイト・データを読み出し、ワード長まで符号拡張し、reg2 に格納します。

＜ロード命令＞

SLD.BU

Short format load byte unsigned

（符号なし）バイト・データのロード

【命令形式】　SLD.BU　disp4 [ep] , reg2

【オペレーション】　adr ← ep + zero-extend (disp4) 注 1
GR[reg2] ← zero-extend (Load-memory (adr, Byte))

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　Format IV

【オペコード】

15	0
rrrrr0000110dddd	

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　エレメント・ポインタと、ワード長までゼロ拡張した 4 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成したアドレスからバイト・データを読み出し、ワード長までゼロ拡張し、reg2 に格納します。

注　意

reg2 には、r0 を指定しないでください。

＜ロード命令＞

SLD.H

Short format load half-word

（符号付き）ハーフワード・データのロード

【命令形式】　SLD.H　disp8 [ep] , reg2

【オペレーション】　adr ← ep + zero-extend (disp8) 注 1
GR[reg2] ← sign-extend (Load-memory (adr, Half-word))

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　Format IV

【オペコード】

15	0
rrrrr1000ddddddd	

ただし、dddddddは disp8 の上位 7 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　エレメント・ポインタと、ワード長までゼロ拡張した 8 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成した 32 ビット・アドレスからハーフワード・データを読み出し、ワード長まで符号拡張し、reg2 に格納します。

＜ロード命令＞

SLD.HU

Short format load half-word unsigned

（符号なし）ハーフワード・データのロード

【命令形式】　SLD.HU　disp5 [ep] , reg2

【オペレーション】　adr ← ep + zero-extend (disp5) 注 1
GR[reg2] ← zero-extend (Load-memory (adr, Half-word))

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　Format IV

【オペコード】

15	0
rrrrr0000111dddd	

rrrrr ≠ 00000（reg2 には r0 を設定しないでください）

ただし、dddd は disp5 の上位 4 ビット

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　エレメント・ポインタと、ワード長までゼロ拡張した 5 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成した 32 ビット・アドレスからハーフワード・データを読み出し、ワード長までゼロ拡張し、reg2 に格納します。

注　意

reg2 には r0 を指定しないでください。

＜ロード命令＞

SLD.W

Short format load word

ワード・データのロード

【命令形式】　SLD.W　disp8 [ep] , reg2

【オペレーション】　adr ← ep + zero-extend (disp8) 注 1
GR[reg2] ← Load-memory (adr, Word)

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　Format IV

【オペコード】

15	0
rrrrr1010ddddddd0	

ただし、dddddd は disp8 の上位 6 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　エレメント・ポインタと、ワード長までゼロ拡張した 8 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。生成した 32 ビット・アドレスからワード・データを読み出し、reg2 に格納します。

＜特殊命令＞

SNOOZE

Snooze

スヌーズ

【命令形式】　SNOOZE

【オペレーション】　一定期間の間、一時停止します。

【フォーマット】　Format X

【オペコード】

15	031	16
0000111111100000	0000000100100000	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　ハードウェア仕様によって定義された期間または特定の状態時に、SNOOZE 命令を実行した CPU コアの動作を一時停止させます。

定義された期間の経過または状態遷移時に自動的に復帰し、次命令から実行を開始します。

一時停止状態の解除条件は次のとおりです。

- 定義された一定期間の経過
- すべての中断型例外の発生

また、上記の例外の受け付け条件（ID および NP の値）を満たしていない場合であっても、要求が存在する場合には一時停止状態の解除が行われます（例：PSW.ID = 1 であっても、INT0 が発生した段階で停止状態が解除されます）。

機能ごとに定義された次のマスク機能によって、中断型例外の発生がマスクされている場合は、一時停止状態は解除されません。

- 割り込みコントローラによる割り込みチャネルのマスク注 1
- 浮動小数点演算の例外許可ビットによるマスク
- 上記以外のハードウェア機能で定義されたマスク

注 1. ISPR レジスタ、PMR レジスタによるマスクは含まれません。

【補　足】　スピンロック時のバス帯域占有によるマルチコア・パフォーマンス低下を防止するための命令です。

注　意

SNOOZE 命令による一時停止期間は、CPU コアのハードウェア仕様によって定められます。詳細は製品のハードウェアマニュアルを参照してください。

＜ストア命令＞

SST.B

Short format store byte

バイト・データのストア

【命令形式】　SST.B　reg2, disp7 [ep]

【オペレーション】　adr ← ep + zero-extend (disp7) 注 1
Store-memory (adr, GR[reg2] , Byte)

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　Format IV

【オペコード】

15	0
rrrrr0111ddddddd	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　エレメント・ポインタと、ワード長までゼロ拡張した 7 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。reg2 の最下位バイト・データを生成したアドレスに格納します。

＜ストア命令＞

SST.H

Short format store half-word

ハーフワード・データのストア

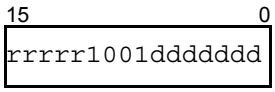
【命令形式】　SST.H　reg2, disp8 [ep]

【オペレーション】　adr ← ep + zero-extend (disp8) 注 1
Store-memory (adr, GR[reg2] , Half-word)

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　Format IV

【オペコード】



ただし、dddddddは disp8 の上位 7 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　エレメント・ポインタと、ワード長までゼロ拡張した 8 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。reg2 の下位ハーフワード・データを生成した 32 ビット・アドレスに格納します。

＜ストア命令＞

SST.W

Short format store word

ワード・データのストア

【命令形式】　SST.W　reg2, disp8 [ep]

【オペレーション】　adr ← ep + zero-extend (disp8) 注 1
Store-memory (adr, GR[reg2], Word)

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　Format IV

【オペコード】

15															0
r	r	r	r	r	1	0	1	0	d	d	d	d	d	d	1

ただし、dddddd は disp8 の上位 6 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　エレメント・ポインタと、ワード長までゼロ拡張した 8 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。reg2 のワード・データを生成した 32 ビット・アドレスに格納します。

<ストア命令>

ST.B

Store byte

バイト・データのストア

【命令形式】　（1）ST.B　reg2, disp16 [reg1]

（2）ST.B　reg3, disp23 [reg1]

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1
Store-memory (adr, GR[reg2], Byte)

（2）adr ← GR[reg1] + sign-extend (disp23) 注 1
Store-memory (adr, GR[reg3], Byte)

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VII

（2）Format XIV

【オペコード】

(1)

15	0	31	16
rrrrr111010RRRRR		ddddddddddddddddd	

(2)

15	0	31	16	47	32
00000111100RRRRR		wwwwwddddddd1101		DDDDDDDDDDDDDDDD	

ただし、RRRRR = reg1, wwwww = reg3 です。

ddddddd は、disp23 の下位 7 ビットです。

DDDDDDDDDDDDDDDD は、disp23 の上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　（1）汎用レジスタ reg1 のデータと、ワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。汎用レジスタ reg2 の最下位のバイト・データを生成したアドレスに格納します。

（2）汎用レジスタ reg1 のデータと、ワード長まで符号拡張した 23 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。汎用レジスタ reg3 の最下位のバイト・データを生成したアドレスに格納します。

＜ストア命令＞

ST.DW

Store Double Word

ダブルワード・データのストア

【命令形式】　ST.DW　reg3, disp23[reg1]

【オペレーション】　adr ← GR[reg1] + sign-extend (disp23) 注 1
data ← GR[reg3+1] || GR[reg3]
Store-memory (adr, data, Double-word)

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　Format XIV

【オペコード】

15	031	1647	32
00000111101RRRRR	wwwwwddddddd01111	DDDDDDDDDDDDDDDD	

ただし、RRRRR = reg1, wwwww = reg3 です。

dddddd は、disp23 の下位側 6 ～ 1 ビットです。

DDDDDDDDDDDDDDDD は、disp23 の上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　汎用レジスタ reg1 のデータと、ワード長まで符号拡張した 23 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。汎用レジスタ reg3 のワード･データを下位 32 ビット、reg3+1 のワード･データを上位 32 ビットとするダブルワード・データを、生成したアドレスに格納します。

【補　足】　reg3 は偶数番号のレジスタである必要があります。

注　意

アドレス計算の結果がワード境界の場合は、ミスアライン例外が発生することはありません。

＜ストア命令＞

ST.H

Store half-word

ハーフワード・データのストア

【命令形式】　（1）ST.H　reg2, disp16 [reg1]

（2）ST.H　reg3, disp23 [reg1]

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1
Store-memory (adr, GR[reg2], Half-word)

（2）adr ← GR[reg1] + sign-extend (disp23) 注 1
Store-memory (adr, GR[reg3], Half-word)

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VII

（2）Format XIV

【オペコード】

(1)

15	0	31	16
rrrrr111011RRRRR		ddddddddddddddd0	

ただし、ddddddddddddddd は disp16 の上位 15 ビットです。

(2)

15	0	31	16	47	32
00000111101RRRRR		wwwwwddddd01101		DDDDDDDDDDDDDDDD	

ただし、RRRRR = reg1, wwwww = reg3 です。

dddddd は、disp23 の下位側ビット 6 ～ 1 です。

DDDDDDDDDDDDDDDD は、disp23 の上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　（1）汎用レジスタ reg1 のデータと、ワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。汎用レジスタ reg2 の下位ハーフワード・データを生成したアドレスに格納します。

（2）汎用レジスタ reg1 のデータと、ワード長まで符号拡張した 23 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。汎用レジスタ reg3 の下位ハーフワード・データを生成したアドレスに格納します。

<ストア命令>

ST.W

Store word

ワード・データのストア

【命令形式】　（1）ST.W　reg2, disp16 [reg1]

（2）ST.W　reg3, disp23 [reg1]

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1
Store-memory (adr, GR[reg2], Word)

（2）adr ← GR[reg1] + sign-extend (disp23) 注 1
Store-memory (adr, GR[reg3], Word)

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VII

（2）Format XIV

【オペコード】

(1)

15	0	31	16
rrrrr111011RRRRR		ddddddddddddddd1	

ただし、ddddddddddddddd は disp16 の上位 15 ビットです。

(2)

15	0	31	16	47	32
00000111100RRRRR		wwwwwddddd01111		DDDDDDDDDDDDDDDD	

ただし、RRRRR = reg1, wwwww = reg3 です。

dddddd は、disp23 の下位側ビット 6 ～ 1 です。

DDDDDDDDDDDDDDDD は、disp23 の上位 16 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　（1）汎用レジスタ reg1 のデータと、ワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。汎用レジスタ reg2 のワード・データを生成したアドレスに格納します。

（2）汎用レジスタ reg1 のデータと、ワード長まで符号拡張した 23 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。汎用レジスタ reg3 のワード・データを生成したアドレスに格納します。

＜特殊命令＞

Store Conditional

STC.W

ワード・データ操作がアトミックに完了した場合にストアする条件付きストア

【命令形式】　STC.W　reg3, [reg1]

【オペレーション】

adr ← GR[reg1] 注 1
data ← GR[reg3]
token ← LLbit 注 2

```
if (token == 1)
  then  Store-memory (adr, data, Word)
        GR[reg3] ← 1
  else  GR[reg3] ← 0
endif
```

LLbit ← 0 注 2

注 1. アドレス計算の結果によって、MAE, MDP 例外が発生する場合があります。

注 2. リンクの動作は、**「5.3.2　LDL.W, STC.W 命令による相互排除」**を参照してください。

【フォーマット】　Format VII

【オペコード】

15	0	31	16
00000111111RRRRR		wwwww01101111010	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

指定したアドレスに対応するリンクが存在する場合のみ成功となり、汎用レジスタ reg3 のワード・データをメモリに格納し、アトミック・リード・モディファイ・ライトを完了します。

すでにリンクが消失していた場合は、メモリへのストアは行わず、失敗となります。

STC.W 命令が成功したかどうかは、命令実行後の汎用レジスタ reg3 の内容で示されます。STC.W 命令が成功すると、汎用レジスタ reg3 の内容はセット（1）され、STC.W 命令が失敗するとクリア（0）されます。

LDL.W 命令と STC.W 命令を使い、マルチコア・システムでのメモリ更新を正確に処理できます。

【補　足】　　マルチコア・システムでのアトミック性保証が必要なメモリ更新では、CAXI 命令の代わりに LDL.W 命令と STC.W 命令を使用してください。

＜特殊命令＞

STSR

Store contents of system register

システム・レジスタの内容のストア

【命令形式】　STSR　regID, reg2, selID

STSR　regID, reg2

【オペレーション】　GR[reg2] ← SR[regID, selID] 注 1

注 1. アクセス権限によって例外が発生する場合があります。詳細は**「2.5.3　レジスタの更新」**を参照してください。

【フォーマット】　Format IX

【オペコード】

15	0	31	16
rrrrr111111RRRRR		sssss00001000000	

rrrrr：reg2, sssss：selID, RRRRR：regID

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　システム・レジスタ番号とグループ番号（regID, selID）で指定されるシステム・レジスタの内容を汎用レジスタ reg2 に設定します。システム・レジスタは影響を受けません。selID が省略された場合は selID = 0 が指定されたものとします。

【補　足】　命令実行の結果、CPU 動作モードの状態とアクセス対象のシステム・レジスタの組み合わせによっては、PIE 例外または UCPOP 例外が発生する場合があります。詳細は、**「2.5.3　レジスタの更新」**を参照してください。

注　意

システム・レジスタ番号、グループ番号は、システム・レジスタを一意に識別するための番号です。未定義レジスタに対する動作は**「2.5.4　未定義レジスタへの操作」**で定められていますが、推奨しません。

＜算術演算命令＞

SUB

Subtract

減算

【命令形式】　SUB　reg1, reg2

【オペレーション】　GR[reg2] ← GR[reg2] － GR[reg1]

【フォーマット】　Format I

【オペコード】

15	0
rrrrr001101RRRRR	

【フラグ】

CY	MSB へのボローがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg2 のワード・データから汎用レジスタ reg1 のワード・データを減算し、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

＜算術演算命令＞

SUBR

Subtract reverse

逆減算

【命令形式】　SUBR　reg1, reg2

【オペレーション】　GR[reg2] ← GR[reg1] － GR[reg2]

【フォーマット】　Format I

【オペコード】

15	0
rrrrr001100RRRRR	

【フラグ】

CY	MSB へのボローがあれば 1、そうでないとき 0
OV	オーバフローが起こったとき 1、そうでないとき 0
S	演算結果が負のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg1 のワード・データから汎用レジスタ reg2 のワード・データを減算し、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

＜特殊命令＞

SWITCH

Jump with table look up

テーブル参照分岐

【命令形式】　SWITCH reg1

【オペレーション】　adr ← (PC + 2) + (GR[reg1] logically shift left by 1) 注 1

PC ← (PC + 2) + (sign-extend (Load-memory (adr, Half-word))) logically shift left by 1

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　Format I

【オペコード】

15	0
00000000010RRRRR	

RRRRR ≠ 00000（reg1 には r0 を設定しないでください）

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　次の順に処理を行います。

<1> テーブルの先頭アドレス（SWITCH 命令の次のアドレス）と 1 ビット論理左シフトした汎用レジスタ reg1 のデータを加算し、32 ビット・テーブル・エントリ・アドレスを生成します。

<2> <1> で生成されたアドレスが指し示すハーフワード・エントリ・データをロードします。

<3> ロードしたハーフワード・データをワード長まで符号拡張し、1 ビット論理左シフトしたあとテーブルの先頭アドレス（SWITCH 命令の次のアドレス）を加算し、32 ビット・ターゲット・アドレスを生成します。

<4> <3> で生成されたターゲット・アドレスへ分岐します。

注　意

1. reg1 には、r0 を指定しないでください。
2. SWITCH 命令のテーブル読み出しのためのメモリからの読み出し操作では、メモリ保護が行われます。

＜データ操作命令＞

SXB

Sign extend byte

バイト・データの符号拡張

【命令形式】　　SXB　reg1

【オペレーション】　　GR[reg1] ← sign-extend (GR[reg1] (7:0))

【フォーマット】　　Format I

【オペコード】

15	0
00000000101RRRRR	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　　汎用レジスタ reg1 の最下位バイトをワード長に符号拡張します。

＜データ操作命令＞

SXH

Sign extend half-word

ハーフワード・データの符号拡張

【命令形式】　　SXH　reg1

【オペレーション】　　GR[reg1] ← sign-extend (GR[reg1] (15:0))

【フォーマット】　　Format I

【オペコード】

15	0
00000000111RRRRR	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　　汎用レジスタ reg1 の下位ハーフワードをワード長に符号拡張します。

＜特殊命令＞

SYNCE

Synchronize exceptions

例外同期化命令

【命令形式】　SYNCE

【オペレーション】　例外の同期化を行います。

【フォーマット】　Format I

【オペコード】

15	0
0000000000011101	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　この命令以前のインプレサイス例外（FPI 例外）の同期化を行います。

“インプレサイス例外の同期化”とは、先行する命令に起因して発生するすべてのインプレサイス例外が CPU に通知され、優先順位判定の対象となる時点まで待ち合わせることを指します。したがって、SYNCE 命令の実行時点で例外の 受け付け条件を満たしていた場合、先行する命令に起因して発生するすべてのインプレサイス例外 （FPI 例外）は、SYNCE 命令を実行することで必ず受け付けられます。

この命令はマルチ・プロセッシング環境において、タスク切り替え／終了前の先行タスクの例外処理の完了を保証するために、利用することが可能です。

【補　足】　同期化機能については、**「5.4　同期化機能」**を参照してください。

＜特殊命令＞

SYNCI

Synchronize instruction pipeline

命令パイプライン同期化命令

【命令形式】　SYNCI

【オペレーション】　命令フェッチの同期化を行います。

【フォーマット】　Format I

【オペコード】

15	0
0000000000011100	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　すでにパイプラインに取り込まれている未実行の後続命令を捨て去り、改めて後続命令の命令フェッチを行います。SYNCI 命令は、先行するロード命令およびストア命令の実行結果を待ち合わせません。

【補　足】　同期化機能については、**「5.4　同期化機能」**を参照してください。

＜特殊命令＞

SYNCM

Synchronize memory

メモリ同期化命令

【命令形式】　SYNCM

【オペレーション】　メモリ・アクセスの同期化を行います。

【フォーマット】　Format I

【オペコード】

15	0
0000000000011110	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　先行するすべての命令の実行、および、先行するすべてのメモリ・アクセス（ロードおよびストア）が完了するまで後続命令の実行開始を待ち合わせます。SYNCM 命令を実行することで、システム内のいずれのマスタ・デバイスから参照しても、SYNCM 命令に先行するメモリ・アクセスの結果が参照できる状態になります。

【補　足】　同期化機能については、**「5.4　同期化機能」**を参照してください。SYNCM 命令による先行するストア命令の実行完了の待ち合わせは、ストア命令の書込み先により保証できない場合があります。詳細は、製品のハードウェアマニュアルを参照してください。

＜特殊命令＞

SYNCP

Synchronize pipeline

パイプライン同期化命令

【命令形式】　SYNCP

【オペレーション】　パイプラインの同期化を行います。

【フォーマット】　Format I

【オペコード】

15	0
0000000000011111	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　先行する命令の実行結果を後続命令に反映させるため、後続命令の実行開始を待ち合わせます。SYNCP 命令はロード命令の実行結果（ロード値を汎用レジスタに格納するまで）を待ち合わせますが、ストア命令の実行結果（ストア先のメモリやレジスタの更新）は待ち合わせません。

【補　足】　同期化機能については、**「5.4　同期化機能」**を参照してください。

＜特殊命令＞

System call

SYSCALL

システム・コール例外

【命令形式】　SYSCALL　vector8

【オペレーション】
EIPC ← PC + 4（復帰 PC）
EIPSW ← PSW
EIIC ← 例外要因コード注 1
PSW.UM ← 0
PSW.EP ← 1
PSW.ID ← 1
if (vector8 <= SCCFG.SIZE) is satisfied
　then adr ← SCBP + zero-extend (vector8 logically shift left by 2) 注 2
　else adr ← SCBP 注 2
PC ← SCBP + Load-memory (adr, Word)

注 1.「表 4.1　例外要因一覧」を参照してください。

注 2. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　Format X

【オペコード】

15	0	31	16
11010111111vvvvv		00VVV00101100000	

ただし、VVV は vector8 の上位 3 ビット、vvvvv は vector8 の下位 5 ビットです。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

<1> 復帰 PC（SYSCALL 命令の次の命令のアドレス）と PSW の内容を EIPC と EIPSW に退避します。

<2> vector8 に対応する例外要因コードを、EIIC レジスタに格納します。
例外要因コードは、8000_H に vector8 を加算した値です。

<3> PSW を**「表 4.1　例外要因一覧」**にしたがって更新します。

<4> SCBP レジスタの値と、2 ビット論理左シフトしワード長までゼロ拡張した vector8 を加算して 32 ビット・テーブル・エントリ・アドレスを生成します。
ただし、vector8 がシステム・レジスタの SCCFG.SIZE ビットで指定された値より大きい場合、32 ビット・テーブル・エントリ・アドレスの生成に用いる vector8 は 0 として扱います。

<5> <4> で生成されたアドレスのワードをロードします。

<6> <5> のデータに SCBP レジスタの値を加算した 32 ビット・ターゲット・アドレスを生成します。

<7> <6> で生成されたターゲット・アドレスへ分岐します。

注　意

SYSCALL 命令のテーブル読み出しのためのメモリからの読み出し操作では、スーパバイザ特権でのメモリ保護が行われます。

＜特殊命令＞

TRAP

Trap

ソフトウェア例外

【命令形式】　TRAP　vector5

【オペレーション】

EIPC ← PC + 4 （復帰 PC）
EIPSW ← PSW
EIIC ← 例外要因コード注 1
PSW.UM ← 0
PSW.EP ← 1
PSW.ID ← 1
PC ← 例外ハンドラ・アドレス注 2

注 1.「表 4.1　例外要因一覧」を参照してください。
注 2.「4.5　例外ハンドラ・アドレス」を参照してください。

【フォーマット】　Format X

【オペコード】

15	031	16
00000111111vvvvv	0000000100000000	

ただし、vvvvv は vector5 です。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】

復帰 PC（TRAP 命令の次の命令のアドレス）と現在の PSW の内容を、それぞれ EIPC と EIPSW に退避し、例外要因コードを EIIC レジスタに格納、PSW を**「表 4.1　例外要因一覧」**にしたがって更新します。続いて、例外ハンドラ・アドレスに分岐し、例外処理を開始します。

vector5 と例外要因コード、例外ハンドラ・アドレスのオフセットの対応は次の表で示されます。例外ハンドラ・アドレスは、次の表が示すオフセット・アドレスを元に計算されます。詳細は、**「4.5　例外ハンドラ・アドレス」**を参照してください。

vector5	例外要因コード	オフセット・アドレス
00_H	0000 0040_H	40_H
01_H	0000 0041_H	
（中略）		
$0F_H$	0000 $004F_H$	
10_H	0000 0050_H	50_H
11_H	0000 0051_H	
（中略）		
$1F_H$	0000 $005F_H$	

＜論理演算命令＞

TST

Test

テスト

【命令形式】　　TST　reg1, reg2

【オペレーション】　　result ← GR[reg2] AND GR[reg1]

【フォーマット】　　Format I

【オペコード】

15	0
rrrrr001011RRRRR	

【フラグ】

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　　汎用レジスタ reg2 のワード・データと汎用レジスタ reg1 のワード・データの論理積をとります。結果は格納されず、フラグだけが影響を受けます。汎用レジスタ reg1, reg2 は影響を受けません。

＜ビット操作命令＞

TST1

Test bit

ビット・テスト

【命令形式】　（1）TST1　bit#3, disp16 [reg1]

（2）TST1　reg2, [reg1]

【オペレーション】　（1）adr ← GR[reg1] + sign-extend (disp16) 注 1
token ← Load-memory (adr, Byte)
Z フラグ ← Not (extract-bit (token, bit#3))

（2）adr ← GR[reg1] 注 1
token ← Load-memory (adr, Byte)
Z フラグ ← Not (extract-bit (token, reg2))

注 1. アドレス計算の結果によって、MDP 例外が発生する場合があります。

【フォーマット】　（1）Format VIII

（2）Format IX

【オペコード】

(1)

15	0	31	16
11bbb111110RRRRR		ddddddddddddddddd	

(2)

15	0	31	16
rrrrr111111RRRRR		0000000011100110	

【フラグ】

CY	—
OV	—
S	—
Z	指定したビットが 0 のとき 1、指定したビットが 1 のとき 0
SAT	—

【説　明】

(1) まず、汎用レジスタ reg1 のワード・データと、ワード長まで符号拡張した 16 ビット・ディスプレースメントを加算して 32 ビット・アドレスを生成します。
生成したアドレスのバイト・データの、3 ビットのビット・ナンバで指定されるビットが 0 ならば Z フラグをセット（1）し、1 ならばクリア（0）します。
指定されたビットも含め、バイト・データは影響を受けません。

(2) まず、汎用レジスタ reg1 のワード・データを読み出して 32 ビット・アドレスを生成します。
生成したアドレスのバイト・データの、汎用レジスタ reg2 の下位 3 ビットで指定されるビットが 0 ならば Z フラグをセット（1）し、1 ならばクリア（0）します。指定されたビットも含め、バイト・データは影響を受けません。
読み出したバイト・データの指定ビットが 0 のとき Z フラグをセット（1）し、指定ビットが 1 のとき Z フラグをクリア（0）します。

＜論理演算命令＞

XOR

Exclusive OR

排他的論理和

【命令形式】　　XOR　reg1, reg2

【オペレーション】　　GR[reg2] ← GR[reg2] XOR GR[reg1]

【フォーマット】　　Format I

【オペコード】

15	0
rrrrr001001RRRRR	

【フラグ】

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　　汎用レジスタ reg2 のワード・データと汎用レジスタ reg1 のワード・データとの排他的論理和をとり、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

＜論理演算命令＞

XORI

Exclusive OR immediate (16-bit)

排他的論理和

【命令形式】　XORI imm16, reg1, reg2

【オペレーション】　GR[reg2] ← GR[reg1] XOR zero-extend (imm16)

【フォーマット】　Format VI

【オペコード】

15	0	31	16
rrrrr110101RRRRR		iiiiiiiiiiiiiiii	

【フラグ】

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 のとき 1、そうでないとき 0
Z	演算結果が 0 のとき 1、そうでないとき 0
SAT	—

【説　明】　汎用レジスタ reg1 のワード・データとワード長までゼロ拡張した 16 ビット・イミディエイトの排他的論理和をとり、その結果を汎用レジスタ reg2 に格納します。汎用レジスタ reg1 は影響を受けません。

＜データ操作命令＞

ZXB

Zero extend byte

バイト・データのゼロ拡張

【命令形式】　　ZXB　reg1

【オペレーション】　　GR[reg1] ← zero-extend (GR[reg1] (7:0))

【フォーマット】　　Format I

【オペコード】

15	0
00000000100RRRRR	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　　汎用レジスタ reg1 の最下位バイトをワード長にゼロ拡張します。

＜データ操作命令＞

ZXH

Zero extend half-word

ハーフワード・データのゼロ拡張

【命令形式】　　ZXH　reg1

【オペレーション】　　GR[reg1] ← zero-extend (GR[reg1] (15:0))

【フォーマット】　　Format I

【オペコード】

15	0
00000000110RRRRR	

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　　汎用レジスタ reg1 の下位ハーフワードをワード長にゼロ拡張します。

7.3　キャッシュ命令

7.3.1　キャッシュ命令の概要

本 CPU では、キャッシュ命令は非搭載です。

キャッシュ機能命令には、次の命令（ニーモニック）があります。

- CACHE 注 1: Cache
- PREF 注 1　: Prefetch

注 1. CACHE, PREF 命令は NOP 命令として処理されます。

7.3.2　キャッシュ命令セット

この節では、各命令のニーモニックごとに（アルファベット順）、次の項目に分けて説明します。

- 命令形式　　：命令の記述方法、オペランドを示します。
- オペレーション：命令の機能を示します。
- フォーマット　：命令形式を命令フォーマットで示します。
- オペコード　　：命令のオペコードをビット・フィールドで示します。
- 説明　　：命令の動作説明をします。
- 補足　　：命令の補足説明をします。

<キャッシュ命令>

CACHE

Cache

キャッシュ操作

【命令形式】　　CACHE　cacheop, [reg1]

【オペレーション】　　何も行いません。

【フォーマット】　　Format X

【オペコード】

15	0	31	16
111pp111111RRRRR		PPPPP00101100000	

ppPPPPP が cacheop を示します。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　　本 CPU では CACHE 命令は NOP 命令として処理されます。

<キャッシュ命令>

PREF

Prefetch

プリフェッチ

【命令形式】　　PREF　prefop, [reg1]

【オペレーション】　　何も行いません。

【フォーマット】　　Format X

【オペコード】

15	0 31	16
11011111111RRRRR	PPPPP00101100000	

PPPPP が prefop を示します。

【フラグ】

CY	—
OV	—
S	—
Z	—
SAT	—

【説　明】　　本 CPU では PREF 命令は NOP 命令として処理されます。

7.4　浮動小数点演算命令

7.4.1　命令フォーマット

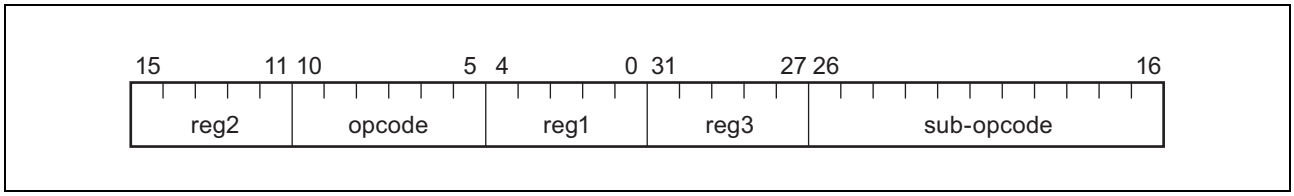
浮動小数点演算は、すべて 32 ビット・フォーマットです。

実際に命令がメモリに格納されるときは、次のように配置されます。

- 各命令形式の下位部分（ビット 0 を含む）→下位アドレス側
- 各命令形式の上位部分（ビット 15 またはビット 31 を含む）→上位アドレス側

(1) Format F:I

6 ビットのオペコード・フィールド、4 ビットのサブ・オペコード・フィールド、3 つの汎用レジスタ指定フィールド、3 ビットのカテゴリ・フィールド、2 ビットのタイプ・フィールドを持つ 32 ビット長浮動小数点演算命令形式。



7.4.2　浮動小数点演算命令の概要

浮動小数点演算命令は単精度浮動小数点演算（Single）をサポートしており、次の命令（ニーモニック）があります。

(1) 基本演算命令

- ABSF.S　：Floating-point Absolute Value（Single）
- ADDF.S　：Floating-point Add（Single）
- DIVF.S　：Floating-point Divide（Single）
- MAXF.S　：Floating-point Maximum（Single）
- MINF.S　：Floating-point Minimum（Single）
- MULF.S　：Floating-point Multiply（Single）
- NEGF.S　：Floating-point Negate（Single）
- RECIPF.S　：Reciprocal of a floating-point value（Single）
- RSQRTF.S　：Reciprocal of the square root of a floating-point value（Single）
- SQRTF.S　：Floating-point Square Root（Single）
- SUBF.S　：Floating-point Subtract（Single）

(2) 拡張基本演算命令

- FMAF.S　：Floating-point Fused-Multiply-Add（Single）
- FMSF.S　：Floating-point Fused-Multiply-Subtract（Single）
- FNMAF.S　：Floating-point Fused-Negate-Multiply-Add（Single）
- FNMSF.S　：Floating-point Fused-Negate-Multiply-Subtract（Single）

(3) 変換命令

- CEILF.SL　：Floating-point Convert Single to Long, round toward positive（Single）
- CEILF.SW　：Floating-point Convert Single to Word, round toward positive（Single）
- CEILF.SUL　：Floating-point Convert Single to Unsigned-Long, round toward positive（Single）
- CEILF.SUW　：Floating-point Convert Single to Unsigned-Word, round toward positive（Single）
- CVTF.LS　：Floating-point Convert Long to Single（Single）
- CVTF.SL　：Floating-point Convert Single to Long（Single）
- CVTF.SUL　：Floating-point Convert Single to Unsigned-Long（Single）
- CVTF.SUW　：Floating-point Convert Single to Unsigned-Word（Single）
- CVTF.SW　：Floating-point Convert Single to Word（Single）
- CVTF.ULS　：Floating-point Convert Unsigned-Long to Single（Single）
- CVTF.UWS　：Floating-point Convert Unsigned-Word to Single（Single）

- CVTF.WS　：Floating-point Convert Word to Single（Single）
- FLOORF.SL　：Floating-point Convert Single to Long, round toward negative（Single）
- FLOORF.SW　：Floating-point Convert Single to Word, round toward negative（Single）
- FLOORF.SUL　：Floating-point Convert Single to Unsigned-Long, round toward negative（Single）
- FLOORF.SUW　：Floating-point Convert Single to Unsigned-Word, round toward negative（Single）
- ROUNDF.SL　：Floating-point Convert Single to Long, round to nearest（Single）
- ROUNDF.SW　：Floating-point Convert Single to Word, round to nearest（Single）
- ROUNDF.SUL　：Floating-point Convert Single to Unsigned-Long, round to nearest（Single）
- ROUNDF.SUW　：Floating-point Convert Single to Unsigned-Word, round to nearest（Single）
- TRNCF.SL　：Floating-point Convert Single to Long, round toward zero（Single）
- TRNCF.SUL　：Floating-point Convert Single to Unsigned-Long, round toward zero（Single）
- TRNCF.SUW　：Floating-point Convert Single to Unsigned-Word, round toward zero（Single）
- TRNCF.SW　：Floating-point Convert Single to Word, round toward zero（Single）
- CVTF.HS　：Floating-point Convert Half to Single（Single）
- CVTF.SH　：Floating-point Convert Single to Half（Single）

(4) 比較命令

- CMPF.S　：Compare floating-point values（Single）

(5) 条件付き転送命令

- CMOVF.S　：Floating-point conditional move（Single）

(6) 条件ビット転送命令

- TRFSR　：Transfers specified CC bit to Zero flag in PSW（Single）

7.4.3　比較命令のための条件

浮動小数点の比較命令（CMPF.S）は、2 つの浮動小数点データの比較演算を行います。結果は、データとコード中に含まれる比較条件に基づいて決定します。**表 7.7** に比較命令で指定可能な条件のニーモニックを示します。

比較命令の結果を TRFSR 命令により PSW（プログラム・ステータス・ワード）の Z フラグに転送して、条件分岐を行う場合、条件の論理を反転して使用できます。**表 7.8** は条件の真偽による論理反転を示しています。浮動小数点比較命令の 4 ビットの条件コードでは、表中の「真」欄の条件を指定します。条件分岐命令 BT は比較の結果が真である場合に分岐し、BF は偽である場合に分岐します。

表 7.7　比較命令のための条件一覧

ニーモニック	定 義	論理反転	
F	常に偽	（T）	常に真
UN	Unordered	（OR）	Ordered
EQ	等しい	（NEQ）	等しくない
UEQ	Unordered か等しい	（OLG）	Ordered で、より小さいか、より大きい
OLT	Ordered で、より小さい	（UGE）	Unordered か、より大きいか、等しい
ULT	Unordered か、より小さい	（OGE）	Ordered で、より大きいか、等しい
OLE	Ordered で、より小さいか、等しい	（UGT）	Unordered か、より大きい
ULE	Unordered か、より小さいか、等しい	（OGT）	Ordered で、より大きい
SF	Signaling で偽	（ST）	Signaling で真
NGLE	より大きくない、かつより小さくない、かつ等しくない	（GLE）	より大きいか、より小さいか、等しい
SEQ	Signaling で等しい	（SNE）	Signaling で等しくない
NGL	より大きくない、かつより小さくない	（GL）	より大きいか、より小さい
LT	より小さい	（NLT）	より小さくない
NGE	より大きくない、かつ等しくない	（GE）	より大きいか、等しい
LE	より小さいか、等しい	（NLE）	より小さくない、かつ等しくない
NGT	より大きくない	（GT）	より大きい

表 7.8　条件コードのビット定義と論理反転

ニーモニック（真）	条件コード fcond		条件コード fcond(3:0) のビット定義				論理反転（偽）
			より小さい	等しい	Unordered	Unordered のとき無効演算例外の検出	
	10 進	2 進	fcond(2)	fcond(1)	fcond(0)	fcond(3)	
F	0	0b0000	F	F	F	しない	（T）
UN	1	0b0001	F	F	T	しない	（OR）
EQ	2	0b0010	F	T	F	しない	（NEQ）
UEQ	3	0b0011	F	T	T	しない	（OLG）
OLT	4	0b0100	T	F	F	しない	（UGE）
ULT	5	0b0101	T	F	T	しない	（OGE）
OLE	6	0b0110	T	T	F	しない	（UGT）
ULE	7	0b0111	T	T	T	しない	（OGT）
SF	8	0b1000	F	F	F	する	（ST）
NGLE	9	0b1001	F	F	T	する	（GLE）
SEQ	10	0b1010	F	T	F	する	（SNE）
NGL	11	0b1011	F	T	T	する	（GL）
LT	12	0b1100	T	F	F	する	（NLT）
NGE	13	0b1101	T	F	T	する	（GE）
LE	14	0b1110	T	T	F	する	（NLE）
NGT	15	0b1111	T	T	T	する	（GT）

7.4.4　浮動小数点演算命令セット

この節では、各命令のニーモニックごとに（アルファベット順）、次の項目に分けて説明します。

- 命令形式　　：命令の記述方法、オペランドを示します（略号については、**表 7.9** 参照）。
- オペレーション：命令の機能を示します（略号については、**表 7.10** 参照）。
- フォーマット　：命令形式を命令フォーマットで示します（**「7.4.1　命令フォーマット」**参照）。
- オペコード　　：命令のオペコードをビット・フィールドで示します（略号については、**表 7.11** 参照）。
- 説明　　：命令の動作説明をします。
- 補足　　：命令の補足説明をします。

表 7.9　命令形式の凡例

略　号	意　味
reg1	汎用レジスタ
reg2	汎用レジスタ
reg3	汎用レジスタ
reg4	汎用レジスタ
fcbit	浮動小数点比較命令の結果を格納するコンディション・ビットのビット・ナンバを指定
imm ×	× ビット・イミディエイト・データ
fcond	比較命令の比較条件のニーモニックまたは条件コードを指定（詳細は**「7.4.3　比較命令のための条件」**を参照してください）

表 7.10　オペレーションの凡例

略　号	意　味
←	代入
GR[a]	汎用レジスタ「a」の格納値
SR[a, b]	システム・レジスタ（RegID =「a」, SelID =「b」）の格納値
result	結果をフラグに反映する。
==	比較（一致で真）
+	加算
–	減算
II	ビット連結
×	乗算
÷	除算
abs	絶対値
ceil	+ ∞方向への丸め
compare	比較
cvt	丸めモードにしたがう型変換
floor	– ∞方向への丸め
max	最大値
min	最小値
neg	符号反転
round	最も近い値への丸め
sqrt	平方根
trunc	ゼロ方向への丸め
fma(a, b, c)	a と b の積に c を加算する結合積和算
fms(a, b, c)	a と b の積から c を減算する結合積差算

表 7.11　オペコードの凡例

略　号	意　味
R	reg1 を指定するコードの 1 ビット分データ
r	reg2 を指定するコードの 1 ビット分データ
w	reg3 を指定するコードの 1 ビット分データ
W	reg4 を指定するコードの 1 ビット分データ
I	イミディエイトの 1 ビット分データ（イミディエイトの上位ビットを示す）
i	イミディエイトの 1 ビット分データ
fff	浮動小数点比較命令の結果を格納するコンディション・ビットのビット・ナンバ（fcbit）を指定する 3 ビット・データ
FFFF	比較命令の比較条件のニーモニックまたは条件コード（fcond）に対応する 4 ビット・データ

＜浮動小数点演算命令＞

ABSF.S

Floating-point Absolute Value (Single)

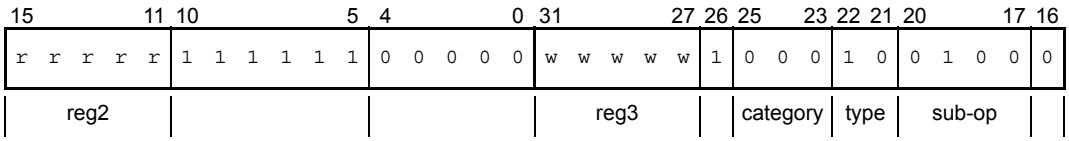
浮動小数点絶対値（単精度）

【命令形式】　　ABSF.S　reg2, reg3

【オペレーション】　　reg3 ← abs(reg2)

【フォーマット】　　Format F: I

【オペコード】



【説　明】　　汎用レジスタ reg2 にある単精度浮動小数点形式の内容の絶対値をとり、汎用レジスタ reg3 に格納します。

【浮動小数点演算例外】　　なし

【補　足】　　FPSR.FS = 1 でもサブノーマル数の入力はフラッシュされません。

＜浮動小数点演算命令＞

ADDF.S

Floating-point Add (Single)

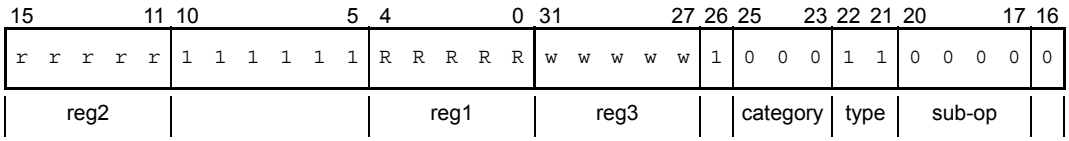
浮動小数点加算（単精度）

【命令形式】　　ADDF.S　reg1, reg2, reg3

【オペレーション】　　reg3 ← reg2 + reg1

【フォーマット】　　Format F: I

【オペコード】



【説　明】　　汎用レジスタ reg2 にある単精度浮動小数点形式の内容と汎用レジスタ reg1 にある単精度浮動小数点形式の内容を加算し、汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し、結果を現在の丸めモードにしたがって丸めます。

【浮動小数点演算例外】　　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
オーバフロー例外（O）
アンダフロー例外（U）

【演算結果】

reg1(A) \ reg2(B)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
+Normal	A + B					– ∞	Q-NaN	Q-NaN[V]
–Normal								
+0								
–0								
+ ∞					+ ∞	Q-NaN[V]		
– ∞	– ∞				Q-NaN[V]	– ∞		
Q-NaN	Q-NaN							
S-NaN	Q-NaN[V]							

備考 1.　［　］は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

CEILF.SL

Floating-point Convert Single to Long, round toward positive (Single)

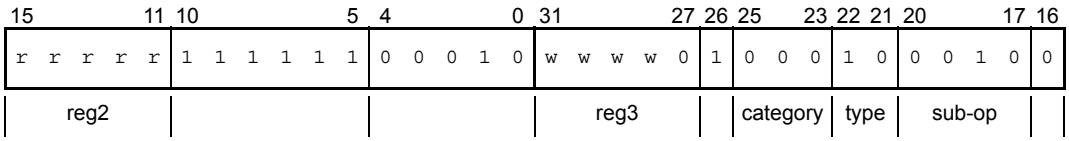
整数形式への変換（単精度）

【命令形式】　CEILF.SL　reg2, reg3

【オペレーション】　reg3 ← ceil reg2 (single → long-word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に 64 ビットの整数形式に変換し、汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。

現在の丸めモードに関係なく、結果を＋∞の方向へ丸めます。

ソース・オペランドが無限大か非数の場合、または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞　　：$2^{63}-1$ を返します。
- ソースが負数、非数または－∞　　：-2^{63} を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A（Integer）		0（Integer）		+Max Int [V]	–Max Int [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Unsigned-Long, round toward positive (Single)

CEILF.SUL

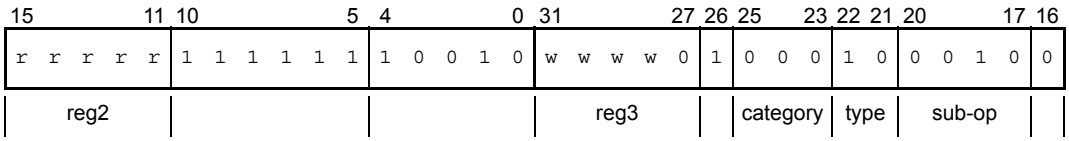
符号なし整数形式への変換（単精度）

【命令形式】　CEILF.SUL　reg2, reg3

【オペレーション】　reg3 ← ceil reg2 (Single → Unsigned long-word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に符号のない 64 ビットの整数形式に変換し、汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。

現在の丸めモードに関係なく、結果を＋∞の方向へ丸めます。

ソース・オペランドが無限大か非数か負数の場合、または丸めた結果が $2^{64}-1 \sim 0$ の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの値により返す値は次のように異なります。

- ソースが $2^{64}-1 \sim 0$ の範囲外の正数または＋∞　：$2^{64}-1$ を返します。
- ソースが負数、非数または－∞　：0 を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A （Integer）	0 [V]	0（Integer）		Max U-Int [V]	0 [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Unsigned-Word, round toward positive (Single)

CEILF.SUW

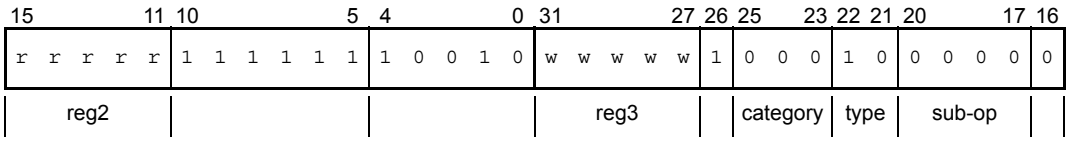
符号なし整数形式への変換（単精度）

【命令形式】　CEILF.SUW　reg2, reg3

【オペレーション】　reg3 ← ceil reg2(Single → Unsigned word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に符号のない 32 ビットの整数形式に変換し、汎用レジスタ reg3 に格納します。

現在の丸めモードに関係なく、結果を + ∞の方向へ丸めます。

ソース・オペランドが無限大か非数か負数の場合、または丸めた結果が $2^{32}-1$ ～ 0 の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの値により返す値は次のように異なります。

- ソースが $2^{32}-1$ ～ 0 の範囲外の正数または + ∞ ：$2^{32}-1$ を返します。
- ソースが負数、非数または – ∞　　：0 を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A （Integer）	0 [V]	0（Integer）		Max U-Int [V]	0 [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

CEILF.SW

Floating-point Convert Single to Word, round toward positive (Single)

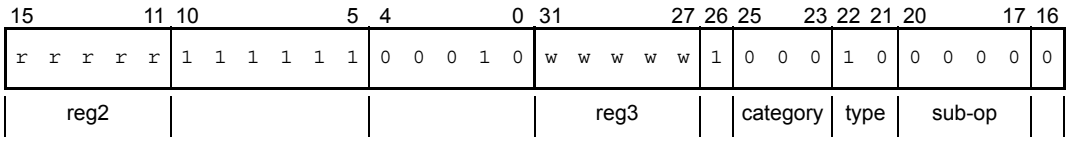
整数形式への変換（単精度）

【命令形式】　CEILF.SW　reg2, reg3

【オペレーション】　reg3 ← ceil reg2 (single → word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し、汎用レジスタ reg3 に格納します。

現在の丸めモードに関係なく、結果を＋∞の方向へ丸めます。

ソース・オペランドが無限大か非数の場合、または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞　　：$2^{31}-1$ を返します。
- ソースが負数、非数または－∞　　：-2^{31} を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A（Integer）		0（Integer）		+Max Int[V]	-Max Int[V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点条命令＞

CMOVF.S

Floating-point Conditional Move (Single)

条件付き転送（単精度）

【命令形式】　CMOVF.S　fcbit, reg1, reg2, reg3

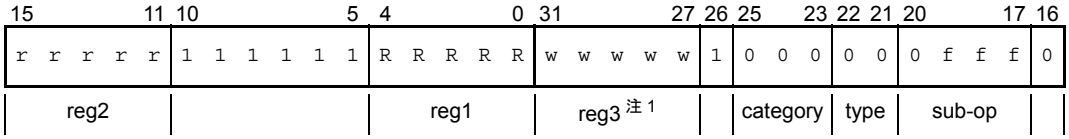
【オペレーション】

```
if (FPSR.CCn == 1) then
    reg3 ← reg1
else
    reg3 ← reg2
endif
```

備考　n = fcbit

【フォーマット】　Format F: I

【オペコード】



注 1.　reg3：wwwww！= 0
wwwww ≠ 00000（reg3 には r0 を設定しないでください）

備考　fcbit：fff

【説　明】　オペコードの fcbit によって指定された FPSR.CC(7:0) ビットが真（1）の場合、reg1 のデータを reg3 に格納します。偽（0）の場合、reg2 のデータを reg3 に格納します。

【浮動小数点演算例外】　なし

【補　足】　FPSR.FS = 1 でもサブノーマル数の入力はフラッシュされません。

注　意

reg3 には r0 を指定しないでください。

＜浮動小数点演算命令＞

CMPF.S

Compare floating-point values (Single)

浮動小数点比較（単精度）

【命令形式】　　CMPF.S　fcond, reg2, reg1, fcbit

CMPF.S　fcond, reg2, reg1

【オペレーション】　　if　isNaN(reg1)　or　isNaN(reg2)　then

result.less　← 0

result.equal　← 0

result.unordered　← 1

if　fcond[3] == 1 then

無効演算例外を検出

endif

else

result.less　← reg2 <　reg1

result.equal ← reg2 == reg1

result.unordered ← 0

endif

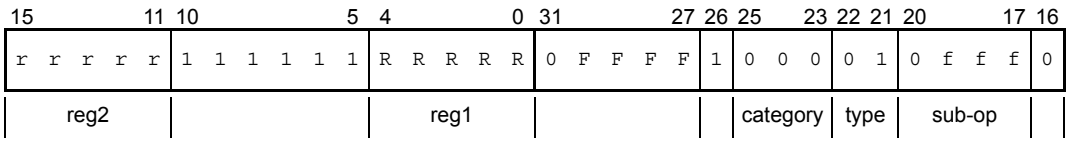
FPSR.CCn　← (fcond[2] & result.less) | (fcond[1] & result.equal) |

(fcond[0] & result.unordered)

備考　n：fcbit

【フォーマット】　　Format F: I

【オペコード】



備考　fcond：FFFF
fcbit：fff

【説　明】

汎用レジスタ reg2 で指定される単精度浮動小数点形式の内容を、比較条件 fcond により、汎用レジスタ reg1 で指定される単精度浮動小数点形式の内容と比較します。結果（真ならば 1、偽ならば 0）をオペコードの fcbit で指定される FPSR レジスタのコンディション・ビット（CC(7:0) ビット：ビット 31 ～ 24）にセットします。fcbit が省略された場合は CC0 ビット（ビット 24）にセットします。

比較条件 fcond のコードについては**「表 7.12　比較条件」**を参照してください。

値の 1 つが非数で、比較条件 fcond の最上位ビットがセットされている場合は、IEEE754 の無効演算例外を検出します。無効演算例外の発生が許可されている場合は、比較結果はセットされず、そのまま例外の処理に移ります。

許可ビットがセットされていない場合は、例外を発生せず、FPSR レジスタの保存ビット（ビット 4）がセットされ、FPSR.CC(7:0) ビットに比較結果がセットされます。

比較も含め、浮動小数点演算命令ではオペランド値として SignalingNaN（S-NaN）を受け取ると、無効演算の条件と見なします。S-NaN だけでなく QuietNaN（Q-NaN）でも無効演算となる比較を使えば、NaN でエラーとなる場合のプログラムをより簡単なものにできます。つまり、結果が Unordered となるような Q-NaN を明確にチェックするためのコードが不要になります。代わりに、無効演算が検出されたときに例外を発生させ、エラーの処理を例外処理システムが行うようにします。次に、2 つの数値が等しいかを調べるとともに、Unordered の場合はエラーとするような比較の場合を示します。

表 7.12　比較条件

比較条件		定　義	説　　明	Unordered による無効演算例外の検出
	fcond			
F	0	FALSE	常に偽	しない
UN	1	Unordered	reg1, reg2 の少なくとも一方が非数	しない
EQ	2	reg2 ＝ reg1	Ordered（いずれも非数ではない）で、等しい	しない
UEQ	3	reg2 ? ＝ reg1	Unordered（少なくとも一方が非数）か、等しい	しない
OLT	4	reg2 ＜ reg1	Ordered（いずれも非数ではない）で、より小さい	しない
ULT	5	reg2 ? ＜ reg1	Unordered（少なくとも一方が非数）か、より小さい	しない
OLE	6	reg2 ≦ reg1	Ordered（いずれも非数ではない）で、より小さいか、等しい	しない
ULE	7	reg2 ? ≦ reg1	Unordered（少なくとも一方が非数）か、より小さいか、等しい	しない
SF	8	FALSE	常に偽	する
NGLE	9	Unordered	reg1, reg2 の少なくとも一方が非数	する
SEQ	10	reg2 ＝ reg1	Ordered（いずれも非数ではない）で、等しい	する
NGL	11	reg2 ? ＝ reg1	Unordered（少なくとも一方が非数）か、等しい	する
LT	12	reg2 ＜ reg1	Ordered（いずれも非数ではない）で、より小さい	する
NGE	13	reg2 ? ＜ reg1	Unordered（少なくとも一方が非数）か、より小さい	する
LE	14	reg2 ≦ reg1	Ordered（いずれも非数ではない）で、より小さいか、等しい	する
NGT	15	reg2 ? ≦ reg1	Unordered（少なくとも一方が非数）か、より小さいか、等しい	する

備考　?：Unordered（比較不能）

```
# Q-NaN を明確にテストする場合
        CMPF.S    OLT,r12,r14,0  # r12 < r14 をチェック
        CMPF.S    UN,r12,r14,1   # Unordered かをチェック
        TRFSR     0
        BT        L2             # 真の場合は L2 へ
        TRFSR     1
        BT        ERROR          # 真の場合はエラー処理へ
# Unordered でなく、かつ、r12 < r14 でない場合の処理コードを記述
L2:
# r12 < r14 の場合の処理コードを記述
        :
# Q-NaN を通知する比較を使用する場合
        CMPF.S    LT,r12,r14,0   # r12 ?< r14 をチェック
        TRFSR     0
        BT        L2             # 真の場合は L2 へ
# Unordered でなく、かつ、r12 < r14 でない場合の処理コードを記述
L2:
# Unordered か、r12 < r14 の場合の処理コードを記述
        :
```

【浮動小数点演算例外】　無効演算例外（V）

【補　足】　FPSR.FS = 1 でもサブノーマル数の入力はフラッシュされません。

【演算結果】　［条件コード（fcond）= 0-7］

reg1(B) / reg2(A)	+Normal	–Normal	+0	-0	+ ∞	– ∞	Q-NaN	S-NaN
±Normal	比較条件（fcond）による比較結果の真偽を FPSR.CCn ビットに格納（n = fcbit）							
±0								
± ∞								
Q-NaN	Unorderd							
S-NaN	Unorderd[V]							

［条件コード（fcond）= 8-15］

reg1(B) / reg2(A)	+Normal	–Normal	+0	-0	+ ∞	– ∞	Q-NaN	S-NaN
±Normal	比較条件（fcond）による比較結果の真偽を FPSR.CCn ビットに格納（n = fcbit）							
±0								
± ∞								
Q-NaN	Unorderd[V]							
S-NaN								

備考　［　］は必ず発生する例外です。

＜浮動小数点演算命令＞

CVTF.HS

Floating-point Convert Half to Single (Single)

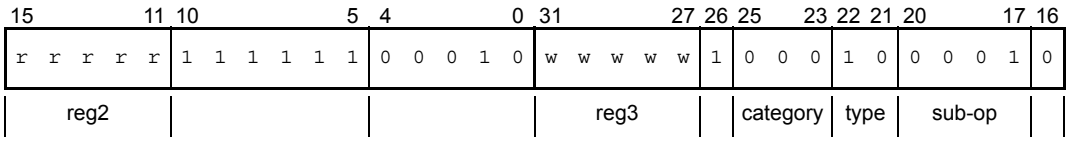
浮動小数点形式への変換（単精度）

【命令形式】　CVTF.HS　reg2, reg3

【オペレーション】　reg3 ← cvt reg2 (half → single)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 の下位 16 ビットにある半精度浮動小数点形式の内容を現在の丸めモードにしたがって算術的に単精度浮動小数点形式に変換し、汎用レジスタ reg3 に格納します。

【浮動小数点演算例外】　無効演算例外（V）

【補　足】　非数を除く、すべての半精度浮動小数点形式を正確に単精度浮動小数点形式に変換することができます。

FPSR.FS = 1 でもサブノーマル数の入力はフラッシュされません。

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A（Half）		+0	–0	+ ∞	– ∞	Q-NaN	Q-NaN[V]

備考 1.　[　] は必ず発生する例外です。

＜浮動小数点演算命令＞

CVTF.LS

Floating-point Convert Long to Single (Single)

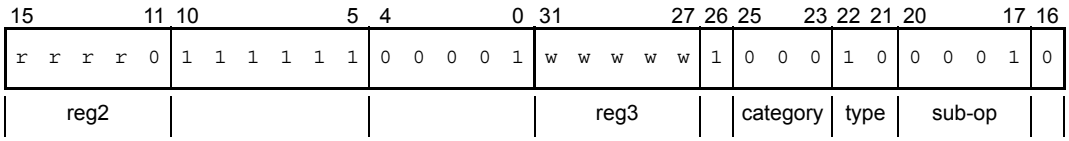
浮動小数点形式への変換（単精度）

【命令形式】　CVTF.LS　reg2, reg3

【オペレーション】　reg3 ← cvt reg2(long-word → single)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 で指定されるレジスタ・ペアにある 64 ビットの整数形式の内容を算術的に単精度浮動小数点形式に変換し、汎用レジスタ reg3 に格納します。結果は、現在の丸めモードにしたがって丸めます。

【浮動小数点演算例外】　不正確演算例外（I）

【演算結果】

reg2(A)	+Integer	–Integer	0（Integer）
演算結果 [例外]	A（Normal）		+0

＜浮動小数点演算命令＞

Floating-point Convert Single to Long (Single)

CVTF.SL

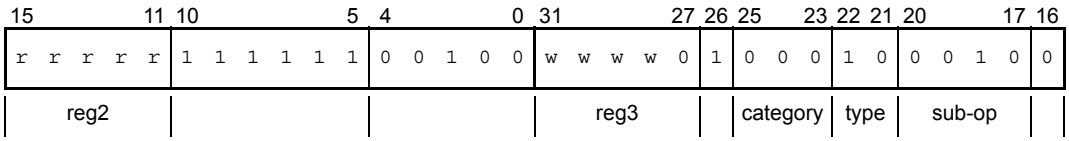
整数形式への変換（単精度）

【命令形式】　CVTF.SL　reg2, reg3

【オペレーション】　reg3 ← cvt reg2(single → long-word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を現在の丸めモードにしたがって算術的に 64 ビットの整数形式に変換し、汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。

ソース・オペランドが無限大か非数の場合、または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または + ∞　　：$2^{63}-1$ を返します。
- ソースが負数、非数または – ∞　　：-2^{63} を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A（Integer）		0（Integer）		+Max Int[V]	–Max Int[V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

CVTF.SH

Floating-point Convert Single to Half (Single)

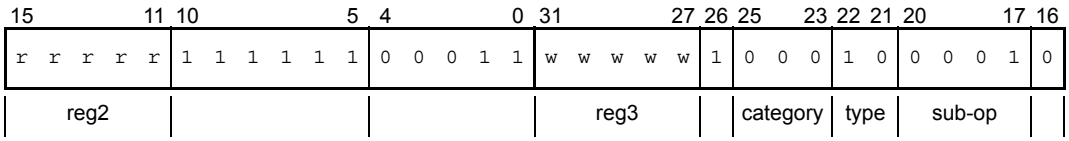
半精度浮動小数点形式への変換（単精度）

【命令形式】　CVTF.SH　reg2, reg3

【オペレーション】　reg3 ← zero-extend (cvt reg2 (single → half))

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を現在の丸めモードにしたがって算術的に半精度浮動小数点形式に変換します。結果はワード長までゼロ拡張し、汎用レジスタ reg3 に格納します。

【浮動小数点演算例外】　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
オーバフロー例外（O）
アンダフロー例外（U）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A（Half）		+0	-0	+ ∞	– ∞	Q-NaN	Q-NaN[V]

備考 1.　[　] は必ず発生する例外です。
備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Unsigned-Long (Single)

CVTF.SUL

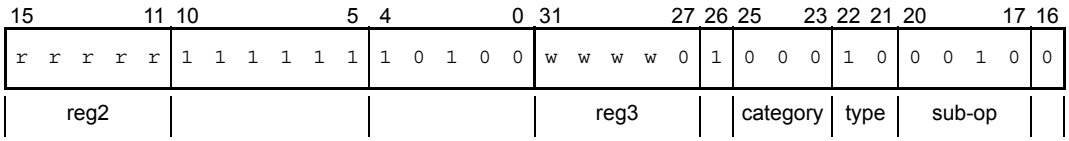
符号なし整数形式への変換（単精度）

【命令形式】　CVTF.SUL　reg2, reg3

【オペレーション】　reg3 ← cvt reg2(Single → unsigned long-word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を現在の丸めモードにしたがって算術的に符号のない 64 ビットの整数形式に変換し、汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。

ソース・オペランドが無限大か非数か負数の場合、または丸めた結果が $2^{64}-1 \sim 0$ の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの値により返す値は次のように異なります。

- ソースが $2^{64}-1 \sim 0$ の範囲外の正数または $+\infty$ ：$2^{64}-1$ を返します。
- ソースが負数、非数または $-\infty$ ：0 を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A （Integer）	0 [V]	0（Integer）		Max U-Int [V]	0 [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Unsigned-Word (Single)

CVTF.SUW

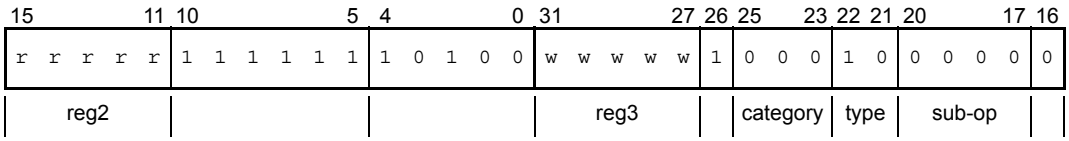
符号なし整数形式への変換（単精度）

【命令形式】　CVTF.SUW　reg2, reg3

【オペレーション】　reg3 ← cvt reg2 (Single → unsigned word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に符号のない 32 ビットの整数形式に変換し、汎用レジスタ reg3 に格納します。

ソース・オペランドが無限大か非数か負数の場合、または丸めた結果が $2^{32}-1 \sim 0$ の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの値により返す値は次のように異なります。

- ソースが $2^{32}-1 \sim 0$ の範囲外の正数または $+\infty$ ：$2^{32}-1$ を返します。
- ソースが負数、非数または $-\infty$ ：0 を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A （Integer）	0 [V]	0（Integer）		Max U-Int [V]	0 [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Word (Single)

CVTF.SW

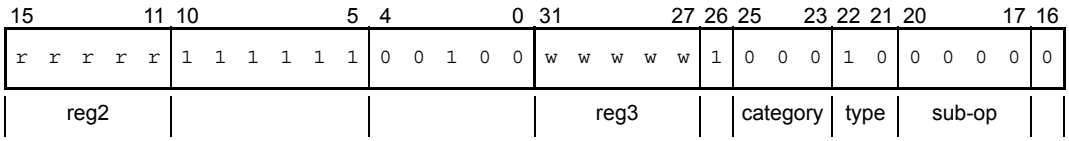
整数形式への変換（単精度）

【命令形式】　　CVTF.SW　reg2, reg3

【オペレーション】　　reg3 ← cvt reg2 (single → word)

【フォーマット】　　Format F: I

【オペコード】



【説　明】　　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し、汎用レジスタ reg3 に格納します。

ソース・オペランドが無限大か非数の場合、または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または + ∞　　: $2^{31}-1$ を返します。
- ソースが負数、非数または – ∞　　: -2^{31} を返します。

【浮動小数点演算例外】　　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A (Integer)		0 (Integer)		+Max Int[V]	–Max Int[V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Unsigned-Long to Single (Single)

CVTF.ULS

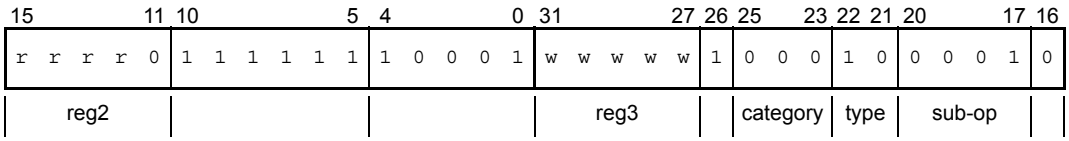
浮動小数点形式への変換（単精度）

【命令形式】　CVTF.ULS　reg2, reg3

【オペレーション】　reg3 ← cvt reg2 (unsigned long-word → Single)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 で指定されるレジスタ・ペアにある符号のない 64 ビットの整数形式の内容を算術的に単精度浮動小数点形式に変換し、汎用レジスタ reg3 に格納します。結果は、現在の丸めモードにしたがって丸めます。

【浮動小数点演算例外】　不正確演算例外（I）

【演算結果】

reg2(A)	+Integer	–Integer	0 (Integer)
演算結果 [例外]	A (Normal)		+0

＜浮動小数点演算命令＞

CVTF.UWS

Floating-point Convert Unsigned-Word to Single (Single)

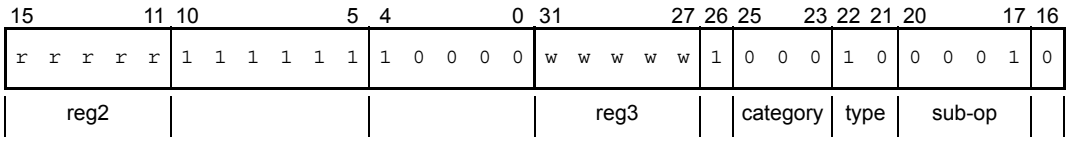
浮動小数点形式への変換（単精度）

【命令形式】　CVTF.UWS reg2, reg3

【オペレーション】　reg3 ← cvt reg2 (unsigned word → Single)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある符号のない 32 ビットの整数形式の内容を算術的に単精度浮動小数点形式に変換し、汎用レジスタ reg3 に格納します。結果は、現在の丸めモードにしたがって丸めます。

【浮動小数点演算例外】　不正確演算例外（I）

【演算結果】

reg2(A)	+Integer	–Integer	0 (Integer)
演算結果 [例外]	A (Normal)		+0

＜浮動小数点演算命令＞

CVTF.WS

Floating-point Convert Word to Single (single)

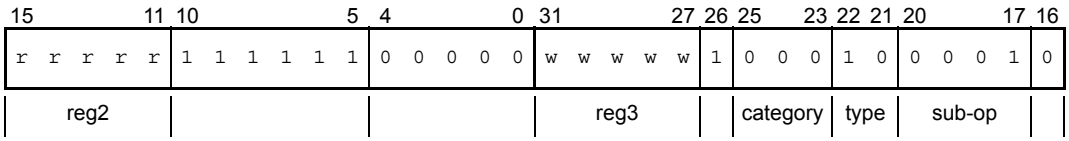
浮動小数点形式への変換（単精度）

【命令形式】　CVTF.WS　reg2, reg3

【オペレーション】　reg3 ← cvt reg2 (word → single)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある 32 ビットの整数形式の内容を算術的に単精度浮動小数点形式に変換し、汎用レジスタ reg3 に格納します。結果は、現在の丸めモードにしたがって丸めます。

【浮動小数点演算例外】　不正確演算例外（I）

【演算結果】

reg2(A)	+Integer	–Integer	0 (Integer)
演算結果 [例外]	A (Normal)		+0

＜浮動小数点演算命令＞

DIVF.S

Floating-point Divide (Single)

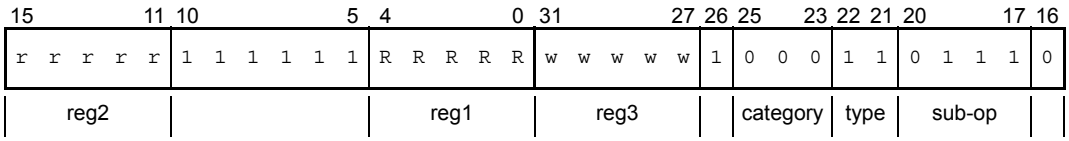
浮動小数点除算（単精度）

【命令形式】　DIVF.S　reg1, reg2, reg3

【オペレーション】　reg3 ← reg2 ÷ reg1

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を汎用レジスタ reg1 にある単精度浮動小数点形式の内容で除算し、汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し、結果を現在の丸めモードにしたがって丸めます。

【浮動小数点演算例外】　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
ゼロ除算例外（Z）
オーバフロー例外（O）
アンダフロー例外（U）

【演算結果】

<table>
<tr><th>reg2(B)
reg1(A)</th><th>Normal</th><th>–Normal</th><th>+0</th><th>–0</th><th>+ ∞</th><th>– ∞</th><th>Q-NaN</th><th>S-NaN</th></tr>
<tr><td>Normal</td><td colspan="4" rowspan="2">B÷A</td><td>+ ∞</td><td>– ∞</td><td rowspan="6"></td><td rowspan="6"></td></tr>
<tr><td>–Normal</td><td>– ∞</td><td>+ ∞</td></tr>
<tr><td>+0</td><td colspan="2" rowspan="2">± ∞ [Z]</td><td colspan="2" rowspan="2">Q-NaN[V]</td><td>+ ∞</td><td>– ∞</td></tr>
<tr><td>–0</td><td>– ∞</td><td>+ ∞</td></tr>
<tr><td>+ ∞</td><td>+0</td><td>-0</td><td>+0</td><td>-0</td><td colspan="2" rowspan="2">Q-NaN[V]</td></tr>
<tr><td>– ∞</td><td>-0</td><td>+0</td><td>-0</td><td>+0</td></tr>
<tr><td>Q-NaN</td><td colspan="7">Q-NaN</td></tr>
<tr><td>S-NaN</td><td colspan="8">Q-NaN[V]</td></tr>
</table>

備考 1.　［　］は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Double to Long, round toward negative (Single)

FLOORF.SL

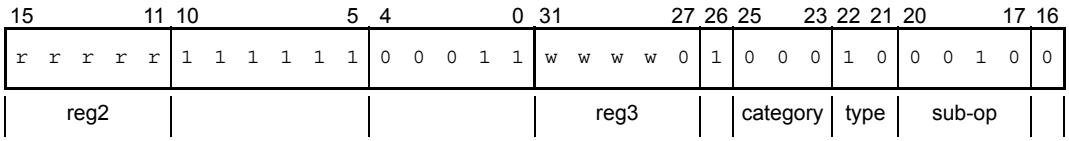
整数形式への変換（単精度）

【命令形式】　　FLOORF.SL　reg2, reg3

【オペレーション】　　reg3 ← floor reg2(single → long-word)

【フォーマット】　　Format F: I

【オペコード】



【説　明】　　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に 64 ビットの整数形式に変換し、汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。

現在の丸めモードにかかわりなく、結果を－∞の方向へ丸めます。

ソース・オペランドが無限大か非数の場合、または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞　　：$2^{63}-1$ を返します。
- ソースが負数、非数または－∞　　：-2^{63} を返します。

【浮動小数点演算例外】　　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A (Integer)		0 (Integer)		Max Int[V]	–Max Int[V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Unsigned-Long, round toward negative (Single)

FLOORF.SUL

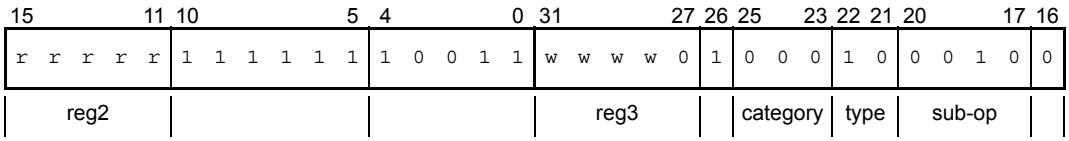
符号なし整数形式への変換（単精度）

【命令形式】　FLOORF.SUL　reg2, reg3

【オペレーション】　reg3 ← floor reg2 (Single → Unsigned long-word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に符号のない 64 ビットの整数形式に変換し、汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。

現在の丸めモードに関係なく、結果を－∞の方向へ丸めます。

ソース・オペランドが無限大か非数か負数の場合、または丸めた結果が $2^{64}-1 \sim 0$ の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの値により返す値は次のように異なります。

- ソースが $2^{64}-1 \sim 0$ の範囲外の正数または＋∞　：$2^{64}-1$ を返します。
- ソースが負数、非数または－∞　：0 を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A (Integer)	0 [V]	0 (Integer)		Max U-Int [V]	0 [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Unsigned-Word, round toward negative (Single)

FLOORF.SUW

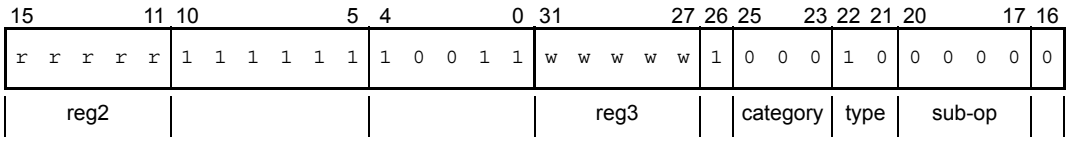
符号なし整数形式への変換（単精度）

【命令形式】　FLOORF.SUW　reg2, reg3

【オペレーション】　reg3 ← floor reg2 (Single → Unsigned word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に符号のない 32 ビットの整数形式に変換し、汎用レジスタ reg3 に格納します。

現在の丸めモードに関係なく、結果を – ∞の方向へ丸めます。

ソース・オペランドが無限大か非数か負数の場合、または丸めた結果が 2^{32} –1 ～ 0 の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの値により返す値は次のように異なります。

- ソースが 2^{32} –1 ～ 0 の範囲外の正数または + ∞ ：2^{32} –1 を返します。
- ソースが負数、非数または – ∞　　：0 を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A (Integer)	0 [V]	0 (Integer)		Max U-Int [V]	0 [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

FLOORF.SW

Floating-point Convert Single to Word, round toward negative (Single)

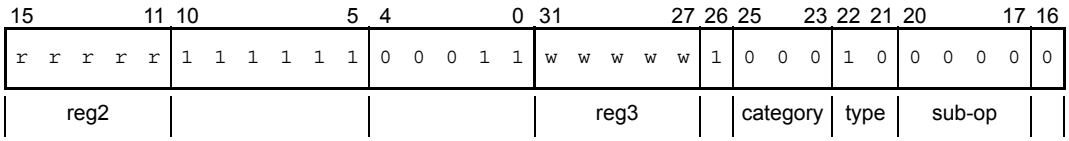
整数形式への変換（単精度）

【命令形式】　FLOORF.SW　reg2, reg3

【オペレーション】　reg3 ← floor reg2 (single → word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し、汎用レジスタ reg3 に格納します。

現在の丸めモードに関係なく、– ∞の方向へ丸めます。

ソース・オペランドが無限大か非数の場合、または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または + ∞　　: $2^{31}-1$ を返します。
- ソースが負数、非数または – ∞　　: -2^{31} を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A (Integer)		0 (Integer)		+Max Int[V]	–Max Int[V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

FMAF.S

Floating-point Fused-Multiply-add (Single)

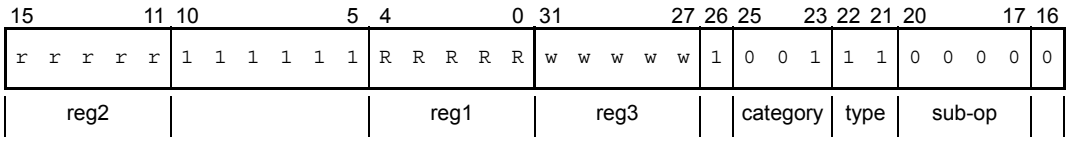
浮動小数点結合積和算（単精度）

【命令形式】　FMAF.S　reg1, reg2, reg3

【オペレーション】　reg3 ← fma (reg2, reg1, reg3)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容と汎用レジスタ reg1 にある単精度浮動小数点形式の内容を乗算した結果から、汎用レジスタ reg3 にある単精度浮動小数点形式の内容を加算し、結果を汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し乗算結果の丸めは行いません、加算結果を現在の丸めモードにしたがって丸めます。

【浮動小数点演算例外】　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
オーバフロー例外（O）
アンダフロー例外（U）

【演算結果】

reg3(C) \ reg2(B) / reg1(A)		+ Normal	- Normal	+0	-0	+ ∞	– ∞	Q-NaN	S-NaN
±Normal	+Normal	FMA (A, B, C)				+ ∞	– ∞		
	-Normal					– ∞	+ ∞		
	±0					Q-NaN[V]			
	+ ∞	+ ∞	- ∞	Q-NaN[V]		+ ∞	– ∞		
	– ∞	- ∞	+ ∞			– ∞	+ ∞		
±0	+Normal	FMA (A, B, C)				+ ∞	– ∞		
	-Normal					– ∞	+ ∞		
	±0					Q-NaN[V]			
	+ ∞	+ ∞	– ∞	Q-NaN[V]		+ ∞	– ∞		
	– ∞	– ∞	+ ∞			– ∞	+ ∞		
+ ∞	+Normal	+ ∞				+ ∞	Q-NaN[V]		
	-Normal					Q-NaN[V]	+ ∞		
	±0					Q-NaN[V]			
	+ ∞	+ ∞	Q-NaN[V]	Q-NaN[V]		+ ∞	Q-NaN[V]		
	– ∞	Q-NaN[V]	+ ∞			Q-NaN[V]	+ ∞		
– ∞	+Normal	– ∞				Q-NaN[V]	– ∞		
	-Normal					– ∞	Q-NaN[V]		
	±0					Q-NaN[V]			
	+ ∞	Q-NaN[V]	– ∞	Q-NaN[V]		Q-NaN[V]	– ∞		
	– ∞	– ∞	Q-NaN[V]			– ∞	Q-NaN[V]		
Q-NaN	±Normal	Q-NaN							
	±0								
	± ∞								
S-NaN 以外	Q-NaN	Q-NaN						Q-NaN	
任意	S-NaN	Q-NaN[V]							
S-NaN	任意								

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

【補　足】　乗算結果を無限精度であるかのように実行し、積和演算結果を現在の丸めモードにしたがって丸めます。そのため、演算結果は ADDF, MULF 命令を組み合わせた結果とは異なります。

＜浮動小数点演算命令＞

FMSF.S

Floating-point Fused-Multiply-subtract (Single)

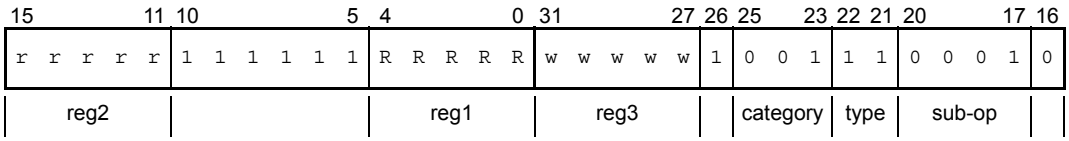
浮動小数点結合積差算（単精度）

【命令形式】　　FMSF.S　reg1, reg2, reg3

【オペレーション】　　reg3 ← fms (reg2, reg1, reg3)

【フォーマット】　　Format F: I

【オペコード】



【説　明】　　汎用レジスタ reg2 にある単精度浮動小数点形式の内容と汎用レジスタ reg1 にある単精度浮動小数点形式の内容を乗算した結果から、汎用レジスタ reg3 にある単精度浮動小数点形式の内容を減算し、結果を汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し乗算結果の丸めは行いません、減算結果を現在の丸めモードにしたがって丸めます。

【浮動小数点演算例外】　　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
オーバフロー例外（O）
アンダフロー例外（U）

【演算結果】

reg3(C) \ reg2(B) / reg1(A)		+ Normal	- Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
±Normal	+Normal	FMS (A, B, C)				+ ∞	– ∞		
	-Normal					– ∞	+ ∞		
	±0					Q-NaN[V]			
	+ ∞	+ ∞	– ∞	Q-NaN[V]		+ ∞	– ∞		
	– ∞	– ∞	+ ∞			– ∞	+ ∞		
±0	+Normal	FMS (A, B, C)				+ ∞	– ∞		
	-Normal					– ∞	+ ∞		
	±0					Q-NaN[V]			
	+ ∞	+ ∞	– ∞	Q-NaN[V]		+ ∞	– ∞		
	– ∞	– ∞	+ ∞			– ∞	+ ∞		
+ ∞	+Normal	– ∞				Q-NaN[V]	– ∞		
	-Normal					– ∞	Q-NaN[V]		
	±0					Q-NaN[V]			
	+ ∞	Q-NaN[V]	– ∞	Q-NaN[V]		Q-NaN[V]	– ∞		
	– ∞	– ∞	Q-NaN[V]			– ∞	Q-NaN[V]		
– ∞	+Normal	+ ∞				+ ∞	Q-NaN[V]		
	-Normal					Q-NaN[V]	+ ∞		
	±0					Q-NaN[V]			
	+ ∞	+ ∞	Q-NaN[V]	Q-NaN[V]		+ ∞	Q-NaN[V]		
	– ∞	Q-NaN[V]	+ ∞			Q-NaN[V]	+ ∞		
Q-NaN	±Normal	Q-NaN							
	±0								
	± ∞								
S-NaN 以外	Q-NaN							Q-NaN	
任意	S-NaN								Q-NaN[V]
S-NaN	任意								

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

【補　足】　乗算結果を無限精度であるかのように実行し、積差演算結果を現在の丸めモードにしたがって丸めます。そのため、演算結果は SUBF, MULF 命令を組み合わせた結果とは異なります。

＜浮動小数点演算命令＞

FNMAF.S

Floating-point Fused-Negate-Multiply-add (Single)

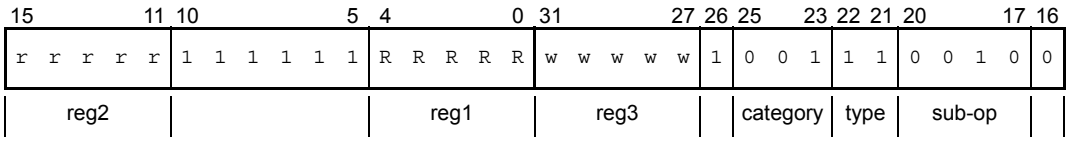
浮動小数点結合積和算（単精度）

【命令形式】　FNMAF.S　reg1, reg2, reg3

【オペレーション】　reg3 ← neg (fma(reg2, reg1, reg3))

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容と汎用レジスタ reg1 にある単精度浮動小数点形式の内容を乗算した結果から、汎用レジスタ reg3 にある単精度浮動小数点形式の内容を加算し、符号を反転して汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し乗算結果の丸めは行いません、加算結果を現在の丸めモードにしたがって丸めます。符号の反転は丸め後に行います。

【浮動小数点演算例外】　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
オーバフロー例外（O）
アンダフロー例外（U）

【演算結果】

reg3(C) \ reg2(B) / reg1(A)		+ Normal	– Normal	+0	-0	+ ∞	– ∞	Q-NaN	S-NaN
±Normal	+Normal	FNMA (A, B, C)				– ∞	+ ∞		
	-Normal					+ ∞	– ∞		
	±0					Q-NaN[V]			
	+ ∞	– ∞	+ ∞	Q-NaN[V]		– ∞	+ ∞		
	– ∞	+ ∞	– ∞			+ ∞	– ∞		
±0	+Normal	FNMA (A, B, C)				– ∞	+ ∞		
	-Normal					+ ∞	– ∞		
	±0					Q-NaN[V]			
	+ ∞	– ∞	+ ∞	Q-NaN[V]		– ∞	+ ∞		
	– ∞	+ ∞	– ∞			+ ∞	– ∞		
+ ∞	+Normal	– ∞				– ∞	Q-NaN[V]		
	-Normal					Q-NaN[V]	– ∞		
	±0					Q-NaN[V]			
	+ ∞	– ∞	Q-NaN[V]	Q-NaN[V]		– ∞	Q-NaN[V]		
	– ∞	Q-NaN[V]	– ∞			Q-NaN[V]	– ∞		
– ∞	+Normal	+ ∞				Q-NaN[V]	+ ∞		
	-Normal					+ ∞	Q-NaN[V]		
	±0					Q-NaN[V]			
	+ ∞	Q-NaN[V]	+ ∞	Q-NaN[V]		Q-NaN[V]	+ ∞		
	– ∞	+ ∞	Q-NaN[V]			+ ∞	Q-NaN[V]		
Q-NaN	±Normal	Q-NaN							
	±0								
	± ∞								
S-NaN 以外	Q-NaN	Q-NaN						Q-NaN	
任意	S-NaN	Q-NaN[V]							
S-NaN	任意								

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

【補　足】　乗算結果を無限精度であるかのように実行し、積和演算結果を現在の丸めモードにしたがって丸めます。そのため、演算結果は ADDF, MULF, NEGF 命令を組み合わせた結果とは異なります。

＜浮動小数点演算命令＞

FNMSF.S

Floating-point Fused-Negate-Multiply-subtract (Single)

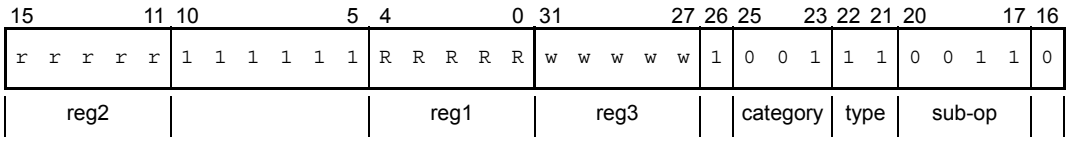
浮動小数点結合積差算（単精度）

【命令形式】　FNMSF.S　reg1, reg2, reg3

【オペレーション】　reg3 ← neg (fms(reg2, reg1, reg3))

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容と汎用レジスタ reg1 にある単精度浮動小数点形式の内容を乗算した結果から、汎用レジスタ reg3 にある単精度浮動小数点形式の内容を減算し、符号を反転して汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し乗算結果の丸めは行いません。減算結果を現在の丸めモードにしたがって丸めます。符号の反転は丸め後に行います。

【浮動小数点演算例外】　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
オーバフロー例外（O）
アンダフロー例外（U）

【演算結果】

reg3(C) \ reg2(B) / reg1(A)		+ Normal	– Normal	+0	-0	+ ∞	– ∞	Q-NaN	S-NaN
±Normal	+Normal	FNMS (A, B, C)				– ∞	+ ∞		
	–Normal					+ ∞	– ∞		
	±0					Q-NaN[V]			
	+ ∞	– ∞	+ ∞	Q-NaN[V]		– ∞	+ ∞		
	– ∞	+ ∞	– ∞			+ ∞	– ∞		
±0	+Normal	FNMS (A, B, C)				– ∞	+ ∞		
	–Normal					+ ∞	– ∞		
	±0					Q-NaN[V]			
	+ ∞	– ∞	+ ∞	Q-NaN[V]		– ∞	+ ∞		
	– ∞	+ ∞	– ∞			+ ∞	– ∞		
+ ∞	+Normal	+ ∞				Q-NaN[V]	+ ∞		
	–Normal					+ ∞	Q-NaN[V]		
	±0					Q-NaN[V]			
	+ ∞	Q-NaN[V]	+ ∞	Q-NaN[V]		Q-NaN[V]	+ ∞		
	– ∞	+ ∞	Q-NaN[V]			+ ∞	Q-NaN[V]		
– ∞	+Normal	– ∞				– ∞	Q-NaN[V]		
	-Normal					Q-NaN[V]	– ∞		
	±0					Q-NaN[V]			
	+ ∞	– ∞	Q-NaN[V]	Q-NaN[V]		– ∞	Q-NaN[V]		
	– ∞	Q-NaN[V]	– ∞			Q-NaN[V]	– ∞		
Q-NaN	±Normal	Q-NaN							
	±0								
	± ∞								
S-NaN 以外	Q-NaN							Q-NaN	
任意	S-NaN								Q-NaN[V]
S-NaN	任意								

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

【補　足】　乗算結果を無限精度であるかのように実行し、積差演算結果を現在の丸めモードにしたがって丸めます。そのため、演算結果は SUBF, MULF, NEGF 命令を組み合わせた結果とは異なります。

＜浮動小数点演算命令＞

MAXF.S

Floating-point Maximum (Single)

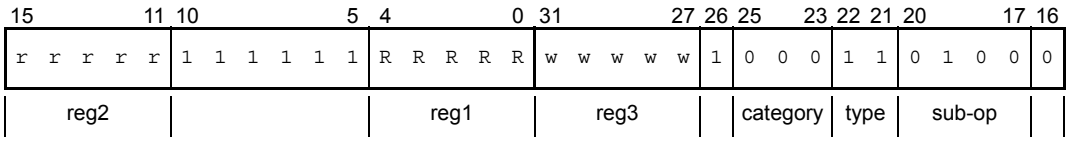
浮動小数点最大値（単精度）

【命令形式】　MAXF.S　reg1, reg2, reg3

【オペレーション】　reg3 ← max (reg2, reg1)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg1 および reg2 にある単精度浮動小数点形式のデータの中から最大値を汎用レジスタ reg3 に格納します。

ソース・オペランドの 1 つが S-NaN の場合は、IEEE754 の無効演算例外を検出します。無効演算例外の発生が許可されていない場合は、例外を発生せず、Q-NaN を格納します。

【浮動小数点演算例外】　無効演算例外（V）

【補　足】　reg1 および reg2 がともに＋ 0、–0 のいずれかである場合、reg3 に＋ 0、–0 いずれを格納するかは未定義です。

FPSR.FS = 1 でもサブノーマル数の入力はフラッシュされません。

【演算結果】

reg2(B) / reg1(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
Normal	MAX (A, B)						reg1 (A)	
–Normal								
+0								
-0								
+ ∞								
– ∞								
Q-NaN	reg2 (A)						Q-NaN	
S-NaN	Q-NaN[V]							

備考　［　］は必ず発生する例外です。

＜浮動小数点演算命令＞

Floating-point Minimum (Single)

MINF.S

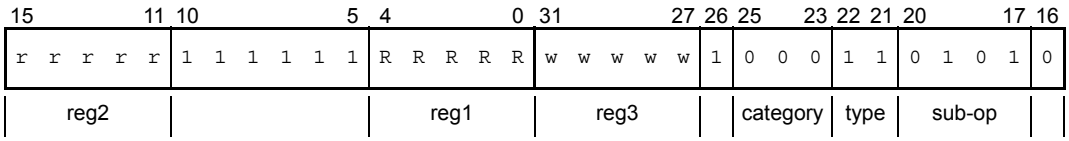
浮動小数点最小値（単精度）

【命令形式】　MINF.S　reg1, reg2, reg3

【オペレーション】　reg3 ← min (reg2, reg1)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg1 および reg2 にある単精度浮動小数点形式のデータの中から最小値を汎用レジスタ reg3 に格納します。

ソース・オペランドの 1 つが S-NaN の場合は、IEEE754 の無効演算例外を検出します。無効演算例外の発生が許可されていない場合は、例外を発生せず、Q-NaN を格納します。

【浮動小数点演算例外】　無効演算例外（V）

【補　足】　reg1 および reg2 がともに＋ 0、–0 のいずれかである場合、reg3 に＋ 0、–0 いずれを格納するかは未定義です。

FPSR.FS = 1 でもサブノーマル数の入力はフラッシュされません。

【演算結果】

reg2(B) / reg1(A)	Normal	-Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
Normal	MIN (A, B)						reg1 (A)	
–Normal								
+0								
-0								
+ ∞								
– ∞								
Q-NaN	reg2 (B)						Q-NaN	
S-NaN	Q-NaN[V]							

備考 1.　[　] は必ず発生する例外です。

＜浮動小数点演算命令＞

MULF.S

Floating-point Multiply (Single)

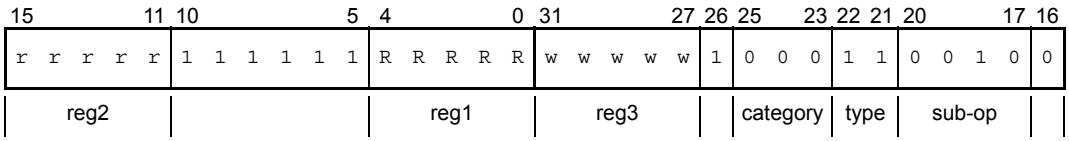
浮動小数点乗算（単精度）

【命令形式】　MULF.S　reg1, reg2, reg3

【オペレーション】　reg3 ← reg2 × reg1

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容と汎用レジスタ reg1 にある単精度浮動小数点形式の内容を乗算し、汎用レジスタ reg3 に格納します。

【浮動小数点演算例外】　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
オーバフロー例外（O）
アンダフロー例外（U）

【演算結果】

reg1(A) \ reg2(B)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
Normal	A × B				+ ∞	– ∞	Q-NaN	Q-NaN[V]
–Normal					– ∞	+ ∞		
+0					Q-NaN[V]			
–0								
+ ∞	+ ∞	– ∞	Q-NaN[V]		+ ∞	– ∞		
– ∞	– ∞	+ ∞			– ∞	+ ∞		
Q-NaN	Q-NaN							
S-NaN	Q-NaN[V]							

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

NEGF.S

Floating-point Negate (Single)

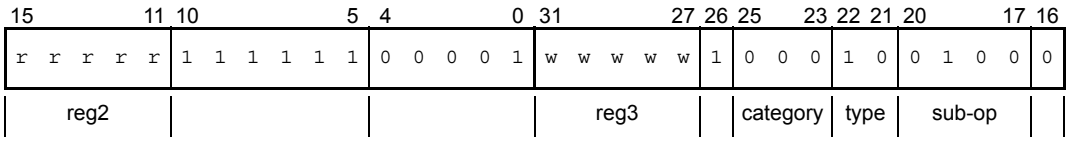
浮動小数点符号反転（単精度）

【命令形式】　NEGF.S　reg2, reg3

【オペレーション】　reg3 ← neg reg2

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容の符号を反転し、汎用レジスタ reg3 に格納します。

【浮動小数点演算例外】　なし

【補　足】　FPSR.FS = 1 でもサブノーマル数の入力はフラッシュされません。

＜浮動小数点演算命令＞

RECIPF.S

Reciprocal of a Floating-point Value (Single)

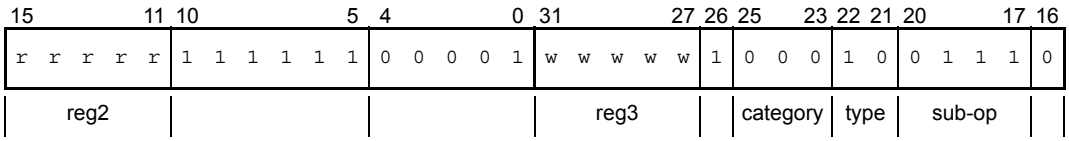
逆数（単精度）

【命令形式】　RECIPF.S　reg2, reg3

【オペレーション】　reg3 ← 1 ÷ reg2

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容の逆数を近似し、汎用レジスタ reg3 に格納します。演算結果は DIVF 命令を用いた結果と異なります。

【浮動小数点演算例外】　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
ゼロ除算例外（Z）
アンダフロー例外（U）

【演算結果】

reg2(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	1/A[I]		+ ∞ [Z]	– ∞ [Z]	+0	–0	Q-NaN	Q-NaN[V]

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Long, round to nearest (Single)

ROUNDF.SL

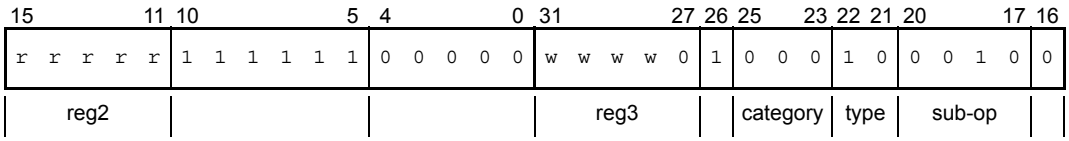
整数形式への変換（単精度）

【命令形式】　　ROUNDF.SL　reg2, reg3

【オペレーション】　　reg3 ← round reg2 (single → long-word)

【フォーマット】　　Format F: I

【オペコード】



【説　明】　　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に 64 ビットの整数形式に変換し、汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。

現在の丸めモードに関係なく、結果を最も近い値または偶数に丸めます。

ソース・オペランドが無限大か非数の場合、または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または + ∞ : $2^{63}-1$ を返します。
- ソースが負数、非数または – ∞ : -2^{63} を返します。

【浮動小数点演算例外】　　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A（Integer）		0（Integer）		Max Int[V]	-Max Int[V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

ROUNDF.SUL

Floating-point Convert Single to Unsigned-Long, round to nearest (Single)

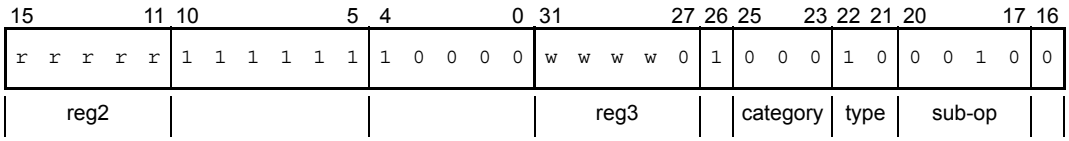
符号なし整数形式への変換（単精度）

【命令形式】　ROUNDF.SUL　reg2, reg3

【オペレーション】　reg3 ← round reg2(Single → Unsigned long-word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に符号のない 64 ビットの整数形式に変換し、汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。

現在の丸めモードに関係なく、結果を最も近い値もしくは偶数に丸めます。

ソース・オペランドが無限大か非数か負数の場合、または丸めた結果が $2^{64}-1$ ～ 0 の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は，例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの値により返す値は次のように異なります。

- ソースが $2^{64}-1$ ～ 0 の範囲外の正数または + ∞ : $2^{64}-1$ を返します。
- ソースが負数，非数または – ∞ : 0 を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A （Integer）	0 [V]	0（Integer）		Max U-Int [V]	0 [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Unsigned-Word, round to nearest (Single)

ROUNDF.SUW

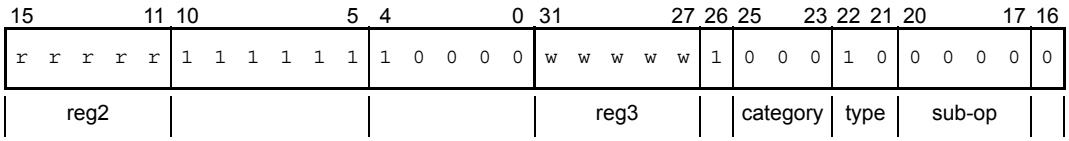
符号なし整数形式への変換（単精度）

【命令形式】　ROUNDF.SUW　reg2, reg3

【オペレーション】　reg3 ← round reg2(Single → Unsigned word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点数形式の内容を算術的に符号のない 32 ビットの整数形式に変換し、汎用レジスタ reg3 に格納します。

現在の丸めモードに関係なく、結果を最も近い値もしくは偶数に丸めます。

ソース・オペランドが無限大か非数か負数の場合、または丸めた結果が $2^{32}-1 \sim 0$ の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの値により返す値は次のように異なります。

- ソースが $2^{32}-1 \sim 0$ の範囲外の正数または $+\infty$: $2^{32}-1$ を返します。
- ソースが負数，非数または $-\infty$: 0 を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A （Integer）	0 [V]	0（Integer）		Max U-Int [V]	0 [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Word, round to nearest (Single)

ROUNDF.SW

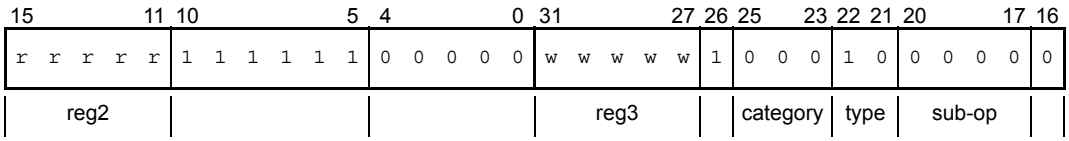
整数形式への変換（単精度）

【命令形式】　　ROUNDF.SW　reg2, reg3

【オペレーション】　　reg3 ← round reg2 (single → word)

【フォーマット】　　Format F: I

【オペコード】



【説　明】　　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し、汎用レジスタ reg3 に格納します。

現在の丸めモードに関係なく、結果を最も近い値または偶数に丸めます。

ソース・オペランドが無限大か非数の場合、または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または + ∞ : $2^{31}-1$ を返します。
- ソースが負数，非数または – ∞ : -2^{31} を返します。

【浮動小数点演算例外】　　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A（Integer）		0（Integer）		Max Int[V]	–Max Int[V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Reciprocal of the Square Root of a Floating-point Value (Single)

RSQRTF.S

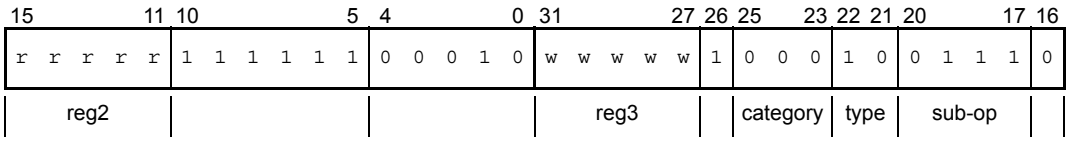
平方根の逆数（単精度）

【命令形式】　RSQRTF.S　reg2, reg3

【オペレーション】　reg3 ← 1 ÷ (sqrt reg2)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容の正の算術平方根の逆数を近似し、汎用レジスタ reg3 に格納します。演算結果は SQRTF, DIVF 命令を組み合わせた結果と異なります。

【浮動小数点演算例外】　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
ゼロ除算例外（Z）

【演算結果】

reg2(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	1/√A[I]	Q-NaN[V]	+ ∞ [Z]	– ∞ [Z]	+0	Q-NaN[V]	Q-NaN	Q-NaN[V]

備考 1.　[　] は必ず発生する例外です。
備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

SQRTF.S

Floating-point Square Root (Single)

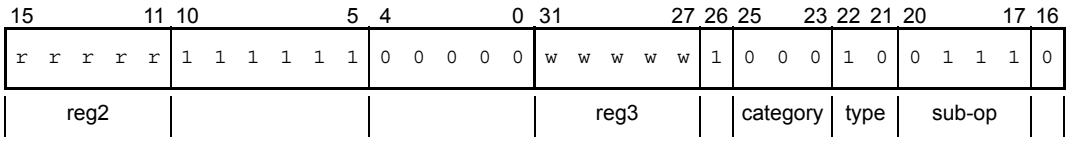
平方根（単精度）

【命令形式】　　SQRTF.S　reg2, reg3

【オペレーション】　　reg3 ← sqrt reg2

【フォーマット】　　Format F: I

【オペコード】



【説　明】　　汎用レジスタ reg2 にある単精度浮動小数点形式の内容の正の算術平方根を求め、汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し、結果を現在の丸めモードにしたがって丸めます。ソース・オペランドの値が –0 の場合、結果は –0 になります。

【浮動小数点演算例外】　　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	√A	Q-NaN[V]	+0	–0	+ ∞	Q-NaN[V]	Q-NaN	Q-NaN[V]

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

SUBF.S

Floating-point Subtract (Single)

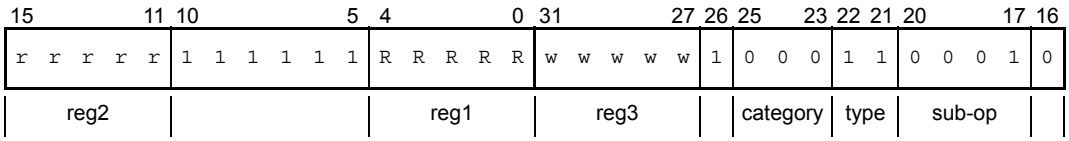
浮動小数点減算（単精度）

【命令形式】　SUBF.S　reg1, reg2, reg3

【オペレーション】　reg3 ← reg2 – reg1

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容から汎用レジスタ reg1 にある単精度浮動小数点形式の内容を減算し、汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し、結果を現在の丸めモードにしたがって丸めます。

【浮動小数点演算例外】　未実装演算例外（E）
無効演算例外（V）
不正確演算例外（I）
オーバフロー例外（O）
アンダフロー例外（U）

【演算結果】

reg1(A) \ reg2(B)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
Normal	B – A				+ ∞	– ∞	Q-NaN	Q-NaN[V]
–Normal								
+0								
–0								
+ ∞	– ∞				Q-NaN[V]			
– ∞	+ ∞					Q-NaN[V]		
Q-NaN	Q-NaN							
S-NaN	Q-NaN[V]							

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

TRFSR

Transfers spefified CC bit to Zero flag in PSW (Single)

フラグ転送

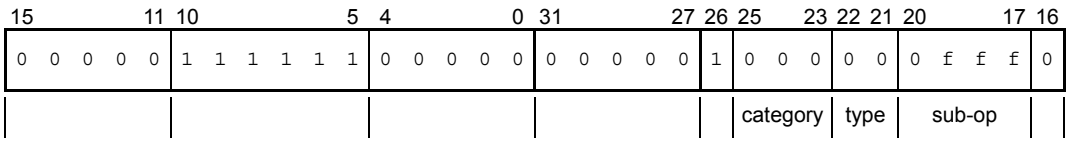
【命令形式】　TRFSR　fcbit

TRFSR

【オペレーション】　PSW.Z ← fcbit

【フォーマット】　Format F: I

【オペコード】



備考　fcbit：fff

【説　明】　fcbit によって指定された FPSR レジスタのコンディション・ビット（CC(7:0) ビット：ビット 31 ～ 24）を、PSW 内の Z フラグへ転送します。fcbit が省略された場合は CC0（ビット 24）を転送します。

【浮動小数点演算例外】　なし

＜浮動小数点演算命令＞

TRNCF.SL

Floating-point Convert Single to Long, round toward zero (Single)

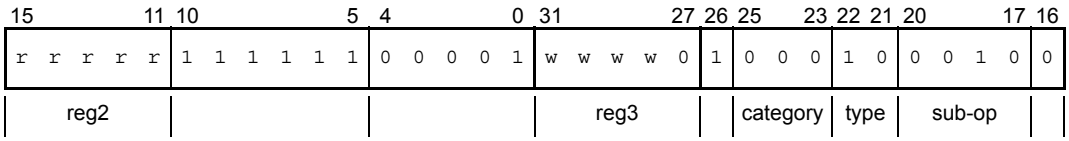
整数形式への変換（単精度）

【命令形式】　TRNCF.SL　reg2, reg3

【オペレーション】　reg3 ← trunc reg2 (single → long-word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に 64 ビットの整数形式に変換し、汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。

現在の丸めモードに関係なく、結果を 0 の方向へ丸めます。

ソース・オペランドが無限大か非数の場合、または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または + ∞　　: $2^{63}-1$ を返します。
- ソースが負数、非数または – ∞　　: -2^{63} を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A (Integer)		0 (Integer)		Max Int[V]	–Max Int[V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

TRNCF.SUL

Floating-point Convert Single to Unsigned-Long, round toward zero (Single)

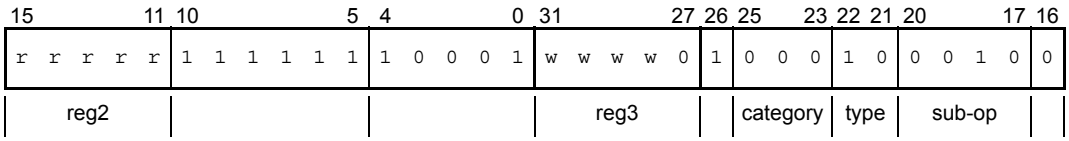
符号なし整数形式への変換（単精度）

【命令形式】　TRNCF.SUL　reg2, reg3

【オペレーション】　reg3 ← trunc reg2 (Single → Unsigned long-word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に符号のない 64 ビットの整数形式に変換し、汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。

現在の丸めモードに関係なく、結果を 0 の方向へ丸めます。

ソース・オペランドが無限大か非数か負数の場合、または丸めた結果が $2^{64}-1 \sim 0$ の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの値により返す値は次のように異なります。

- ソースが $2^{64}-1 \sim 0$ の範囲外の正数または $+\infty$ ：$2^{64}-1$ を返します。
- ソースが負数、非数または $-\infty$ ：0 を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A (Integer)	0 [V]	0 (Integer)		Max U-Int [V]	0 [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

Floating-point Convert Single to Unsigned-Word, round toward zero (Single)

TRNCF.SUW

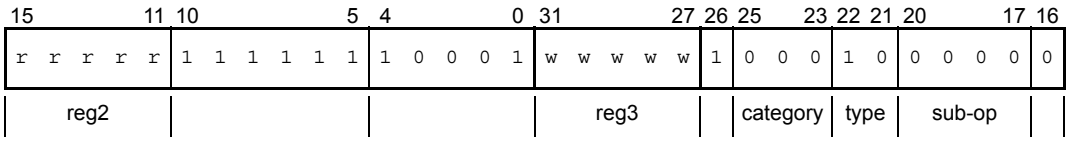
符号なし整数形式への変換（単精度）

【命令形式】　TRNCF.SUW　reg2, reg3

【オペレーション】　reg3 ← trunc reg2 (Single → Unsigned word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に符号のない 32 ビットの整数形式に変換し、汎用レジスタ reg3 に格納します。

現在の丸めモードに関係なく、結果を 0 の方向へ丸めます。

ソース・オペランドが無限大か非数か負数の場合、または丸めた結果が $2^{32}-1 \sim 0$ の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの値により返す値は次のように異なります。

- ソースが $2^{32}-1 \sim 0$ の範囲外の正数または $+\infty$ ：$2^{32}-1$ を返します。
- ソースが負数、非数または $-\infty$ ：0 を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	+Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A (Integer)	0 [V]	0 (Integer)		Max U-Int [V]	0 [V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

＜浮動小数点演算命令＞

TRNCF.SW

Floating-point Convert Single to Word、round toward zero (Single)

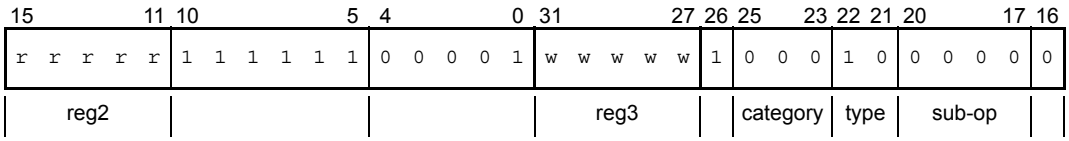
整数形式への変換（単精度）

【命令形式】　TRNCF.SW　reg2, reg3

【オペレーション】　reg3 ← trunc reg2 (single → word)

【フォーマット】　Format F: I

【オペコード】



【説　明】　汎用レジスタ reg2 にある単精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し、汎用レジスタ reg3 に格納します。

現在の丸めモードに関係なく、結果を 0 の方向へ丸めます。

ソース・オペランドが無限大か非数の場合、または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は、IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は、例外を発生せず、無効演算として FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または + ∞　　：$2^{31}-1$ を返します。
- ソースが負数、非数または – ∞　　：-2^{31} を返します。

【浮動小数点演算例外】　未実装演算例外（E）

無効演算例外（V）

不正確演算例外（I）

【演算結果】

reg2(A)	Normal	–Normal	+0	–0	+ ∞	– ∞	Q-NaN	S-NaN
演算結果 [例外]	A (Integer)		0 (Integer)		Max Int[V]	–Max Int[V]		

備考 1.　[　] は必ず発生する例外です。

備考 2.　FPSR.FS = 1 のとき、サブノーマル数は**「6.1.9　サブノーマル数のフラッシュ」**で示す正規化数にフラッシュされます。

第 8 章　リセット

8.1　リセット後のレジスタの状態

ハードウェア仕様によって定義されたリセット入力方法によってリセットが指示された場合、プログラム・レジスタとシステム・レジスタは、**「第 3 章　レジスタ・セット」**の各レジスタのリセット後の値が示す状態になり、プログラムの実行を開始します。レジスタの内容は、プログラムの中で必要に応じて適切な値に設定を行ってください。

CPU はリセットにより、**「4.5　例外ハンドラ・アドレス」**によって定められたリセット・アドレスからプログラムの実行を開始します。

なお、リセット直後は、PSW.ID ビットがセット（1）されているため、受け付け条件のある EI レベル例外は受け付けられません。受け付け条件のある EI レベル例外を使用する場合は、PSW.ID ビットをクリア（0）してください。

付録 A　システム・レジスタのハザード解消手続き

一部のシステム・レジスタでは、LDSR 命令によりレジスタ値を更新する際のハザードを解消するために、以下の手続きが必要です。

- 命令フェッチ

  以下のレジスタを更新してから命令フェッチする場合、レジスタを更新する命令を実行後に、EIRET 命令、FERET 命令または SYNCI 命令を実行してから、命令フェッチを開始してください。

  – PSW.UM，MCFG0.SPID

  以下のレジスタを更新してから命令フェッチする場合、レジスタを更新する命令を実行後に、SYNCI 命令を実行してから命令フェッチを開始してください。

  – ASID、MPU 関連すべてのレジスタ（レジスタ番号：SR*, 5-7）

- SYSCALL 命令

  以下のレジスタを更新してから SYSCALL 命令を実行する場合、レジスタを更新する命令を実行後に、SYNCP 命令を実行してから SYSCALL 命令を実行してください。

  – SCCFG

- ロード／ストア

  以下のレジスタを更新してからロード／ストアをともなう命令を実行する場合、レジスタを更新する命令を実行後に、SYNCP 命令を実行してからロード／ストア命令を実行してください。

  – ASID、MPU 保護領域設定レジスタ（レジスタ番号：SR*, 6-7）

- 割り込み

  以下のレジスタは、割り込み禁止状態（PSW.ID = 1）で更新してください。

  – PSW.EBV、EBASE、INTBP、FPIPR、ISPR、PMR、ICSR、INTCFG

- FPU レジスタの更新

  以下のレジスタを更新する命令を実行した後に、SYNCP 命令、EIRET 命令、または FERET 命令を実行してください。

  – FPU 関連すべてのレジスタ（レジスタ番号：SR6-11, 0）

- FPP/FPI 例外モードの変更

  FPP/FPI 例外モードを変更する場合、SYNCP 命令と SYNCE 命令を実行してから以下のレジスタを更新してください。また、レジスタを更新するために、上記「FPU レジスタの更新」の手続きも合わせて実施してください。

  – FPSR.PEM

備　考

SYNCP 命令、SYNCE 命令、上記レジスタを更新する命令の間に FPP/FPI 例外を発生する浮動小数点演算命令以外の命令を実行することは可能です。

- コプロセッサ命令

  以下のレジスタを更新してからコプロセッサ命令（浮動小数点演算命令）を実行する場合、レジスタを更新する命令を実行後に、EIRET 命令、FERET 命令、SYNCI 命令、またはSYNCP 命令を実行してから、コプロセッサ命令を実行してください。

  – PSW.CU0

付録 B　G3KH 命令実行クロック数

(1) 基本命令

命令の種類	ニーモニック	オペランド	命令長（バイト数）	実行クロック数		
				issue	repeat	latency
ロード命令	LD.B	disp16 [reg1] , reg2	4	1	1	2 注1
		disp23 [reg1] , reg3	6	1	1	2 注1
	LD.BU	disp16 [reg1] , reg2	4	1	1	2 注1
		disp23 [reg1] , reg3	6	1	1	2 注1
	LD.H	disp16 [reg1] , reg2	4	1	1	2 注1
		disp23 [reg1] , reg3	6	1	1	2 注1
	LD.HU	disp16 [reg1] , reg2	4	1	1	2 注1
		disp23 [reg1] , reg3	6	1	1	2 注1
	LD.W	disp16 [reg1] , reg2	4	1	1	2 注1
		disp23 [reg1] , reg3	6	1	1	2 注1
	LD.DW	disp23 [reg1] , reg3	6	2	2	3 注1
ep 相対	SLD.B	disp7 [ep] , reg2	2	1	1	2 注1
	SLD.BU	disp4 [ep] , reg2	2	1	1	2 注1
	SLD.H	disp8 [ep] , reg2	2	1	1	2 注1
	SLD.HU	disp5 [ep] , reg2	2	1	1	2 注1
	SLD.W	disp8 [ep] , reg2	2	1	1	2 注1
ストア命令	ST.B	reg2, disp16 [reg1]	4	3 注4	3 注4	3 注1 注4
		reg3, disp23 [reg1]	6	3 注4	3 注4	3 注1 注4
	ST.H	reg2, disp16 [reg1]	4	3 注4	3 注4	3 注1 注4
		reg3, disp23 [reg1]	6	3 注4	3 注4	3 注1 注4
	ST.W	reg2, disp16 [reg1]	4	1	1	3 注1
		reg3, disp23 [reg1]	6	1	1	3 注1
	ST.DW	reg3, disp23 [reg1]	6	2	2	3 注1
ep 相対	SST.B	reg2, disp7 [ep]	2	3 注4	3 注4	3 注1 注4
	SST.H	reg2, disp8 [ep]	2	3 注4	3 注4	3 注1 注4
	SST.W	reg2, disp8 [ep]	2	1	1	1 注1
乗算命令	MUL	reg1, reg2, reg3	4	2	2	4
		imm9, reg2, reg3	4	2	2	4
	MULH	reg1, reg2	2	1	1	3
		imm5, reg2	2	1	1	3
	MULHI	imm16, reg1, reg2	4	1	1	3
	MULU	reg1, reg2, reg3	4	2	2	4
		imm9, reg2, reg3	4	2	2	4
加算付き乗算	MAC	reg1, reg2, reg3, reg4	4	3	3	5
	MACU	reg1, reg2, reg3, reg4	4	3	3	5

命令の種類	ニーモニック	オペランド	命令長（バイト数）	実行クロック数		
				issue	repeat	latency
算術演算命令	ADD	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
	ADDI	imm16, reg1, reg2	4	1	1	1
	CMP	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
	MOV	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
		imm32, reg1	6	1	1	1
	MOVEA	imm16, reg1, reg2	4	1	1	1
	MOVHI	imm16, reg1, reg2	4	1	1	1
	SUB	reg1, reg2	2	1	1	1
	SUBR	reg1, reg2	2	1	1	1
条件付き演算	ADF	cccc, reg1, reg2, reg3	4	1	1	1
	SBF	cccc, reg1, reg2, reg3	4	1	1	1
飽和演算	SATADD	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SATSUB	reg1, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SATSUBI	imm16, reg1, reg2	4	1	1	1
	SATSUBR	reg1, reg2	2	1	1	1
論理演算命令	AND	reg1, reg2	2	1	1	1
	ANDI	imm16, reg1, reg2	4	1	1	1
	NOT	reg1, reg2	2	1	1	1
	OR	reg1, reg2	2	1	1	1
	ORI	imm16, reg1, reg2	4	1	1	1
	TST	reg1, reg2	2	1	1	1
	XOR	reg1, reg2	2	1	1	1
	XORI	imm16, reg1, reg2	4	1	1	1

命令の種類	ニーモニック	オペランド	命令長（バイト数）	実行クロック数		
				issue	repeat	latency
データ操作命令	BINS	reg1, pos, width, reg2	4	1	1	1
	BSH	reg2, reg3	4	1	1	1
	BSW	reg2, reg3	4	1	1	1
	CMOV	cccc, reg1, reg2, reg3	4	1	1	1
		cccc, imm5, reg2, reg3	4	1	1	1
	HSH	reg2, reg3	4	1	1	1
	HSW	reg2, reg3	4	1	1	1
	ROTL	imm5, reg2, reg3	4	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SAR	reg1, reg2	4	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SASF	cccc, reg2	4	1	1	1
	SETF	cccc, reg2	4	1	1	1
	SHL	reg1, reg2	4	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SHR	reg1, reg2	4	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SXB	reg1	2	1	1	1
	SXH	reg1	2	1	1	1
	ZXB	reg1	2	1	1	1
	ZXH	reg1	2	1	1	1
ビット・サーチ命令	SCH0L	reg2, reg3	4	1	1	1
	SCH0R	reg2, reg3	4	1	1	1
	SCH1L	reg2, reg3	4	1	1	1
	SCH1R	reg2, reg3	4	1	1	1
除算命令	DIV	reg1, reg2, reg3	4	20	20	20
	DIVH	reg1, reg2	2	20	20	20
		reg1, reg2, reg3	4	20	20	20
	DIVHU	reg1, reg2, reg3	4	20	20	20
	DIVU	reg1, reg2, reg3	4	20	20	20
高速除算	DIVQ	reg1, reg2, reg3	4	N+4 注2	N+4 注2	N+4 注2
	DIVQU	reg1, reg2, reg3	4	N+4 注2	N+4 注2	N+4 注2

命令の種類	ニーモニック	オペランド	命令長（バイト数）	実行クロック数		
				issue	repeat	latency
分岐命令	Bcond	disp9（条件不成立時）	2	1	1	1
		disp9（条件成立時）	2	3	3	3
		disp17（条件不成立時）	4	1	1	1
		disp17（条件成立時）	4	3	3	3
	JARL	disp22, reg2	4	3	3	3
		disp32, reg1	6	3	4	3
		[reg1], reg3	4	3	3	3
	JMP	[reg1]	2	3	3	3
		disp32 [reg1]	6	3	4	3
	JR	disp22	4	2	2	2
		disp32	6	2	3	2
ループ命令	LOOP	reg1, disp16（更新後 reg1 ＝ 0 時）	4	2	2	2
		reg1, disp16（更新後 reg1 ≠ 0 時）	4	4	4	4
ビット操作命令	CLR1	bit#3, disp16 [reg1]	4	3 注1	3 注1	3 注1
		reg2, [reg1]	4	3 注1	3 注1	3 注1
	NOT1	bit#3, disp16 [reg1]	4	3 注1	3 注1	3 注1
		reg2, [reg1]	4	3 注1	3 注1	3 注1
	SET1	bit#3, disp16 [reg1]	4	3 注1	3 注1	3 注1
		reg2, [reg1]	4	3 注1	3 注1	3 注1
	TST1	bit#3, disp16 [reg1]	4	3 注1	3 注1	3 注1
		reg2, [reg1]	4	3 注1	3 注1	3 注1

命令の種類	ニーモニック	オペランド	命令長（バイト数）	実行クロック数		
				issue	repeat	latency
特殊命令						
テーブル参照分岐	SWITCH	reg1	2	6	6	6
サブルーチン・コール	CALLT	imm6	2	6	6	6
	CTRET	—	4	3	3	3
システム・コール例外	SYSCALL	vector8	4	6	6	6
ソフトウェア例外	FETRAP	vector4	2	3	3	3
	TRAP	vector5	4	3	3	3
例外復帰	EIRET	—	4	3	3	3
	FERET	—	4	3	3	3
EI レベル割り込み制御	DI	—	4	1	1	1
	EI	—	4	1	1	1
スタック復帰／退避	DISPOSE	imm5, list12	4	n+2 注1 注3	n+2 注1 注3	n+2 注1 注3
		imm5, list12, [reg1]	4	n+2 注1 注3	n+2 注1 注3	n+2 注1 注3
	PREPARE	list12, imm5	4	n+2 注1 注3	n+2 注1 注3	n+2 注1 注3
		list12, imm5, sp	4	n+3 注1 注3	n+3 注1 注3	n+3 注1 注3
		list12, imm5, imm16	6	n+3 注1 注3	n+3 注1 注3	n+3 注1 注3
		list12, imm5, imm16<<16	6	n+3 注1 注3	n+3 注1 注3	n+3 注1 注3
		list12, imm5, imm32	8	n+3 注1 注3	n+3 注1 注3	n+3 注1 注3
	POPSP	rh-rt	4	n+2 注1 注3	n+2 注1 注3	n+2 注1 注3
	PUSHSP	rh-rt	4	n+2 注1 注3	n+2 注1 注3	n+2 注1 注3
	DISPOSE/ PREPARE POPSP/ PUSHSP	No reg in the list	4 or 6 or 8	1	1	1
システム・レジスタ操作	LDSR	reg2, regID, selID	4	1	1	1
	STSR	regID, reg2, selID	4	1	1	2
排他制御	CAXI	[reg1], reg2, reg3	4	4 注1	4 注1	5 注1
	LDL.W	[reg1], reg3	6	1	1	2
	STC.W	reg3, [reg1]	6	1	1	2
停止	HALT	—	4	不定	不定	不定
	SNOOZE	—	4	不定	不定	不定
同期化	SYNCE	—	2	不定	不定	不定
	SYNCI	—	2	不定	不定	不定
	SYNCM	—	2	不定	不定	不定
	SYNCP	—	2	不定	不定	不定
その他	NOP	—	2	1	1	1
	RIE	—	4	3	3	3

注 1.　ウエイト・ステートがない場合。

注 2.　N = int (((被除数の絶対値の有効ビット数) – (除数の絶対値の有効ビット数)) ÷ 2) + 1。
ただし、N < 1 の場合は N = 1、ゼロで割った場合は N = 0 となります。N の範囲は 0-16 です。
注 3.　n は、list で指定されるレジスタの合計数です。
注 4.　ECC 制御が必要な RAM アクセスの場合でその 2 clock 分を含みます。

(2) キャッシュ命令

命令の種類	ニーモニック	オペランド	命令長（バイト数）	実行クロック数		
				issue	repeat	latency
キャッシュ操作命令	CACHE 注 1	cacheop, [reg1]	4	1	1	1
プリフェッチ命令	PREF 注 1	prefop, [reg1]	4	1	1	1

注 1.　NOP 命令として動作します。

(3) 浮動小数点演算命令（ 単精度）

命令の種類	ニーモニック	オペランド	命令長（バイト数）	実行クロック数（インプレサイス）			実行クロック数（プレサイス）		
				issue	repeat	latency	issue	repeat	latency
浮動小数点算術演算	ABSF.S	reg2, reg3	4	1	1	5	6	6	6
	ADDF.S	reg1, reg2, reg3	4	1	1	5	6	6	6
	NEGF.S	reg2, reg3	4	1	1	5	6	6	6
	SUBF.S	reg1, reg2, reg3	4	1	1	5	6	6	6
浮動小数点乗算	MULF.S	reg1, reg2, reg3	4	1	1	5	6	6	6
加減算付き乗算	FMAF.S	reg1, reg2, reg3	4	2	2	6	7	7	7
	FMSF.S	reg1, reg2, reg3	4	2	2	6	7	7	7
	FNMAF.S	reg1, reg2, reg3	4	2	2	6	7	7	7
	FNMSF.S	reg1, reg2, reg3	4	2	2	6	7	7	7
浮動小数点除算	DIVF.S	reg1, reg2, reg3	4	14 注1	14	18	19	19	19
浮動小数点平方根／逆数	RECIPF.S	reg2, reg3	4	4 注1	4	8	9	9	9
	RSQRTF.S	reg2, reg3	4	4 注1	4	8	9	9	9
	SQRTF.S	reg2, reg3	4	14 注1	14	18	19	19	19
浮動小数点形式間の変換／固定小数点形式と浮動小数点形式間の変換	CVTF.HS	reg2, reg3	4	1	1	5	6	6	6
	CVTF.LS	reg2, reg3	4	1	1	5	6	6	6
	CVTF.SH	reg2, reg3	4	1	1	5	6	6	6
	CVTF.SL	reg2, reg3	4	2	2	6	6	6	6
	CVTF.SUL	reg2, reg3	4	2	2	6	6	6	6
	CVTF.SUW	reg2, reg3	4	1	1	5	6	6	6
	CVTF.SW	reg2, reg3	4	1	1	5	6	6	6
	CVTF.ULS	reg2, reg3	4	1	1	5	6	6	6
	CVTF.UWS	reg2, reg3	4	1	1	5	6	6	6
	CVTF.WS	reg2, reg3	4	1	1	5	6	6	6
	CEILF.SL	reg2, reg3	4	2	2	6	6	7	6
	CEILF.SUL	reg2, reg3	4	2	2	6	6	7	6
	CEILF.SUW	reg2, reg3	4	1	1	5	6	6	6
	CEILF.SW	reg2, reg3	4	1	1	5	6	6	6
	FLOORF.SL	reg2, reg3	4	2	2	6	6	7	6
	FLOORF.SUL	reg2, reg3	4	2	2	6	6	7	6
	FLOORF.SUW	reg2, reg3	4	1	1	5	6	6	6
	FLOORF.SW	reg2, reg3	4	1	1	5	6	6	6
	TRNCF.SL	reg2, reg3	4	2	2	6	6	7	6
	ROUNDF.SL	reg2, reg3	4	1	1	4	7	7	7
	ROUNDF.SUL	reg2, reg3	4	1	1	4	7	7	7
	ROUNDF.SUW	reg2, reg3	4	1	1	4	7	7	7
	ROUNDF.SW	reg2, reg3	4	1	1	4	7	7	7
	TRNCF.SUL	reg2, reg3	4	2	2	6	6	7	6
	TRNCF.SUW	reg2, reg3	4	1	1	5	6	6	6
	TRNCF.SW	reg2, reg3	4	1	1	5	6	6	6

命令の種類	ニーモニック	オペランド	命令長（バイト数）	実行クロック数（インプレサイス）			実行クロック数（プレサイス）		
				issue	repeat	latency	issue	repeat	latency
浮動小数点比較	CMPF.S	cond, reg1, reg2, cc	4	1	1	5	6	6	6
条件付き転送	CMOVF.S	cc, reg1, reg2, reg3	4	1	1	5	6	6	6
条件ビット転送	TRFSR	cc	4	1	1	1	1	1	1
浮動小数点最大と最小	MAXF.S	reg1, reg2, reg3	4	1	1	5	6	6	6
	MINF.S	reg1, reg2, reg3	4	1	1	5	6	6	6

注 1.　浮動小数点演算命令以外の後続命令への issue は 1 となります。

備考 1.　実行クロックの凡例

略号	意味
issue	命令実行直後に他の命令を実行する場合
repeat	命令実行直後に同一命令を繰り返す場合
latency	命令実行結果をその命令実行直後の命令で利用する場合

付録 C レジスタ索引

付録 D 命令索引

改訂記録	RH850G3KH　ソフトウェア編

箇所	内容	分類
第5章　メモリ管理		
P108	5.1.7 メモリ保護設定チェック機能　(2) サンプル・コード　修正：be → bnz	(a)
第6章　コプロセッサ		
P134	6.1.11 近傍へのフラッシュ　説明を変更	(b)

備考：表中の「分類」により，改訂内容を次のように区分しています。
（a）：誤記訂正、（b）：仕様（スペック含む）の追加／変更、（c）：説明，注意事項の追加／変更

RH850G3KH ユーザーズマニュアル
ソフトウェア編

発行年月日　2015年04月23日　Rev.0.50
　　2015年08月21日　Rev.1.00
　　2016年07月29日　Rev.1.10
　　2016年12月22日　Rev.1.20

発行　　ルネサス エレクトロニクス株式会社
　　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社 〒135-0061 東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/

Colophon 3.0

RH850G3KH

RENESAS

ルネサス エレクトロニクス株式会社

R01US0165JJ0120