# PM0253
# プログラミング·マニュアル

## STM32F7 シリーズおよび STM32H7 シリーズ Cortex®-M7 プロセッサプログラミングマニュアル

## 概要

このプログラミング·マニュアルには、アプリケーションやシステムレベルのソフトウェアの開発者向けの情報が記載されています。本書では、STM32F7 シリーズおよび STM32H7 シリーズ Cortex®-M7 プロセッサのプログラミング·モデル、命令セット、およびコア·ペリフェラルについて詳しく説明します。

STM32F7 シリーズおよび STM32H7 シリーズ Cortex-M7 プロセッサは、マイクロコントローラ市場向けに設計された高性能な 32 ビットプロセッサです。

Arm® Cortex®-M7 は、Cortex-M プロセッサの中で最高性能を持ったものです。これは 6 段のスーパースカラ・パイプラインを、AXI、AHB、キャッシュ、密結合メモリなどを搭載した柔軟性の高いシステムやメモリ・インタフェースと組み合わせ、高い整数、浮動小数点、および、DSP 性能をもった STM32F7 シリーズ、および STM32H7 シリーズ MPU を提供します。また、複数のメモリ・インタフェースへのロード／ロードおよびロード／ストア命令ペアの二重発行もサポートしています。

Cortex-M7 プロセッサも、これまで同様の使いやすさや、C 言語と相性の良いプログラマのモデルを利用しており、これまでの Cortex-M プロセッサやツールと完全なバイナリ互換です。すべての Cortex-M プロセッサと共に、
Arm Cortex-M の開発エコシステムによる完全なサポートを利用できます。ソフトウェアに互換性があるので、Cortex-M3 や Cortex-M4 プロセッサからの移行も容易に行えます。

# 目次

# 表の一覧

# 図の一覧

# 1 本書について

本書には、アプリケーションおよびシステムレベルのソフトウェアの開発に必要な情報が記載されています。デバッグに関連するコンポーネント、機能、動作についての情報は含まれません。

マイクロコントローラのソフトウェアおよびハードウェアを扱うエンジニアを対象にした資料であり、Arm 製品に対する経験の有無は問いません。

arm

## 1.1 表記規則

本書の表記規則は次のとおりです。

| | |
|---|---|
| *italic* | 重要な注記を強調したり、特別な用語の説明、内部の相互参照や引用を表示したりします。 |
| **bold** | メニュー名などのユーザ·インタフェース要素の強調に太字を使用します。信号名も太字で示します。適宜、記述リスト内の用語にも使用します。 |
| `monospace` | コマンド、ファイル名、プログラム名、ソース·コードなどキーボードから入力可能なテキストを表します。 |
| `monospace` | コマンドやオプションの許容されている略称を表します。コアではフル・コマンドやオプション名の代わりに、下線を引いたテキストを入力することができます。 |
| *`monospace italic`* | 特定の値に置き換えられる引数は等幅フォントのテキストで表されます。 |
| ***`monospace bold`*** | サンプル·コード以外で使用される言語キーワードを示します。 |
| < および > | コードまたはコード·フラグメント内でアセンブラ構文の置き換えが可能な用語を囲みます。例：<br>LDRSB<cond> <Rt>, [<Rn>, #<offset>] |

## 1.2 レジスタに関する略記

レジスタの説明では、次の略記が使用されます。

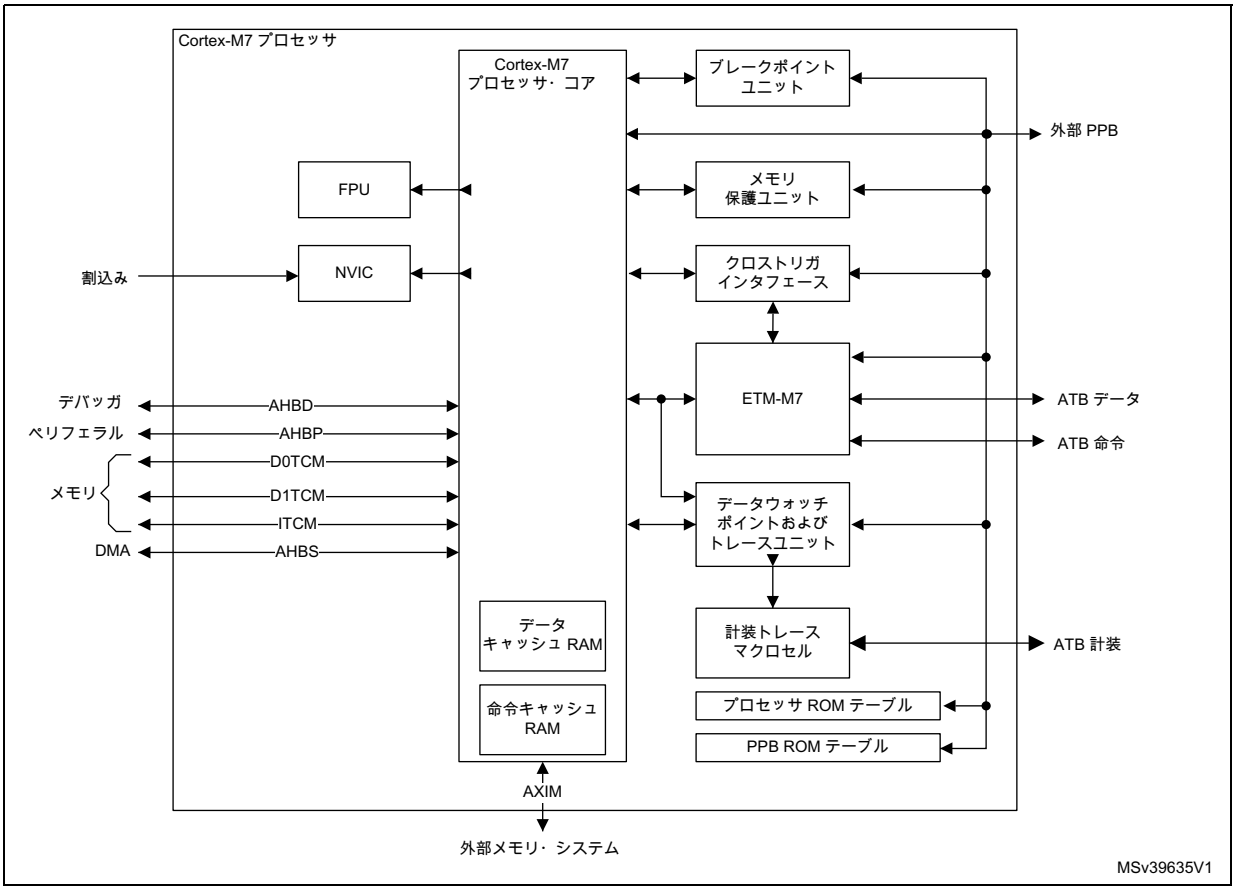
| | |
|---|---|
| 読出し／書込み（rw） | これらのビットは、ソフトウェアによる読出しと書込みができます。 |
| 読出し専用（r） | これらのビットは、ソフトウェアによる読出しのみが可能です。 |
| 書込み専用（w） | このビットは、ソフトウェアによる書込みのみが可能です。<br>ビットを読み出すと、リセット値が返されます。 |
| 読出し／クリア（rc_w） | このビットは、ソフトウェアによって読み出すことができるほか、「任意の値」を書き込むことによってクリアすることもできます。 |
| 読出し/クリア（rc_w1） | このビットは、ソフトウェアによって読み出すことができ、“1”を書き込むことによってクリアできます。<br>“0” を書き込んでも、ビットの値は変化しません。 |
| 読出し/クリア（rc_w0） | このビットは、ソフトウェアによって読み出すことができ、“0”を書き込むことによってクリアできます。<br>“1” を書き込んでも、ビットの値は変化しません。 |
| 反転（t） | このビットは、ソフトウェアによって“1”を書き込むことで反転だけできます。“0”を書き込んでも、ビットの値は変化しません。 |
| 予約済み（Res.） | 予約済みビットであり、リセット値に保持する必要があります。 |

## 1.3 Cortex®-M7 プロセッサおよびコア･ペリフェラルについて

Cortex®-M7 プロセッサは、マイクロコントローラ市場向けに設計された高性能の 32bit プロセッサです。開発者に次のような大きなメリットを提供します。

- 抜群の処理性能と高速な割込み処理の融合。
- 幅広いブレークポイント設定やトレース機能によって強化されたシステム･デバッグ機能。
- 効率的なプロセッサ･コア、システム、およびメモリ。
- 統合された SLEEP モードによる低消費電力。
- プラットフォームに対する、内蔵Memory Protection Unit（メモリ保護ユニット）（MPU）による堅牢なセキュリティ。

**図 1. STM32 Cortex®-M7 の実装プロセッサ**




Cortex®-M7 プロセッサは、6 段パイプラインを備えたハーバード・アーキテクチャの高性能プロセッサ・コアを基盤として構築されているため、要求の厳しい組込みアプリケーションに最適です。この、インオーダ・スーパースカラ・プロセッサは、効率的な命令セットと徹底的に最適化された設計によって並外れた電力効率を実現し、IEEE754 準拠の単精度および倍精度浮動小数点演算、シングルサイクルや SIMD の多彩な乗算、および積和演算、飽和演算、専用ハードウェア除算などを実行する、ハイエンドの処理ハードウェアを搭載しています。

Cortex®-M7 プロセッサでは、コスト重視の機器の設計を容易にするため、システム・コンポーネントを高密度で結合してプロセッサ占有面積を縮小するとともに、割込み処理とシステム・デバッグ機能を大幅に強化しています。Thumb-2 テクノロジに基づいた Thumb® バージョンの命令セットを搭載することで、コード密度を高め、必要なプログラム・メモリ容量を抑えます。Cortex®-M7 の命令セットは、8bit および 16bit マイクロコントローラの高いコード密度を保ちつつ、最新の 32bit アーキテクチャならではの並外れた性能を発揮します。

Cortex®-M7 プロセッサは、設定可能なNVICを綿密に統合して業界トップクラスの割込み性能を提供します。NVIC はNon Maskable Interrupt（ノンマスカブル割込み）（NMI）機能を備え、最大 256 レベルの割込み優先度を提供します。プロセッサ・コアと NVIC を密接に統合することで、interrupt service routines（割込みサービス・ルーチン）（ISR）の実行を高速化し、割込みレイテンシを大幅に短縮しています。これを実現するのが、レジスタのハードウェア・スタッキングと、多重ロードおよび多重ストア動作をサスペンドする機能です。割込みハンドラはアセンブラ・コードに折り返しを必要としないため、ISR のあらゆるコード・オーバーヘッドが取り除かれます。テールチェイン最適化も、ISR の切り換え時のオーバーヘッドを著しく短縮します。

低電力設計を最適化するため、NVIC は、プログラム状態を保持したままデバイス全体の迅速なパワーダウンを可能にする、DEEP SLEEP 機能を含むSLEEP モードを内蔵しています。

自動フォールト検知とハンドリングを内蔵することで、信頼性を向上しています。
Cortex®-M7 プロセッサは、メモリ・アクセスにECC と SECDED を使用し、Memory Build-in Self Test（メモリ内蔵のセルフ・テスト）（MBIST）機能があります。Cortex®-M7 プロセッサは二重冗長なので、ロックステップでも動作が可能です。MCU ベンダが信頼性機能設定を決定するため、これはデバイスやファミリーにより異なることがあります。

命令のスループットを向上するために、Cortex®-M7 プロセッサは、ある特定の命令ペアを同時に実行することができます。これを二重発行と呼びます。

### 1.3.1 システム･レベル･インタフェース

Cortex®-M7 プロセッサは、高速、低レイテンシのメモリ･アクセスを実現するため、AMBA® テクノロジによる複数のインタフェースを備えています。これはアンアライインド･データ･アクセスをサポートします。

Cortex®-M7 プロセッサは、きめ細かくメモリを制御する MPU を備えているため、アプリケーションは複数の特権レベルを活用し、コード、データ、スタックをタスクごとに分離および保護できます。これらの機能を備えることは、車載用途など多くの組込み用途に不可欠な要件となりつつあります。

### 1.3.2 設定可能な統合デバッグ機能

Cortex®-M7 プロセッサは、総合的なハードウェア･デバッグ･ソリューションを搭載しています。これにより、従来の JTAG ポートまたはマイクロコントローラやその他の小型パッケージ･デバイスに最適な 2 ピンのSerial Wire Debug（シリアル･ワイヤ･デバッグ）（SWD）ポートのいずれかを介して、プロセッサやメモリに対するシステム可視性が高まります。MCU ベンダがデバッグ機能設定を決定するため、これはデバイスやファミリーにより異なることがあります。

システム･トレースに関しては、データ･ウォッチポイントやプロファイリング･ユニットに加えて Instrumentation Trace Macrocell（計装トレース･マクロセル）（ITM）が統合されています。これらのトレース機能が生成するシステム･イベントを簡単にコスト効率よくプロファイリングできるように、シリアル･ワイヤ･ビューア（SWV）によって、ソフトウェア生成メッセージ、データ･トレース、プロファイリング情報といったストリーム･データを単一ピンを介してエクスポートできます。

オプションの CoreSight テクノロジー・コンポーネント、Embedded Trace Macrocell（エンベデッド・トレース・マクロセル）™（ETM）は、他に類を見ない命令トレースとデータ・トレース・キャプチャを、従来のトレースユニットよりはるかに小さな領域で実現し、多くの低コスト MCU で完全な命令トレースを初めて可能としました。

ブレークポイント・ユニットは、デバッガが使用することができる、最大 8 つのハードウェア・ブレークポイント・コンパレータを提供します。

### 1.3.3 Cortex®-M7 プロセッサの特徴と利点の概要

- システム･ペリフェラルとの密接な統合による占有面積と開発コストの削減。
- Thumb 命令セットによる 32bit 性能と高いコード密度の両立。
- IEEE754 準拠の単精度および倍精度 Floating-Point Unit（浮動小数点ユニット）（FPU）。
- システム･コンポーネントの電源制御の最適化。
- 統合された SLEEP モードによる省電力化。
- 高速なコード実行によりプロセッサ･クロックの低速化と SLEEP モード期間の増加が可能。
- ハードウェア除算と高速デジタル信号処理指向の積和。
- 信号処理のための飽和演算
- 速度重視のアプリケーション向けの高性能な決定性の割込み処理。
- 安全性が重視されるアプリケーション向け MPU。
- 命令キャッシュおよびデータ・キャッシュ内蔵の Arm Cortex®-M7。
- キャッシュ、DMA ポート付き Tightly-Coupled Memory（密結合メモリ）（TCM）などのメモリ・システム機能と、高性能 AXI 外部メモリ・インタフェース。
- TCM へのシステム・アクセスのための、専用 AHB スレーブ（AHBS）インタフェース。
- 幅広いデバッグおよびトレース機能：
  - シリアル･ワイヤ･デバッグおよびシリアル･ワイヤ･トレースにより、デバッグやトレース、コード・プロファイリングに必要なピン数が削減されます。

### 1.3.4 Cortex®-M7 プロセッサのコア･ペリフェラル

Cortex®-M7 プロセッサのコア･ペリフェラルは次のとおり。

**ネスト化されたベクタ割込みコントローラ**

NVIC は、低レイテンシの割込み処理をサポートする内蔵割込みコントローラです。

**システム制御ブロック**

System Control Block（システム制御ブロック）（SCB）は、プログラマ向けに用意されたプロセッサへのモデル･インタフェースです。システムの実装情報と、システム例外の設定、制御、報告などのシステム制御機能を提供します。

**統合命令およびデータ・キャッシュ**

命令キャッシュおよびデータ・キャッシュは、頻繁にアクセスするデータや命令への高速アクセスを提供し、システム・ベース・メモリを使用した場合の平均性能向上をサポートします。

**システム･タイマ**

システム･タイマの SysTick は、24bit のカウントダウン･タイマです。Real Time Operating System（リアルタイム OS）（RTOS）のティック･タイマまたは単純なカウンタとして使用します。

**メモリ保護ユニット**

Memory Protection Unit（メモリ保護ユニット）（MPU）は、各種メモリ領域に対してメモリ属性を定義することでシステムの信頼性を向上させます。最大 8 つの異なる領域と、必要に応じて事前定義が可能なバックグラウンド領域が提供されます。

**浮動小数点ユニット**

FPU は、32 ビット単精度および 64 ビット倍精度浮動小数点値の、IEEE754 に準拠した動作を提供します。

# 2 Cortex-M7 プロセッサ

## 2.1 プログラマ·モデル

このセクションでは、Cortex®-M7 のプログラマ·モデルについて説明します。個々のコア·レジスタの説明に加えて、ソフトウェアの実行およびスタックに対するプロセッサのモードや特権レベルの情報も提示します。

### 2.1.1 ソフトウェアの実行に対するプロセッサのモードと特権レベル

このプロセッサには、以下のモードがあります。

**スレッド·モード** アプリケーション・ソフトウェアを実行します。プロセッサはリセット状態が解除されるとスレッド·モードに移行します。

**ハンドラ・モード** 例外を処理します。すべての例外処理が終了すると、プロセッサはスレッド·モードに戻ります。

ソフトウェア実行には、次の特権レベルがあります。

**非特権** ソフトウェア：

- MSR および MRS 命令を使ったシステム・レジスタへのアクセスに制限され、マスク割り込みに CPS 命令は使用できません。
- システム·タイマ、NVIC、システム制御ブロックにはアクセスできません。
- メモリやペリフェラルへのアクセスが制限される場合があります。

非特権ソフトウェアは非特権レベルで実行されます。

**特権** ソフトウェアは、すべての命令を使用でき、すべてのリソースにアクセスできます。

特権ソフトウェアは特権レベルで実行されます。

スレッド·モードでは、ソフトウェア実行に対する特権の有無は CONTROL レジスタによって制御します（27 ページのCONTROL レジスタを参照）。ハンドラ・モードでは、ソフトウェア実行は常に特権を持ちます。

特権ソフトウェアのみ、CONTROL レジスタへ書込みを行って、スレッド・モードでのソフトウェア実行に対する特権レベルを変更することが可能です。非特権ソフトウェアは SVC 命令を使って、特権ソフトウェアに制御を移すためのスーパバイザ·コールを実行することができます。

### 2.1.2 スタック

このプロセッサは完全降順スタックを使用します。つまり、スタック・ポインタはメモリ内にスタックされた最後の項目のアドレスを保持しています。プロセッサがスタックに新しい項目をプッシュすると、スタック·ポインタがデクリメントされ、その項目が新しいメモリ位置に書き込まれます。プロセッサには、メイン·スタックとプロセス·スタックの 2 つのスタックが搭載され、それぞれのポインターは独立したレジスタに保持されています（21 ページのスタック·ポインタを参照）。

スレッド·モードの場合、プロセッサがメイン·スタックとプロセス·スタックのどちらを使用するかは、CONTROL レジスタによって制御されます（27 ページのCONTROL レジスタを参照）。ハンドラ·モードでは、プロセッサは常にメイン·スタックを使用します。プロセッサ動作には、次のような選択肢があります。

表 1. プロセッサ·モード、実行特権レベル、スタックの使用オプションの概要

| プロセッサ・モード | 実行に使用 | ソフトウェア実行の特権レベル | 使用するスタック |
|---|---|---|---|
| スレッド | アプリケーション | 特権または非特権(1) | メイン·スタックまたはプロセス·スタック (1) |
| ハンドラ | 例外ハンドラ | 常に特権 | メイン·スタック |

1. 27 ページのCONTROL レジスタを参照してください。

## 2.1.3 コア·レジスタ

プロセッサのコア・レジスタは、次のとおりです。

図 2. プロセッサのコア·レジスタ

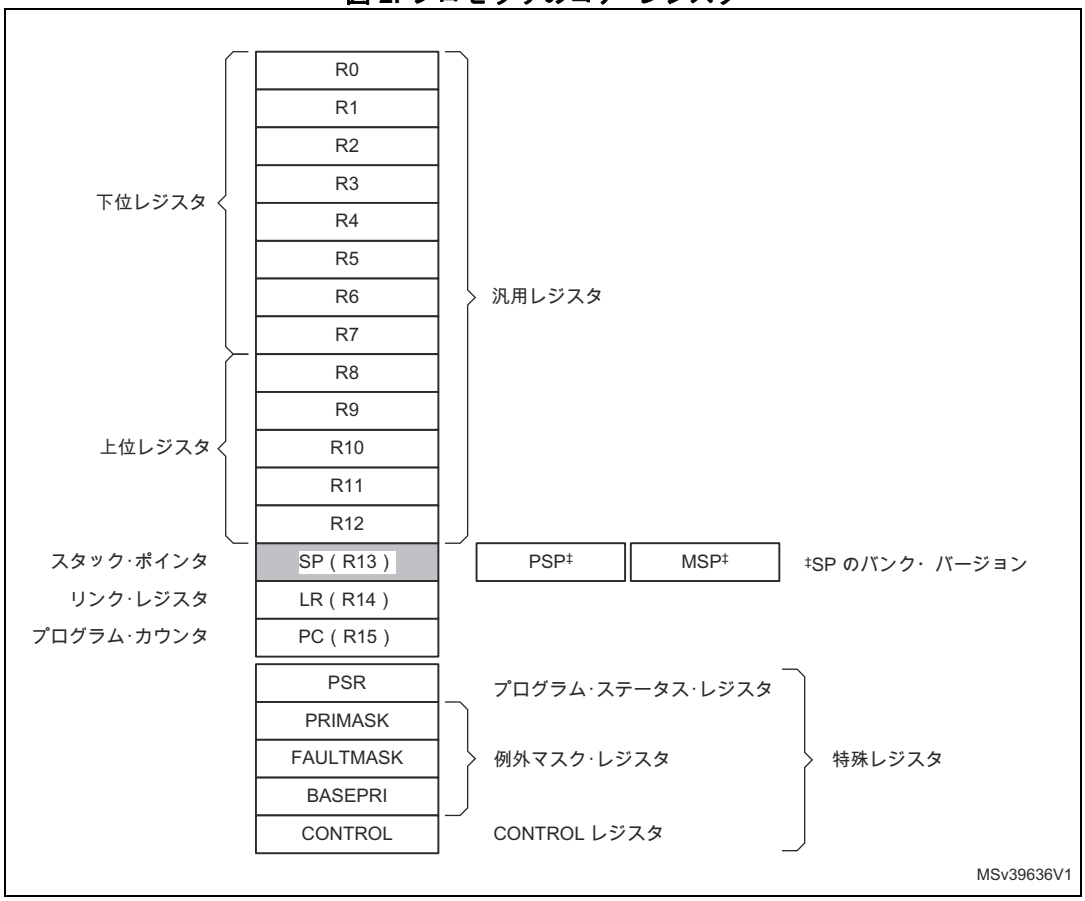



表 2. コア·レジスタ·セットの概要

| レジスタ | 名前 | タイプ(1) | 必要な特権(2) | リセット値 | 説明 |
|---|---|---|---|---|---|
| 汎用レジスタ | R0-R12 | RW | 両方 | 不明 | 21 ページの汎用レジスタを参照してください。 |
| スタック·ポインタ | MSP | RW | 両方 | 説明を参照 | 21 ページのスタック·ポインタを参照してください。 |
| スタック·ポインタ | PSP | RW | 両方 | 不明 | 21 ページのスタック·ポインタ |

表 2. コア·レジスタ·セットの概要 （続き）

| レジスタ | 名前 | タイプ(1) | 必要な特権(2) | リセット値 | 説明 |
|---|---|---|---|---|---|
| リンク·レジスタ | LR | RW | 両方 | 0xFFFFFFFF | 21 ページのリンク·レジスタ |
| プログラム·カウンタ | PC | RW | 両方 | 説明を参照 | 21 ページのプログラム·カウンタ |
| プログラム·ステータス·レジスタ | PSR | RW | 両方 | 0x01000000(3) | 22 ページのプログラム·ステータス·レジスタ |
| アプリケーション·プログラム·ステータス·レジスタ | APSR | RW | 両方 | 不明 | 23 ページのアプリケーション·プログラム·ステータス·レジスタ |
| 割込みプログラム·ステータス·レジスタ | IPSR | RO | 特権 | 0x00000000 | 23 ページの割込みプログラム·ステータス·レジスタ |
| 例外プログラム·ステータス·レジスタ | EPSR | RO | 特権 | 0x01000000 (3) | 24 ページの例外プログラム·ステータス·レジスタ |
| 優先度マスク·レジスタ | PRIMASK | RW | 特権 | 0x00000000 | 25 ページの優先度マスク·レジスタ |
| フォールト·マスク·レジスタ | FAULTMASK | RW | 特権 | 0x00000000 | 25 ページのフォールト·マスク·レジスタ |
| ベース優先度マスク·レジスタ | BASEPRIS | RW | 特権 | 0x00000000 | 25 ページの優先度マスク·レジスタ |
| 制御レジスタ | CONTROL | RW | 特権 | 0x00000000 | 27 ページのCONTROL レジスタ |

1. スレッド·モードとハンドラ·モードでのプログラム実行時のアクセス·タイプを示しています。デバッグ時のアクセス·タイプは異なる場合があります。
2. 両方というエントリは、特権と非特権の両方のソフトウェアがそのレジスタにアクセスできることを示しています。
3. MRS 命令を実行するとき、EPSR はゼロとして読み込まれます。

### 汎用レジスタ

R0 ~ R12 は、データ操作のための 32bit 汎用レジスタです。

### スタック·ポインタ

スタック·ポインタ（SP）はレジスタ R13 です。スレッド·モードでは、CONTROL レジスタのビット [1] によって、以下のどちらのスタック·ポインタを使用するかを示します。

- 0=Main Stack Pointer（メイン·スタック·ポインタ）（MSP）これがリセット値です。
- 1=Process Stack Pointer（プロセス·スタック·ポインタ）（PSP）

リセット時、プロセッサは MSP にアドレス 0x00000000 の値をロードします。

### リンク·レジスタ

リンク·レジスタ（LR）はレジスタ R14 です。サブルーチン、関数呼び出し、および例外の復帰情報を格納します。リセット時、プロセッサは LR に値 0xFFFFFFFF をセットします。

### プログラム·カウンタ

プログラム·カウンタ（PC）はレジスタ R15 です。現在のプログラム·アドレスを格納します。リセット時、プロセッサは PC にリセット·ベクタ（アドレス 0x00000004）の値をロードします。この値のビット [0] は、リセット時に EPSR の T ビットにロードされ、1 である必要があります。

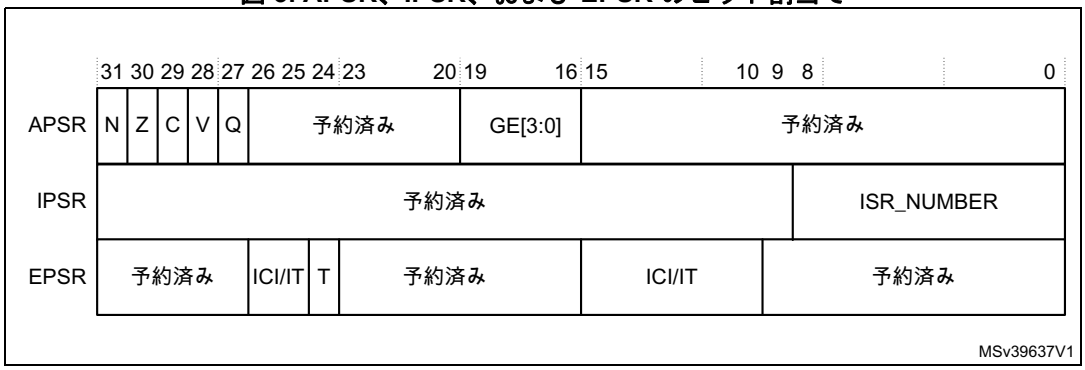
### プログラム･ステータス･レジスタ

Program Status register（プログラム･ステータス･レジスタ）（PSR）は、以下のレジスタを組み合わせたものです。

- Application Program Status register（アプリケーション･プログラム･ステータス･レジスタ）（APSR）
- Interrupt Program Status register（割込みプログラム･ステータス･レジスタ）（IPSR）。
- Execution Program Status register（実行プログラム･ステータス･レジスタ）（EPSR）。

これらのレジスタは、32bit PSR 内の互いに重ならないビットフィールドです。ビット割当てを次に示します。

**図 3. APSR、IPSR、および EPSR のビット割当て**




これらのレジスタに個別に、あるいは任意に組み合わせてアクセスするには、MSR または MRS 命令の引数としてこれらのレジスタ名を指定します。例：

- MRS 命令で PSR を使用して、3 つすべてのレジスタを読み出します。
- MSR 命令で APSR_nzcvq を使用して、APSR の N、Z、C、V、Q の各ビットに書き込みます。

PSR の組合せと属性は次のとおりです。

**表 3. PSR レジスタの組合せ**

| レジスタ | タイプ | 組合せ |
|---|---|---|
| PSR | RW(1) .(2) | APSR、EPSR、および IPSR |
| IEPSR | RO | EPSR および IPSR |
| IAPSR | RW (1) | APSR および IPSR |
| EAPSR | RW (2) | APSR および EPSR |

1. プロセッサは IPSR のビットへの書込みを無視します。
2. EPSR のビットを読み出すとゼロが返され、プロセッサはこれらのビットへの書込みを無視します。

プログラム･ステータス･レジスタへのアクセス方法の詳細については、175 ページのMRS および 176 ページのMSR の命令の説明を参照してください。

### アプリケーション・プログラム・ステータス・レジスタ

APSR には、前回の命令実行によって設定された現在の条件フラグの状態が格納されます。属性については、23 ページの表 4 のレジスタの概要を参照してください。ビット割当てを次に示します。

**表 4. APSR のビット割当て**

| ビット | 名前 | 説明 |
|---|---|---|
| [31] | N | 負のフラグ |
| [30] | Z | ゼロ・フラグ |
| [29] | C | キャリーまたはボロー・フラグ |
| [28] | V | オーバーフロー・フラグ |
| [27] | Q | DSP オーバーフローおよびサチュレーション・フラグ |
| [26:20] | - | 予約済みです。 |
| [19:16] | GE[3:0] | 以上フラグ。詳細については、107 ページのSEL を参照してください。 |
| [15:0] | - | 予約済みです。 |

### 割込みプログラム・ステータス・レジスタ

IPSR には、現在の割込みサービス・ルーチン（ISR）の例外タイプの番号が格納されます。属性については、23 ページの表 5 のレジスタの概要を参照してください。ビット割当てを次に示します。

**表 5. IPSR のビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:9] | - | 予約済みです。 |
| [8:0] | ISR_NUMBER | これは現在の例外の番号です。<br>0 = スレッド・モード<br>1 = 予約済み<br>2 = NMI<br>3 = ハード・フォールト<br>4 = メモリ管理<br>5 = バス・フォールト<br>6 = 用法フォールト<br>7-10 = 予約済み<br>11 = SVCall<br>12 = デバッグ用に予約済み<br>13 = 予約済み<br>14 = PendSV<br>15 = SysTick<br>16 = IRQ0<br>。<br>。<br>256 = IRQ239<br>詳細は、39 ページの例外のタイプを参照してください。 |

### 例外プログラム・ステータス・レジスタ

EPSR には、Thumb 状態ビットおよび次のいずれかの実行状態ビットが格納されます。

- If-Then（IT）命令。
- 割り込まれた多重ロードまたは多重ストア命令の中断可能で中断後から継続可能な命令（ICI）フィールド。

EPSR の属性については、24 ページの表 6 のレジスタの概要を参照してください。ビット割当てを次に示します。

**表 6. EPSR のビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:27] | - | 予約済み。 |
| [26:25], [15:10] | ICI | 中断可能で中断後から継続可能な命令ビット（24 ページの中断可能で中断後から継続可能な命令を参照） |
| [26:25], [15:10] | IT | IT 命令の実行状態ビットを示します（145 ページのIT を参照）。 |
| [24] | T | Thumb 状態ビット（Thumb 状態を参照） |
| [23:16] | - | 予約済み。 |
| [9:0] | - | 予約済み。 |

アプリケーション・ソフトウェアから MSR 命令を使用して EPSR を直接読み出そうとすると、常にゼロが返されます。アプリケーション・ソフトウェアでの MSR 命令による EPSR への書込み動作は無視されます。

### 中断可能で中断後から継続可能な命令

LDM、STM、PUSH、POP、VLDM、VSTM、VPUSH、または VPOP 命令の実行中に割込みが発生した場合、プロセッサは次のように処理します。

- 多重ロードまたは多重ストア操作を一時的に停止します。
- 次のレジスタ・オペランドを多重操作で EPSR のビット [15:12] に格納します。

割込み処理の後、プロセッサは次のように動作します。

- ビット [15:12] が示すレジスタに復帰します。
- 多重ロードまたは多重ストア命令の実行を再開します。

EPSR に ICI の実行状態が保持されている場合、ビット [26:25,11:10] はゼロになります。

### If-Then ブロック

If-Then ブロックには、IT 命令に続いて最大 4 つの命令が含まれます。ブロック内の各命令は条件付きです。これらの命令に対する条件はすべて同じか、一部の条件を他の部分と反転させることができます。詳細については、145 ページのIT を参照してください。

### Thumb 状態

Cortex®-M7 プロセッサは、Thumb 状態での命令の実行のみをサポートしています。以下によって T ビットを 0 にクリアできます。

- BLX、BX、および POP{PC} 命令。
- 例外から復帰したときのスタックされた xPSR 値からの復元。
- 例外の開始時またはリセット時のベクタ値のビット [0]。

T ビットが 0 の状態で命令を実行するとフォールトまたはロックアップが発生します。詳細については、49 ページのロックアップを参照してください。

### 例外マスク・レジスタ

例外マスク・レジスタは、プロセッサによる例外処理を無効にします。タイミング重視のタスクに影響を与える可能性がある例外を無効にします。

例外マスク・レジスタにアクセスするには MSR および MRS 命令を使用するか、CPS 命令を使用して、PRIMASK または FAULTMASK の値を変更します。詳細は、175 ページのMRS、176 ページのMSR、および 173 ページのCPS を参照してください。

### 優先度マスク・レジスタ

PRIMASK レジスタは、設定可能な優先度を持つすべての例外のアクティブ化を禁止します。属性については、表 7 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 4. PRIMASK のビット割当て：**

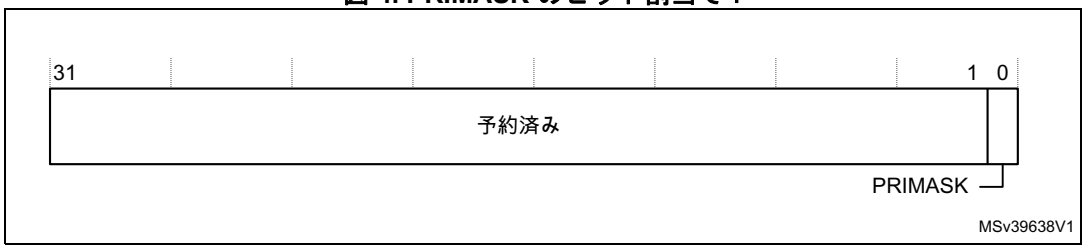



**表 7. PRIMASK レジスタのビット割り当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:1] | - | 予約済み。 |
| [0] | PRIMASK | 優先化可能な割込みマスク：<br>0 = 影響なし。<br>1 = 設定可能な優先度を持つすべての例外のアクティブ化を禁止します。 |

### フォールト・マスク・レジスタ

FAULTMASK レジスタはNon Maskable Interrupt（ノンマスカブル割込み）（NMI）を除くすべての例外のアクティブ化を禁止します。属性については、26 ページの表 8 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 5. FAULTMASK のビット割当て**

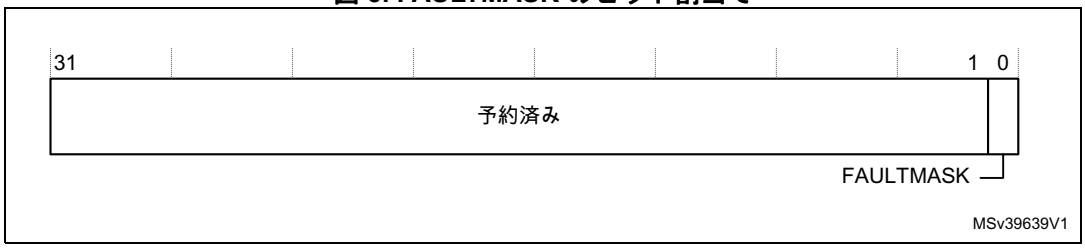


表 8. FAULTMASK レジスタのビット割り当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:1] | - | 予約済み。 |
| [0] | FAULTMASK | 優先化可能な割込みマスク：<br>0 = 影響なし。<br>1 = NMIを除くすべての例外のアクティブ化を禁止します。 |

プロセッサは、NMI ハンドラを除くすべての例外ハンドラの終了時に、FAULTMASK ビットを 0 にクリアします。

## ベース優先度マスク・レジスタ

BASEPRI レジスタは、例外処理に対する最低優先度を定義します。BASEPRI をゼロ以外の値にセットすると、優先度レベルが BASEPRI 値以下のすべての例外のアクティブ化を禁止します。属性については、26 ページの表 9 のレジスタ概要を参照してください。ビット割当てを次に示します。

図 6. BASEPRI のビット割当て

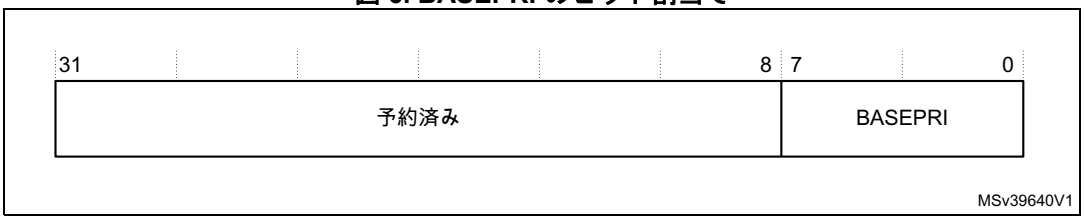



表 9. BASEPRI レジスタのビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:8] | - | 予約済み。 |
| [7:0] | BASEPRI(1) | 優先度マスク・ビット：<br>0x00 影響はありません。<br>ゼロ以外：例外処理のベース優先度を定義します。<br>プロセッサは、BASEPRI の値以上の優先度の値を持つすべての例外を処理しません。 |

1. このフィールドは、割込み優先度レジスタの優先度フィールドと同じです。デバイスはこのフィールドのビット [7:M] のみ実装し、ビット [M-1:0] は 0 として読み出され、書込みは無視されます。詳細については、23 ページの割込みプログラム・ステータス・レジスタ を参照してください。優先度フィールドの値が大きいほど、例外の優先度は低いことに注意してください。

### CONTROL レジスタ

CONTROL レジスタは、使用されるスタック、およびプロセッサがスレッド･モードにある場合のソフトウェア実行の特権レベルを管理し、FPU状態がアクティブかどうかを示します。属性については、27 ページの表 10 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 7. 制御ビット割り当て**

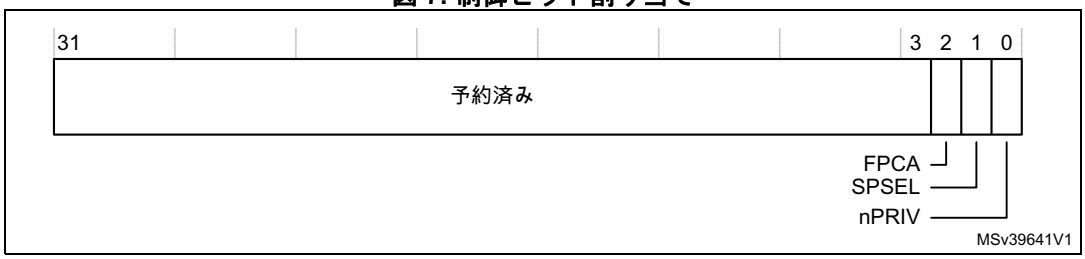



**表 10. コントロール レジスタのビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:3] | - | 予約済み。 |
| [2] | FPCA | 浮動小数点コンテキストが現在アクティブであるかどうかを示します。<br>0：浮動小数点コンテキストはアクティブではありません。<br>1：浮動小数点コンテキストがアクティブです。<br>例外を処理する際に、浮動小数点の状態を維持すべきかどうかを、このビットに基づいて判断します。 |
| [1] | SPSEL | 現在アクティブなスタック・ポインタを定義します。<br>0 = MSP が現在のスタック･ポインタです。<br>1 = PSP が現在のスタック･ポインタです。<br>ハンドラ･モードでは、このビットはゼロとして読み出され、書込みは無視します。Cortex®-M7 プロセッサは、例外からの復帰時に、このビットを自動的に更新します。 |
| [0] | nPRIV | スレッド･モードの特権レベルを定義します：<br>0 = 特権。<br>1 = 非特権。 |

ハンドラ･モードでは常に MSP が使用されます。このため、ハンドラ･モードで動作中のプロセッサは、CONTROL レジスタのアクティブなスタック･ポインタ･ビットへの明示的な書込みを無視します。例外開始および復帰メカニズムが、EXC_RETURN 値に基づいて CONTROL レジスタを自動的に更新します。46 ページの表 20 を参照してください。

OS 環境では、スレッド･モードで動作するスレッドはプロセス･スタックを使用し、カーネルおよび例外ハンドラはメイン･スタックを使用することを Arm 社では推奨します。

スレッド･モードではデフォルトで MSP が使用されます。スレッド･モードで使用していたスタック･ポインタを PSP に切り替えるには、以下のいずれかを実行します。

- MSR 命令を使用して、現在アクティブなスタック･ポインタ･ビット、CONTROL.SPSELbit、を 1 にセットします（176 ページのMSR を参照）。
- 適切な EXC_RETURN 値により例外からスレッド･モードへ復帰します（46 ページの表 20 を参照）。

スタック･ポインタを変更する場合、ソフトウェアは MSR 命令の直後に ISB 命令を使用する必要があります。これによって、ISB 命令の後の命令が確実に新しいスタック･ポインタを使用して実行できます。175 ページのISB を参照してください。

### 2.1.4 例外と割込み

Cortex®-M7 プロセッサは、割込みとシステム例外をサポートしています。プロセッサとNVICがすべての例外の優先度を決定し、処理します。例外は、ソフトウェア制御の通常のフローを変化させます。プロセッサは、リセットを除くすべての例外の処理にハンドラ·モードを使用します。詳細は、44 ページの例外の開始および 46 ページの例外からの復帰を参照してください。

NVIC レジスタは割込み処理を制御します。詳細については、181 ページのネスト化されたベクタ割込みコントローラを参照してください。

### 2.1.5 データ型

プロセッサは、以下のとおりです。

- 以下のデータ型をサポートします。
  - 32 ビットワード
  - 16 bit ハーフワード
  - 8 bit バイト
  - 32 ビット単精度浮動小数点数
  - 64 ビット倍精度浮動小数点数
- すべてのデータ・メモリ·アクセスをリトルエンディアンとして管理します。詳細については、33 ページのメモリの領域、タイプ、および属性を参照してください。

### 2.1.6 Cortex Microcontroller Software Interface Standard（Cortex マイクロコントローラ･ソフトウェア･インタフェース規格）（CMSIS）

Cortex Microcontroller Software Interface Standard（Cortex マイクロコントローラ･ソフトウェア･インタフェース規格）（CMSIS）では、Cortex®-M7 マイクロコントローラ･システムに対して、以下を定義しています。

- 以下を実行するための共通の方法
  - ペリフェラル･レジスタへのアクセス
  - 例外ベクタの定義
- 以下の名称
  - コア･ペリフェラルのレジスタ
  - コア例外ベクタ
- RTOS カーネル用のデバイスに依存しないインタフェース（デバッグ･チャネルを含む）。

CMSIS には、Cortex®-M7 プロセッサのコア･ペリフェラルのアドレス定義およびデータ構造が含まれます。

CMSIS では、テンプレート･コードの再利用や、さまざまなミドルウェア･ベンダが提供する CMSIS 準拠のソフトウェア･コンポーネントの組合せを可能にすることで、ソフトウェア開発を簡素化しています。ソフトウェア･ベンダは CMSIS を拡張して、ペリフェラルの定義やそれらのペリフェラルのアクセス機能を含めることができます。

本書は、CMSIS によって定義されたレジスタ名を記載し、プロセッサ･コアおよびコア･ペリフェラルに対応する CMSIS 関数についても簡単に説明しています。

本書で使用するレジスタの短縮名は CMSIS によって定義されたものです。これらの短縮名は、他のドキュメントで使用されているアーキテクチャ上の短縮名と異なる場合があります。

以降のセクションでは、CMSIS の詳細について説明します。

- 51 ページの電源管理に関するプログラミングのヒントを参照してください。
- 61 ページのCMSIS 関数を参照してください。
- 213 ページのSysTick 設計のヒントとコツ
- 182 ページのCMSIS を使用した Cortex®-M7 NVIC レジスタへのアクセスを参照してください。
- 188 ページのNVIC のプログラミングのヒントを参照してください。
- 238 ページのCMSIS を使用した Cortex®-M7 キャッシュのメンテナンス操作

## 2.2 Cortex®-M7 の構成

表 11、表 12 そして 表 13 は STM32F7 シリーズ Cortex-M7 の構成を表示します。

**表 11. STM32F746xx/STM32F756xx Cortex®-M7 の構成**

| 機能 | STM32F746xx/STM32F756xx |
|---|---|
| 浮動小数点ユニット | 単精度浮動小数点ユニット |
| MPU | 8 領域 |
| 命令 TCM のサイズ | Flash TCM：1 MB<br>RAM ITCM：16 KB |
| データ TCM のサイズ | 64 KB |
| 命令キャッシュのサイズ | 4 KB |
| データ・キャッシュのサイズ | 4 KB |
| キャッシュ ECC | 実装なし |
| 割込み優先順位レベル | 16 の優先順位レベル |
| IRQ 数 | 98 |
| WIC、CTI | 実装なし |
| デバッグ | JTAG および、シリアルワイヤデバッグポート<br>8 つのブレークポイントと 4 つのウォッチポイント |
| ITM サポート | データトレース（DWT）、および計装トレース（ITM） |
| ETM サポート | 命令トレースインタフェース |

**表 12. STM32F76xxx/STM32F77xxx Cortex®-M7 の構成**

| 機能 | STM32F76xxx/STM32F77xxx |
|---|---|
| 浮動小数点ユニット | 倍精度および単精度浮動小数点ユニット |
| MPU | 8 領域 |
| 命令 TCM のサイズ | Flash TCM：2 MB<br>RAM ITCM：16 KB |
| データ TCM のサイズ | 128 KB |
| 命令キャッシュのサイズ | 16 KB |
| データ・キャッシュのサイズ | 16 KB |
| キャッシュ ECC | 実装なし |
| 割込み優先順位レベル | 16 の優先順位レベル |
| IRQ 数 | 110 |
| WIC、CTI | 実装なし |
| デバッグ | JTAG および、シリアルワイヤデバッグポート<br>8 つのブレークポイントと 4 つのウォッチポイント |
| ITM サポート | データトレース（DWT）、および計装トレース（ITM） |
| ETM サポート | 命令トレースインタフェース |

表 13. STM32F72xxx/STM32F73xxx Cortex®-M7 の構成

| 機能 | STM32F72xxx/STM32F73xxx |
|---|---|
| 浮動小数点ユニット | 単精度浮動小数点ユニット |
| MPU | 8 領域 |
| 命令 TCM のサイズ | Flash TCM：512 KB<br>RAM ITCM：16 KB |
| データ TCM のサイズ | 64 KB |
| 命令キャッシュのサイズ | 8 KB |
| データ・キャッシュのサイズ | 8 KB |
| キャッシュ ECC | 実装なし |
| 割込み優先順位レベル | 16 の優先順位レベル |
| IRQ 数 | 104 |
| WIC、CTI | 実装なし |
| デバッグ | JTAG および、シリアルワイヤデバッグポート<br>8 つのブレークポイントと 4 つのウォッチポイント |
| ITM サポート | データトレース（DWT）、および計装トレース（ITM） |
| ETM サポート | 命令トレースインタフェース |

表 14 に、STM32H7 シリーズ Cortex-M7 の構成を示します。

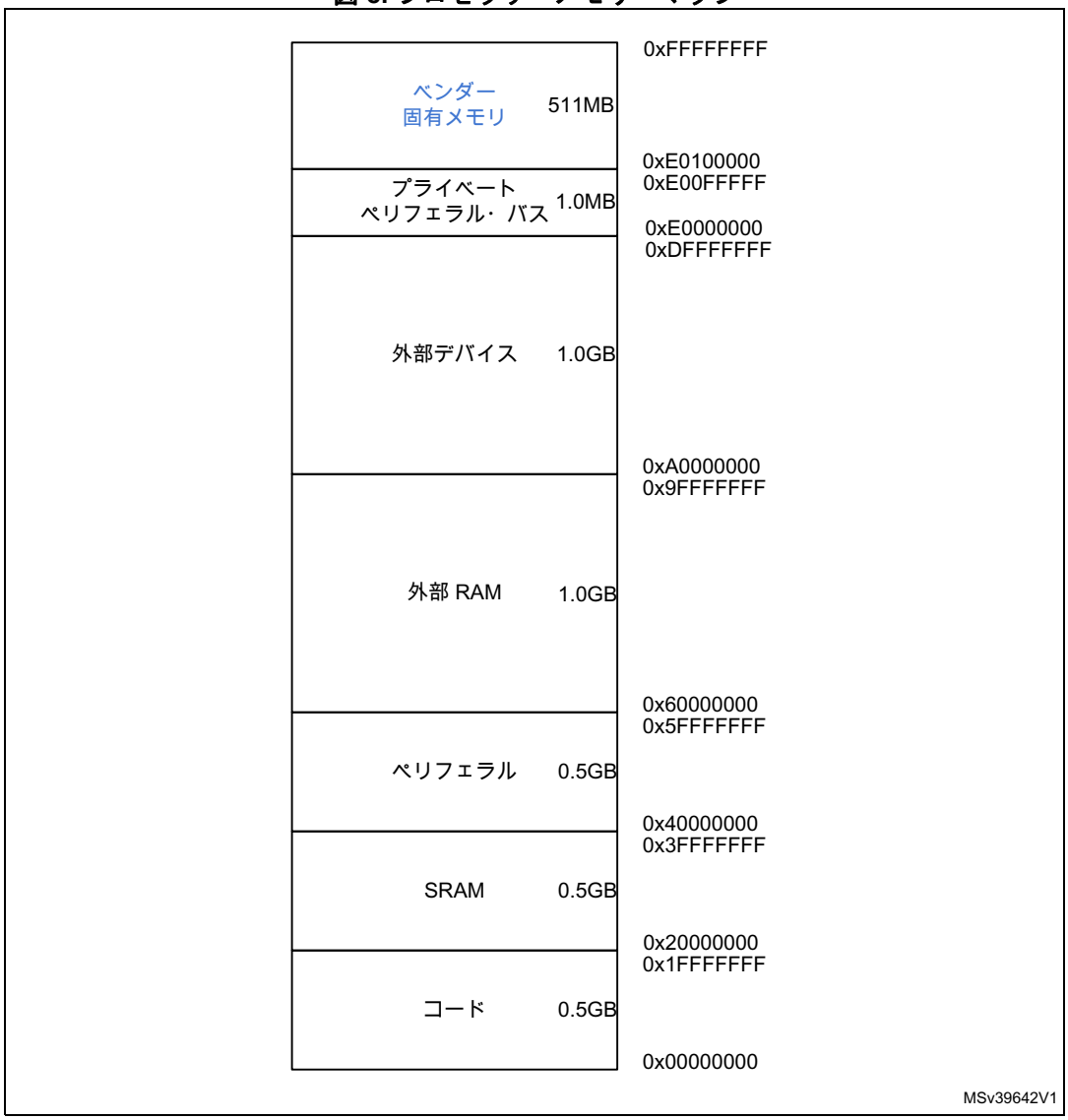
表 14. STM32H7 シリーズ Cortex®-M7の構成

| 機能 | STM32H7 シリーズ |
|---|---|
| 浮動小数点ユニット | 倍精度および単精度浮動小数点ユニット |
| MPU | 16 領域 |
| 命令 TCM のサイズ | RAM ITCM：64 KB |
| データ TCM のサイズ | 128 KB |
| 命令キャッシュのサイズ | 16 KB |
| データ・キャッシュのサイズ | 16 KB |
| キャッシュ ECC | 実装されます。 |
| 割込み優先順位レベル | 16 の優先順位レベル |
| IRQ 数 | 149 |
| WIC、CTI | WIC 未実装、CTI 実装 |
| デバッグ | JTAG および、シリアルワイヤデバッグポート<br>8 つのブレークポイントと 4 つのウォッチポイント |
| ITM サポート | データトレース（DWT）、および計装トレース（ITM） |
| ETM サポート | 命令トレースインタフェース |

## 2.3 メモリ·モデル

このセクションでは、プロセッサのメモリ·マップおよびメモリ·アクセスの動作について説明します。プロセッサには、最大 4 GB のアドレス指定可能なメモリを提供する固定されたデフォルトのメモリ·マップがあります。メモリ・マップは次のとおりです。

**図 8. プロセッサ・メモリ・マップ**




プロセッサでは、コア·ペリフェラルのレジスタ向けに、プライベート·ペリフェラル·バス（PPB）のアドレス範囲の領域を確保しています（180 ページのCortex-M7 ペリフェラルについてを参照）。

### 2.3.1 メモリの領域、タイプ、および属性

メモリ·マップと MPU のプログラミングにより、メモリ·マップは複数の領域に分割されます。各領域にはメモリ·タイプが定義され、一部の領域ではその追加のメモリ属性も設定されています。メモリ·タイプとメモリ属性が、その領域へのアクセス動作を決定します。

メモリ·タイプに次のようなものがあります。

**ノーマル** プロセッサは、効率を高めるためトランザクションの順番を変えたり、投機的読出しを実行することができます。

**デバイス（Device）および Strongly-ordered** プロセッサは、デバイス·メモリへの他のトランザクションや Strongly-ordered メモリへのトランザクションとの相対的なトランザクション順序を保持します。

デバイス·メモリと Strongly-ordered メモリに対して順序付けの要件が異なるため、メモリ·システムはデバイス·メモリへの書込みをバッファできますが、Strongly-ordered メモリへの書込みはバッファできません。

追加のメモリ属性として、次のようなものがあります。

**共有可能（shareable）** 共有可能なメモリ領域の場合、メモリ・システムは、たとえば、DMA コントローラを備えたプロセッサなど、複数のバス・マスタを備えるシステム内のバス・マスタ間のデータ同期を提供します。

Strongly-ordered メモリは常に共有可能です。

複数のバス・マスタが共有不可メモリ領域にアクセスできる場合、ソフトウェアでバス・マスタ間のデータのコヒーレンシを保証する必要があります。

**実行不可（XN）** プロセッサが命令へのアクセスを禁止することを意味します。メモリの XN 領域からフェッチされた命令を実行すると、ハード・フォルト例外が生成されます。

### 2.3.2 メモリ·システムでのメモリ·アクセスの順序付け

明示的なメモリ·アクセス命令によって発生するほとんどのメモリ·アクセスについて、メモリ·システムは、アクセスが完了する順序がプログラムにおける命令の順序と一致することを保証しません。順序を変更しても、命令シーケンスの動作には影響しないことが条件です。通常は、プログラムが正しく実行されるために、2 つのメモリ·アクセスがプログラム順に完了することが必要な場合、ソフトウェアで、これらのメモリ·アクセス命令の間にメモリ·バリア命令を挿入する必要があります（2.3.4： 36ページのソフトウェアによるメモリ·アクセスの順序付けを参照）。

ただし、デバイス·メモリおよび Strongly-ordered メモリへのアクセスの順序付けの一部については、メモリ·システムによって保証されます。2 つのメモリ·アクセス命令 A1 と A2 について、プログラム順では A1 が A2 より前に出現する場合、これら 2 つの命令によって発生するメモリ·アクセスの順序は次のようになります。

**表 15. メモリ·アクセスの順序(1)**

| A1 | A2 | | | |
|---|---|---|---|---|
| | ノーマル·アクセス | デバイス·アクセス | | Strongly-ordered アクセス |
| | | 共有不可 | 共有可能 | |
| ノーマル·アクセス | - | - | - | - |
| デバイス·アクセス、共有不可 | - | < | - | < |

表 15. メモリ·アクセスの順序(1) （続き）

| A1 | A2 | | | |
|---|---|---|---|---|
| | ノーマル·アクセス | デバイス·アクセス | | Strongly-ordered アクセス |
| | | 共有不可 | 共有可能 | |
| デバイス·アクセス、共有可能 | - | - | < | < |
| Strongly-ordered アクセス | - | < | < | < |

1. - はメモリ·システムがアクセスの順序を保証していないことを意味します。
< は、アクセスがプログラム順に従うこと、つまり A1 は常に A2 より前に出現することを意味します。

## 2.3.3 メモリ·アクセスの動作

メモリ·マップの各領域へのアクセスは、次のように動作します。

表 16. メモリ·アクセスの動作(1)

| アドレス範囲 | メモリ領域 | メモリタイプ | XN | 説明 |
|---|---|---|---|---|
| 0x00000000 - 0x1FFFFFFF | エンコード | ノーマル | - | プログラム·コードの実行可能領域。ここにデータを書き込むこともできます。<br>命令フェッチとデータアクセスは、ITCM または AXIM インタフェースを介して実行されます。 |
| 0x20000000 - 0x3FFFFFFF | SRAM | ノーマル | - | データの実行可能領域。ここにコードを入力することもできます。<br>命令フェッチとデータアクセスは、DTCM または AXIM インタフェースを介して実行されます。 |
| 0x40000000 - 0x5FFFFFFF | ペリフェラル | デバイス | XN | 外部デバイス·メモリ<br>データアクセスは、AHBP または AXIM インタフェースを介して実行されます。 |
| 0x60000000 - 0x9FFFFFFF | 外部 RAM | ノーマル | - | データの実行可能領域。<br>命令フェッチとデータアクセスは、AXIM インタフェースを介して実行されます。 |
| 0xA0000000 - 0xDFFFFFFF | 外部デバイス | デバイス | XN | 外部デバイス·メモリ<br>命令フェッチとデータアクセスは、AXIM インタフェースを介して実行されます。 |
| 0xE0000000 - 0xE00FFFFF | 専用ペリフェラル·バス | Strongly-ordered | XN | この領域には、NVIC、システム·タイマ、システム制御ブロックが含まれます。<br>この領域ではワードアクセスのみを使用できます。 |
| 0xE0100000 - 0xFFFFFFFF | ベンダ固有デバイス | デバイス | XN | この領域へのアクセスは、ベンダー固有のペリフェラルです。 |

1. 詳細については、33 ページのメモリの領域、タイプ、および属性 を参照してください。

コード、SRAM、外部RAMの領域にプログラムを保持できます。

MPU は、このセクションに示したデフォルトのメモリ·アクセス動作を上書きできます。詳細については、218 ページのメモリ保護ユニットを参照してください。

### キャッシュおよび共有メモリに対する追加のメモリアクセスの制約

システムにキャッシュや共有メモリが内蔵されている場合、一部のメモリ領域には追加のアクセスの制約があり、表 17 に示すように、一部の領域は再分割されます。

**表 17. メモリ領域の共有可能性とキャッシュのポリシー**

| アドレス範囲 | メモリ領域 | メモリタイプ(1) | 共有可能性 (1) | キャッシュポリシー(2) |
|---|---|---|---|---|
| 0x00000000 - 0x1FFFFFFF | エンコード | ノーマル | 共有不可 | WT |
| 0x20000000 - 0x3FFFFFFF | SRAM | ノーマル | 共有不可 | WBWA |
| 0x40000000 - 0x5FFFFFFF | ペリフェラル | デバイス | 共有不可 | - |
| 0x60000000 - 0x7FFFFFFF | 外部 RAM | ノーマル | 共有不可 | - |
| 0x80000000 - 0x9FFFFFFF | | | | WT |
| 0xA0000000 - 0xBFFFFFFF | 外部デバイス | デバイス | 共有可能 | - |
| 0xC0000000 - 0xDFFFFFFF | | | 共有不可 | |
| 0xE0000000 - 0xE00FFFFF | 専用ペリフェラル·バス | Strongly-ordered | 共有可能 | - |
| 0xE0100000 - 0xFFFFFFFF | ベンダ固有デバイス | デバイス | 共有不可 | - |

1. 詳細は、セクション 2.3.1 : 33 ページのメモリの領域、タイプ、および属性を参照してください。
2. WT = ライトスルー、書込み割当てなしWBWA = ライトバック、書込み割当てあり

### 命令プリフェッチと分岐予測

Cortex®-M7 プロセッサには、次の機能があります。

- 実行前の命令のプリフェッチ。
- 分岐先アドレスからの投機的プリフェッチ

### 2.3.4 ソフトウェアによるメモリ･アクセスの順序付け

プログラム･フロー内の命令の順序は、対応するメモリ･トランザクションの順序を必ずしも保証するわけではありません。これは、次の理由によるものです。

- プロセッサは、命令シーケンスの動作に影響を与えない限り、効率向上のために一部のメモリ･アクセスの順序を入れ換える場合がある。
- プロセッサが複数のバス･インタフェースを備えている。
- メモリ･マップ内のメモリまたはデバイスのウェイト･ステートが異なる。
- バッファされるメモリ･アクセスや投機的なメモリ･アクセスが存在する。

33 ページのメモリ･システムでのメモリ･アクセスの順序付けに、メモリ･システムがメモリ･アクセスの順序を保証する場合についての説明が記載されています。それらの場合に該当せず、メモリ･アクセスの順序が重要な場合は、ソフトウェアにメモリ･バリア命令を挿入して、強制的にアクセスを順序付ける必要があります。プロセッサでは、以下のメモリ･バリア命令を用意しています。

**DMB** Data Memory Barrier（データ･メモリ･バリア）（DMB）命令は、未処理のメモリ･トランザクションが、後続のメモリ･トランザクションよりも前に完了することを保証します。174 ページのDMB を参照してください。

**DSB** Data Synchronization Barrier（データ同期バリア）（DSB）命令は、未処理のメモリ･トランザクションが、後続の命令を実行する前に完了することを保証します。174 ページのDSB を参照してください。

**ISB** Instruction Synchronization Barrier（命令同期バリア）（ISB）命令は、完了したすべてのメモリ･トランザクションの影響が、後続の命令によって認識可能であることを保証します。175 ページのISB を参照してください。

#### MPU プログラミング

ISB 命令または例外からの復帰の前に DSB を使用して、後続の命令が新しい MPU 設定を使用するようにする必要があります。

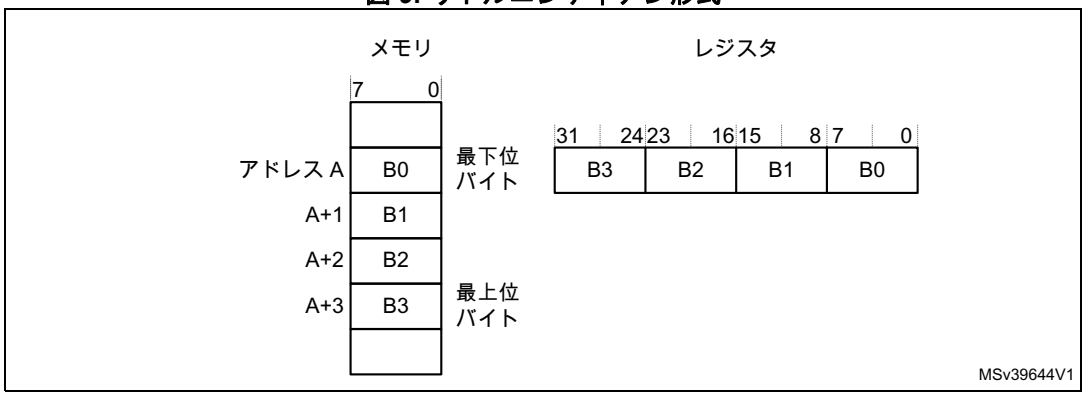
### 2.3.5 メモリのエンディアン形式

プロセッサは、メモリを 0 から昇順に番号が付けられたバイトの線形配列としてとらえます。たとえば、バイト 0 ~ 3 は最初にストアされるワードを、バイト 4 ~ 7 は 2 番目にストアされるワードを保持します。36 ページのリトルエンディアン形式に、ワードのデータをメモリに格納する方法を示します。

#### リトルエンディアン形式

リトルエンディアン形式では、プロセッサは、ワードの最下位バイトを一番小さい番号のバイトにストアし、最上位バイトを一番大きい番号のバイトにストアします。例：

図 9. リトルエンディアン形式




## 2.3.6 同期プリミティブ

Cortex®-M7 プロセッサの命令セットのサポートには、同期プリミティブ.のペアが含まれます。同期プリミティブは、スレッドまたはプロセスがメモリ位置への排他的アクセスを得るために使用できる非ブロッキング·メカニズムを提供します。ソフトウェアは、保証された読出し-変更-書込みのメモリ更新シーケンスの実行やセマフォ·メカニズムに、同期プリミティブを使用できます。

同期プリミティブのペアは、次の要素で構成されます。

**排他的ロード命令**

メモリ位置の値の読出しに使用され、その位置への排他的アクセスを要求します。

**排他的ストア命令**

同じメモリ位置への書込みに使用され、ステータス·ビットをレジスタに返します。このビットの値は、以下を意味します。

0 : そのスレッドまたはプロセスがメモリへの排他的アクセスを獲得し、書込みは成功であることを示します。

1 : そのスレッドまたはプロセスはメモリへの排他的アクセスを獲得できず、書込みは実行されなかったことを示します。

排他ロード命令と排他ストア命令のペアには次のものがあります。

- ワード命令 LDREX と STREX。
- ハーフワード命令 LDREXH と STREXH。
- バイト命令 LDREXB と STREXB

ソフトウェアでは、排他ロード命令とそれに対応する排他ストア命令を合わせて使用する必要があります。

メモリ位置の排他的な読出し／変更／書込みを実行するには、ソフトウェアは次の手順に従う必要があります。

1. 排他ロード命令を使用して、その位置の値を読み出します。
2. 必要に応じて値を修正します。
3. 排他ストア命令を使用して、そのメモリ位置に新しい値のライトバックを試行します。
4. 返されたステータス·ビットをテストします。このビットの値は、以下を意味します。

   0 : 読出し-変更-書込みは正常に完了しました。

   1 : 書込みは実行されませんでした。これは、ステップ 1 で返された値が古くなっている可能性があることを示します。ソフトウェアが読出し／変更／書込みのシーケンスを再試行する必要があります。

ソフトウェアは、同期プリミティブを使用して次のようにセマフォを実装できます。

1. 排他ロード命令を使用してセマフォ·アドレスを読み出して、セマフォが解放されているかどうかをチェックします。
2. セマフォが解放されている場合、排他ストア命令を使用してセマフォ·アドレスに要求値を書き込みます。
3. ステップ 2 から返されたステータス·ビットが排他ストアが成功したことを示している場合、セマフォの要求は完了しています。しかし、排他ストアが失敗した場合は、ソフトウェアがステップ 1 を実行した後に別のプロセスがそのセマフォを要求していた可能性があります。

Cortex®-M7 プロセッサには、プロセッサが排他ロード命令を実行した事実をタグ付けする排他アクセス·モニタが含まれています。プロセッサがマルチプロセッサ·システムに属していて、アドレスがメモリの共有領域にある場合はさらに、各プロセッサによる排他アクセスでアドレス指定したメモリ位置を、システムがグローバルにタグ付けします。

プロセッサは、次の場合に排他アクセスのタグを削除します。

- CLREX 命令を実行した場合。
- STREX命令を実行した場合（書込みが成功したかどうかは無関係）。
- 例外が発生した場合。これは、プロセッサが複数のスレッド間のセマフォ競合を解決できることを意味します。

マルチプロセッサ実装では：

- CLREX 命令の実行は、プロセッサのローカルな排他アクセス·タグのみ削除します。
- STREX 命令または例外の実行は、プロセッサのローカルな排他アクセス·タグを削除します。
- 共有メモリ領域に対する STREX 命令の実行でも、システム内のプロセッサのグローバル排他的アクセスタグが削除されることがあります。

同期プリミティブ命令の詳細については、82 ページのLDREX と STREX および 83 ページのCLREX を参照してください。

### 2.3.7 同期プリミティブのプログラミングのヒント

ISO/IEC C は直接排他アクセス命令を生成できません。これらの命令の生成に対して、CMSIS では次の組込み関数を用意しています。

**表 18. 排他アクセス命令のための CMSIS 関数**

| 命令 | CMSIS 関数 |
|---|---|
| LDREX | `uint32_t __LDREXW (uint32_t *addr)` |
| LDREXH | `uint16_t __LDREXH (uint16_t *addr)` |
| LDREXB | `uint8_t __LDREXB (uint8_t *addr)` |
| STREX | `uint32_t __STREXW (uint32_t value, uint32_t *addr)` |
| STREXH | `uint32_t __STREXH (uint16_t value, uint16_t *addr)` |
| STREXB | `uint32_t __STREXB (uint8_t value, uint8_t *addr)` |
| CLREX | `void __CLREX (void)` |

例：

```
uint16_t  value;
uint16_t *address = 0x20001002;
value = __LDREXH (address);    // load 16-bit value from memory address
0x20001002
```

## 2.4 例外モデル

このセクションでは、例外モデルについて説明します。次の内容を説明します。

- 例外状態を参照してください。
- 例外のタイプを参照してください。
- 41 ページの例外ハンドラを参照してください。
- 42 ページのベクタテーブルを参照してください。
- 43 ページの例外の優先度を参照してください。
- 43 ページの割込み優先度のグループ化を参照してください。
- 44 ページの例外の開始と復帰を参照してください。

### 2.4.1 例外状態

各例外は、次のいずれかの状態です。

**非アクティブ** 例外はアクティブでも保留中でもありません。

**ペンディング** 例外はプロセッサによる処理の待機中です。

ペリフェラルまたはソフトウェアからの割込み要求により、対応する割込みの状態が保留中に変化することがあります。

**アクティブ** 例外はプロセッサにより処理中で、完了していません。

**注： 例外ハンドラは、他の例外ハンドラの実行に割り込むことができます。この場合、両方の例外がアクティブ状態になります。**

**アクティブかつ保留中** 例外はプロセッサにより処理中であり、さらに同じソースの別の例外が保留中です。

### 2.4.2 例外のタイプ

例外のタイプを次に示します。

**リセット** リセットは、パワーアップまたはウォーム·リセットによって起動されます。例外モデルでは、リセットは特殊な形式の例外として扱われます。リセットがアサートされると、命令のどの時点においても、プロセッサの動作が停止します。リセットがネゲートされると、ベクタ·テーブルのリセット·エントリにより提供されるアドレスから実行が再開されます。実行は、スレッド·モードで特権実行として再開されます。

**NMI** ノンマスカブル割込み（NMI）は、ペリフェラルからの信号またはソフトウェアからのトリガによって発生します。これは、リセット以外では、優先度が最も高い例外です。常に有効であり、優先度は -2 に固定されています。NMI は、

- 他の例外によって、マスクされたり、アクティブ化を妨げられることはありません。
- リセット以外の例外によって横取りされることはありません。

**HardFault** ハード・フォルトは、通常処理または例外処理でのエラーが原因で発生する例外です。ハード・フォルトの優先度は -1 に固定されています。これは、設定可能な優先度を持つどの例外よりも優先度が高いことを意味します。

**SVCall** *Supervisor Call*（スーパーバイザ･コール）（SVC）は、SVC 命令によりトリガされる例外です。OS 環境では、アプリケーションは SVC 命令を使用して OS カーネル関数やデバイス･ドライバにアクセスできます。

**PendSV** PendSV は割込み駆動のシステムレベル･サービス要求です。OS 環境では、他にアクティブな例外が存在しない場合に PendSV を使用してコンテキストを切り替えます。

**SysTick** SysTick 例外は、システム･タイマが 0 に達したときに生成する例外です。ソフトウェアで SysTick 例外を生成することもできます。OS 環境では、プロセッサはこの例外をシステム･ティックとして使用できます。

**割込み（IRQ）** 割込み（IRQ）は、ペリフェラルによる信号またはソフトウェアの要求によって生成される例外です。割込みはすべて、命令の実行に対して非同期です。システム内で、ペリフェラルは割込みを使用してプロセッサとやりとりします。

**表 19. さまざまな例外タイプのプロパティ**

| 例外番号(1) | IRQ 番号（1） | 例外のタイプ | 優先順位 | ベクタアドレス(2) | アクティブ化 |
|---|---|---|---|---|---|
| 1 | - | リセット | -3、最高 | 0x00000004 | 非同期 |
| 2 | -14 | NMI | -2 | 0x00000008 | 非同期 |
| 3 | -13 | HardFault | -1 | 0x0000000C | 同期 |
| 4 | -12 | MemManage | 設定可能(3) | 0x00000010 | 同期 |
| 5 | -11 | BusFault | 設定可能（3） | 0x00000014 | 正確な場合は同期、不正確な場合は、非同期 |
| 6 | -10 | UsageFault | 設定可能（3） | 0x00000018 | 同期 |
| 7～10 | - | 予約済みです。 | - | - | - |
| 11 | -5 | SVCall | 設定可能（3） | 0x0000002C | 同期 |
| 12～13 | - | 予約済みです。 | - | - | - |
| 14 | -2 | PendSV | 設定可能（3） | 0x00000038 | 非同期 |
| 15 | -1 | SysTick | 設定可能（3） | 0x0000003C | 非同期 |
| 15 | - | 予約済みです。 | - | - | - |
| 16 以上 | 0 以上 | 割込み（IRQ） | 設定可能(4) | 0x00000040 以上(5) | 非同期 |

1. ソフトウェア層を単純化するため、CMSIS は IRQ 番号のみを使用します。割込み以外の例外には負の値を使用します。IPSR は、例外番号を返します。23 ページの割込みプログラム･ステータス･レジスタを参照してください。
2. 詳細については、図 10：42 ページのベクタテーブルを参照してください。
3. 199 ページのシステム･ハンドラ優先度レジスタを参照してください。
4. 185 ページの割込み優先度レジスタを参照
5. 4 ずつ増加します。

リセット以外の非同期例外の場合、プロセッサは、例外がトリガされてから例外ハンドラを開始するまでに、追加の命令を実行できます。

特権ソフトウェアは、40 ページの表 19 で優先度設定可能と記載されている例外を無効にできます。

- 201 ページのシステム･ハンドラ制御およびステート・レジスタ
- 183 ページの割込みクリア・イネーブル･レジスタを参照してください。

ハード・フォルト、メモリ管理フォールト、バス・フォールト、用法フォールトの詳細については、セクション 2.5 : 46 ページのフォールト処理を参照してください。

### 2.4.3 例外ハンドラ

プロセッサは、以下を使用して例外を処理します。

**割込みサービス･ルーチン（ISR）** IRQ0 ~ IRQ239 の割込みは、ISR が処理する例外です。

**フォールト･ハンドラ** ハード・フォルト、メモリ管理フォールト、用法フォールト、およびバス・フォールトは、フォールト・ハンドラによって処理されるフォールト例外です。

**システム･ハンドラ** NMI、PendSV、SVCall、SysTick、およびフォールト例外はすべて、システム･ハンドラが処理するシステム例外です。

### 2.4.4 ベクタテーブル

ベクタ·テーブルには、スタック·ポインタのリセット値、およびすべての例外ハンドラの開始アドレス（例外ベクタとも呼ばれます）が格納されています。42 ページの図 10 に、ベクタ·テーブルの例外ベクタの順序を示します。各ベクタの最下位ビットは、例外ハンドラが Thumb コードであることを示す、1 である必要があります。24 ページのThumb 状態を参照してください。

**図 10. ベクタテーブル**


システム·リセット時のベクタ·テーブル·アドレスは 0x00000000 です。特権ソフトウェアは、VTOR への書込みによって、ベクタ·テーブルの開始アドレスを 0x00000000 ~ 0xFFFFFF80 の範囲の別のメモリ位置に再配置できます。

シリコン ベンダーは、実装される割込みの数に依存する上限値を設定する必要があります。最小アライメントは 32 ワードで、最大 16 の割込みに対して十分です。割込みの数を増やす場合は、次の 2 のべき乗に切り上げて配置を調整します。たとえば、21 本の割込みが必要である場合、必要なテーブルサイズは 37 ワードであり、次の 2 のべき乗は 64 なので、アライメントは 64 ワードの境界上にある必要があります。194 ページのベクタ·テーブル·オフセット·レジスタを参照してください。

Arm 社では、システム・メモリマップの CODE、SRAM、外部 RAM、または外部デバイス領域のいずれかにベクタテーブルを配置することを推奨します。30 ページのCortex®-M7 の構成を参照してください。ペリフェラル、プライベート・ペリフェラル・バス、またはベンダー固有のメモリ領域を使用すると、システムによっては予測できない動作になることがあります。これは、プロセッサが、ロー

ド／ストア命令と、これらのメモリ領域でのベクタ・フェッチに異なるインタフェースを使用するためです。ベクタテーブルがキャッシュ可能（cacheable）なメモリ領域にある場合、コアは、ベクタへのロードまたはストアをすべて自己修正コードとして扱い、キャッシュメンテナンス命令を使用して、データ・キャッシュおよび命令キャッシュへの更新を同期させる必要があります。240 ページのキャッシュ・メンテナンスの設計のヒントとコツを参照してください。

### 2.4.5 例外の優先度

40 ページの表 19 に示すように、すべての例外には次のような優先度が関連付けられています。

- 優先度の値が小さいほど、優先順位が高いことを示します。
- リセット、ハード・フォルト、および NMI を除くすべての例外の優先度は設定可能です。

ソフトウェアで優先度が設定されない場合、優先度が設定可能なすべての例外の優先度は 0 になります。例外の優先度の設定の詳細については、次を参照してください。

- 199 ページのシステム･ハンドラ優先度レジスタを参照してください。
- 185 ページの割込み優先度レジスタを参照してください。

注： **設定可能な優先度の値の範囲は、0 ~ 255 です。これは、固定された負の優先度値を持つリセット、ハード・フォルト、および NMI 例外は、常に他のどの例外よりも優先度が高いことを意味します。**

たとえば、IRQ[0] に大きい優先度値を、IRQ[1] に小さい優先度値を割り当てることは、IRQ[1] のほうが IRQ[0] より優先度が高いことを意味します。IRQ[1] と IRQ[0] の両方がアサートされている場合、IRQ[0] より先に IRQ[1] が処理されます。

同じ優先度を持つ複数の例外が保留中の場合、例外番号が最も小さい例外が優先されます。たとえば、同じ優先度を持つ IRQ[0] と IRQ[1] が保留中の場合、IRQ[1] より先に IRQ[0] が処理されます。

プロセッサが例外ハンドラを実行中に、それより優先度の高い例外が発生した場合、例外ハンドラは横取りされます。例外を処理中にその例外と同じ優先度の例外が発生した場合、例外番号に関係なく、ハンドラは横取りされません。ただし、新しい割込みのステータスは保留中になります。

### 2.4.6 割込み優先度のグループ化

割込みを使用するシステム内の優先度制御を強化するために、NVIC は優先度のグループ化をサポートしています。これにより、割込み優先度レジスタの各エントリが、次の 2 つのフィールドに分けられます。

- グループ優先度を定義する上位フィールド。
- グループ内のサブ優先度を定義する下位フィールド

割込み例外の横取りは、グループ優先度のみによって決まります。プロセッサが割込み例外ハンドラを実行中に、処理されている割込みと同じグループ優先度を持つ別の割込みが発生しても、ハンドラは横取りされません。

同じグループ優先度を持つ複数の割込みが保留中の場合、それらの処理順序はサブ優先度フィールドによって決まります。同じグループ優先度とサブ優先度を持つ複数の割込みが保留中の場合、IRQ 番号が最も小さい割込みが最初に処理されます。

割込み優先度フィールドをグループ優先度とサブ優先度に分割することについての詳細は、194 ページのアプリケーション割込みおよびリセット制御レジスタを参照してください。

### 2.4.7 例外の開始と復帰

例外処理の説明では、次の用語を使用します。

**横取り** プロセッサが例外ハンドラを実行しているとき、処理されている例外より優先度が高い例外が発生すると、例外ハンドラが横取りされる可能性があります。割込みによる横取りの詳細については、43 ページの割込み優先度のグループ化を参照してください。

例外が別の例外を横取りする場合、これらをネストされた例外と呼びます。詳細については、44 ページの例外の開始を参照してください。

**復帰** 例外ハンドラが完了し、次の条件が満たされたときに発生します。

- 処理されるための十分な優先度を持つ保留中の例外が存在しない。
- 完了した例外ハンドラが、後着の例外を処理していなかった

プロセッサは、スタックをポップして、プロセッサの状態を割込み発生前の状態に復元します。詳細については、46 ページの例外からの復帰を参照してください。

**テールチェイン** 例外処理を高速化するメカニズムです。例外ハンドラの完了時に、例外開始要件を満たす保留中の例外が存在する場合、スタックのポップはスキップされて、新しい例外ハンドラに制御が移されます。

**後着** 横取りを高速化するメカニズムです。例外の状態を保存している途中で、それより優先度の高い例外が発生した場合、プロセッサは優先度の高い例外の処理に切り替えて、その例外のベクタ・フェッチを開始します。状態保存は後着の影響を受けません。保存される状態はどちらの例外でも同じだからです。したがって、状態保存は割り込まれずに続行されます。プロセッサは、元の例外の例外ハンドラの最初の命令がプロセッサの実行ステージに入るまでは、後着の例外を受け入れることができます。後着の例外の例外ハンドラから復帰するときは、通常のテールチェイン・ルールが適用されます。

#### 例外の開始

例外は、十分な優先度を持つ保留中の例外が存在し、次のどちらかの条件が満たされる場合に開始されます。

- プロセッサがスレッド・モードである。
- 新しい例外の優先度が処理されている例外より高い（この場合は新しい例外が元の例外を横取りする）

例外が別の例外を横取りする場合、これらの例外はネストされています。

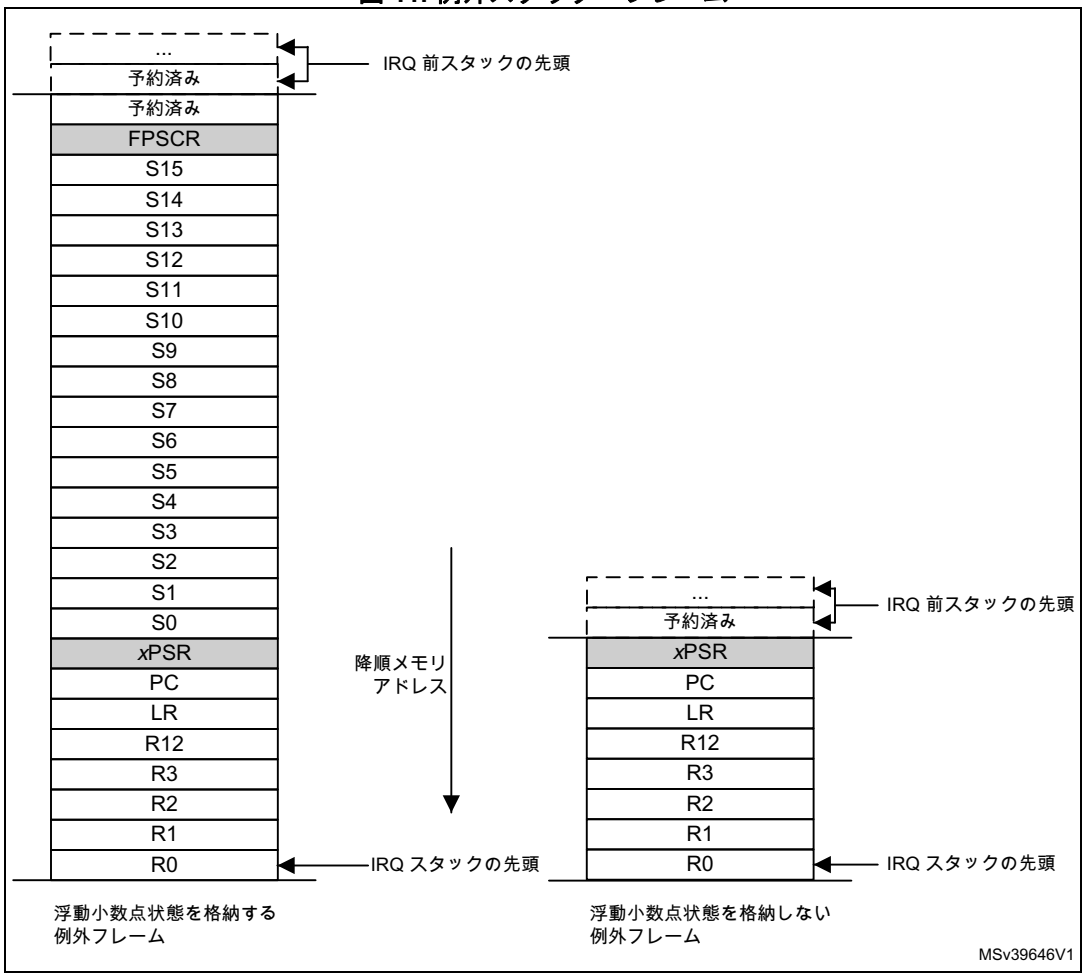
十分な優先度とは、例外の優先順位が、マスク・レジスタによって設定されている制限よりも高いことを意味します。25 ページの例外マスク・レジスタを参照ください。これより低い優先度を持つ例外は保留され、プロセッサによって処理されません。

プロセッサは、例外を取得するとき、それがテールチェインされている例外または後着の例外である場合を除いて、情報を現在のスタックにプッシュします。この動作はスタッキングと呼ばれ、8 データ・ワードの構造はスタック・フレームと呼ばれます。

Cortex®-M7 プロセッサは、浮動小数点ルーチンを使用しているとき、例外開始時に構成されている浮動小数点状態を自動的にスタックします。45 ページの図 11 に、割込みまたは例外の結果として浮動小数点状態がスタックに保持されている場合の Cortex®-M7 スタック・フレーム・レイアウトを示します。

**注：** **浮動小数点状態用のスタック空間が割り当てられていない場合のスタック･フレームは、FPU のない Armv7-M 実装のスタック･フレームと同じです。45 ページの図 11 には、このスタック･フレームも示されています。**

**図 11. 例外スタック・フレーム**




スタッキングの直後のスタック･ポインタは、スタック･フレームの最下位アドレスを示しています。スタック･フレームのアライメントは、Configuration Control register（構成制御レジスタ）（CCR）の STKALIGN ビットを使って制御されます。

**注：** **Cortex®-M7 プロセッサの CCR.STKALIGN は読出し専用で、値は 1 です。これは、スタック・アドレスが常に 8 バイトにそろっていることを意味します。**

スタック･フレームには、復帰アドレスが含まれています。これは、割り込まれたプログラムの次の命令のアドレスです。この値は、割り込まれたプログラムが再開されるように、例外からの復帰時に PC に復元されます。

プロセッサは、スタック操作と並行して、ベクタ･テーブルから例外ハンドラの開始アドレスを読み出すベクタ･フェッチを実行します。スタッキングが完了すると、プロセッサは例外ハンドラの実行を開始し、それと同時に EXC_RETURN 値を LR に書き込みます。これは、スタック･フレームに対応するスタック･ポインタおよび例外ハンドラが開始される前のプロセッサの動作モードを示します。

例外を開始する間にそれより優先度の高い例外が発生しなかった場合、プロセッサは、例外ハンドラの実行を開始し、それに対応する割込みのステータスを自動的に保留中からアクティブに変更します。

例外を開始する間にそれより優先度の高い別の例外が発生した場合、プロセッサは、この例外の例外ハンドラの実行を開始し、その前の例外のステータスは保留中のままで変更しません。これは、後着のケースです。

### 例外からの復帰

例外からの復帰は、プロセッサがハンドラ·モードであり、次のいずれかの命令を実行して EXC_RETURN 値を PC にロードすると発生します。

- PC をロードする LDM 命令または POP 命令。
- PC をデスティネーションとする LDR 命令。
- 任意のレジスタを使用する BX 命令

EXC_RETURN は、例外開始時に LR にロードされた値です。例外メカニズムはこの値を使用して、プロセッサが例外ハンドラを完了したタイミングを検出します。この値の最下位 5 ビットは、復帰スタックとプロセッサ·モードに関する情報を提供します。表 20 に、EXC_RETURN 値と例外からの復帰動作の説明を示します。

EXC_RETURN 値はすべて、ビット [31:5] が 1 にセットされています。この値が PC にロードされると、プロセッサは例外が完了したことを認識し、適切な例外からの復帰シーケンスを開始します。

**表 20. 例外からの復帰動作**

| EXC_RETURN[31:0] | 説明 |
|---|---|
| 0xFFFFFFF1 | ハンドラ·モードに復帰します。例外からの復帰では MSP の非浮動小数点状態を使用し、復帰後の実行では MSP を使用します。 |
| 0xFFFFFFF9 | スレッド·モードに復帰します。例外からの復帰では MSP の非浮動小数点状態を使用し、復帰後の実行では MSP を使用します。 |
| 0xFFFFFFFD | スレッド·モードに復帰します。例外からの復帰では PSP の非浮動小数点状態を使用し、復帰後の実行では PSP を使用します。 |
| 0xFFFFFFE1 | ハンドラ·モードに復帰します。例外からの復帰では MSP の浮動小数点状態を使用し、復帰後の実行では MSP を使用します。 |
| 0xFFFFFFE9 | スレッド·モードに復帰します。例外からの復帰では MSP の浮動小数点状態を使用し、復帰後の実行では MSP を使用します。 |
| 0xFFFFFFED | スレッド·モードに復帰します。例外からの復帰では PSP の浮動小数点状態を使用し、復帰後の実行では PSP を使用します。 |

## 2.5 フォールト処理

フォールトは、例外のサブセットです。39 ページの例外モデルをご覧ください。フォールトは次によって生成されます。

- 次のタイミングでのバス·エラー
  - 命令フェッチまたはベクタ·テーブルのロード
  - データ·アクセス
- 未定義命令など、内部で検出されたエラー
- Execute Never（決して実行しない）（XN）とマークされたメモリ領域にある命令の実行の試行
- 特権違反または管理されていない領域へのアクセス試行により発生する MPU フォールト

### 2.5.1 フォールト・タイプ

表 21 に、フォールトのタイプ、フォールトに使用するハンドラ、対応するフォールト・ステータス・レジスタ、およびフォールト発生を示すレジスタ・ビットを示します。フォールト・ステータス・レジスタの詳細については、197 ページの設定および制御レジスタを参照してください。

**表 21. フォールト**

| フォールト | ハンドラ | ビット名 | フォールト・ステータス・レジスタ |
|---|---|---|---|
| ベクタ読出し時のバス・エラー | HardFault | VECTTBL | 207 ページのハード・フォールト・ステータス・レジスタ |
| ハード・フォールトに移行されたフォールト | | FORCED | |
| MPU またはデフォルトのメモリ・マップの不一致： | MemManage | - | - |
| 命令アクセス時 | | IACCVIOL(1) | 203 ページのメモリ管理フォールト・ステータス・レジスタ |
| データ・アクセス時 | | DACCVIOL | |
| 例外スタッキング中 | | MSTKERR | |
| 例外アンスタッキング中 | | MUNSKERR | |
| レイジーな浮動小数点状態の保持中 | | MLSPERR | |
| バスエラー： | BusFault | - | - |
| 例外スタッキング中 | | STKERR | 204 ページのバス・フォールト・ステータス・レジスタ |
| 例外アンスタッキング中 | | UNSTKERR | |
| 命令プリフェッチ中 | | IBUSERR | |
| レイジーな浮動小数点状態の保持中 | | LSPERR | |
| 正確なデータ・バス・エラー | | PRECISERR | |
| 不正確なデータ・バス・エラー | | IMPRECISERR | |
| コプロセッサへのアクセスの試行 | UsageFault | NOCP | 206 ページの用法フォールト・ステータス・レジスタ |
| 未定義命令 | | UNDEFINSTR | |
| 無効な命令セット状態への移行の試行(2) | | INVSTATE | |
| 無効な EXC_RETURN 値 | | INVPC | |
| 整列されていない不正なロードまたはストア | | UNALIGNED | |
| 0 による除算 | | DIVBYZERO | |

1. プロセッサに MPU が搭載されていない場合、または MPU が無効になっている場合でも、XN 領域へのアクセス時に発生します。
2. Thumb 命令セット以外の命令セットの使用を試行、または ICI の継続により多重ロード／ストア以外の命令に復帰。

### 2.5.2 フォールトの移行とハード・フォールト

ハード・フォルトを除くすべてのフォールト例外には、設定可能な例外優先度があります。199 ページのシステム・ハンドラ優先度レジスタを参照してください。ソフトウェアは、これらのフォールトに対してハンドラの実行を無効にすることができます。201 ページのシステム・ハンドラ制御およびステート・レジスタを参照してください。

通常は、例外優先度と例外マスク・レジスタの値によって、プロセッサがフォールト・ハンドラを開始するかどうか、およびフォールト・ハンドラが別のフォールト・ハンドラを横取りできるかどうかが決まります（39 ページの例外モデルを参照）。

状況によっては、設定可能な優先度を持つフォールトは、ハード･フォールトとして扱われます。これは優先度の移行と呼ばれ、フォールトはハード・フォルトに移行されたと言われます。次の場合にハード・フォルトへの移行が発生します。

- フォールト･ハンドラが、現在処理中のフォールトと同じ種類のフォールトを発生させた場合。フォールト･ハンドラが、現在の優先度レベルと同じ優先度であるため、それ自身を横取りできない場合に、このハード・フォルトへの移行が発生します。
- フォールト･ハンドラが、現在処理中のフォールトと同じかそれより低い優先度のフォールトを発生させた場合。これは、新しいフォールトのハンドラが現在実行中のフォールト･ハンドラを横取りできないためです。
- 例外ハンドラが、現在実行中の例外と同じかそれより低い優先度のフォールトを発生させた場合。
- フォールトが発生し、そのフォールトのハンドラが有効ではない場合。

バス・フォールト・ハンドラ開始時のスタックへのプッシュ中にバス・フォールトが発生した場合、バス・フォールトはハード・フォルトに移行しません。これは、破損したスタックによってフォールトが発生した場合、フォールト･ハンドラのスタックへのプッシュが失敗したとしてもそのハンドラは実行されることを意味します。フォールト･ハンドラは動作しますが、スタックの内容は破損しています。

**注：** **固定優先度のハード・フォルトを横取りできるのは、リセットと NMI のみです。ハード･フォールトは、リセット、NMI、または別のハード・フォルト除くすべての例外を横取りできます。**

### 2.5.3 同期および非同期バス・フォールト

Cortex®-M7 プロセッサの全てのバス・フォールトは、次のいずれかの動作でトリガされます。

- プロセッサのロード操作が同期している。
- プロセッサのストア操作が、デバイス領域および Strongly-ordered 領域へのストアを含め、非同期である。
- デバッガのロードまたはストア・アクセスは同期していて、デバッガ・インタフェースでのみ確認可能。

非同期バス・フォールトがトリガされた場合、バス・フォールト例外は保留されます。バス・フォールト・ハンドラが有効でない場合、代わりにハード・フォルト例外が保留されます。非同期バス・フォールトによって発生するハード・フォルトがロックアップに移行することはありません。

書込み後に IRQ がトリガされた場合､ISR が実行される前に書込みバッファが排出されない可能性があります。したがって、非同期バス・フォールトはコンテキスト境界を越えて発生する可能性があります。

同期バス・フォールトは、NMI またはハード・フォルト・ハンドラ内で発生した場合、ロックアップに移行します。

キャッシュのメンテナンス操作もバス・フォールトをトリガできます。詳細については、240 ページのフォールト処理の考慮事項を参照してください。

### 2.5.4 フォールト･ステータス･レジスタとフォールト･アドレス･レジスタ

フォールト･ステータス･レジスタは、フォールトの原因を示します。同期バス・フォールトとメモリ管理フォールトの場合、フォールト･アドレス･レジスタは、表 22 に示すように、フォールトを発生させた操作がアクセスしていたアドレスを示します。

**表 22. フォールト･ステータス･レジスタとフォールト･アドレス･レジスタ**

| ハンドラ | ステータスレジスタ名 | アドレス･レジスタ名 | レジスタの説明 |
|---|---|---|---|
| HardFault | HFSR | - | 207 ページのハード・フォールト・ステータス･レジスタ |
| MemManage | MMFSR | MMFAR | 203 ページのメモリ管理フォールト･ステータス･レジスタ<br>208 ページのメモリ管理フォールト･アドレス・レジスタ |
| BusFault | BFSR | BFAR | 204 ページのバス･フォールト･ステータス･レジスタ<br>209 ページのバス･フォールト･アドレス･レジスタ |
| UsageFault | UFSR | - | 206 ページの用法フォールト･ステータス･レジスタ |

### 2.5.5 ロックアップ

NMI またはハード・フォルトのハンドラを実行している間にフォールトが発生した場合、プロセッサはロックアップ状態に移行します。ロックアップ状態のプロセッサは、命令を一切実行しません。プロセッサは、次のいずれかが発生するまでロックアップ状態のままです。

- リセットされる。
- NMI が発生する。
- デバッガにより停止される。

注： **NMI ハンドラからロックアップ状態が発生した場合、その後の NMI では、プロセッサのロックアップ状態は変わりません。**

## 2.6 電源管理

Cortex®-M7 プロセッサの SLEEP モードは、消費電力を削減します。

- SLEEP モードでは、プロセッサのクロックが停止します。
- DEEP SLEEP モードでは、システムクロックを停止し、PLL とFlashメモリの電源を切ります。

SCR の SLEEPDEEP ビットにより、使用する SLEEP モードを選択します（196 ページのシステム制御レジスタを参照）。

SLEEP モードの動作の詳細については、STM32 製品用のリファレンス･マニュアルを参照してください。

このセクションでは、SLEEP モードに移行するメカニズムおよび SLEEP モードからウェイクアップするための条件について説明します。

### 2.6.1 SLEEP モードへの移行

このセクションでは、ソフトウェアがプロセッサを SLEEP モードに移行するために使用できるメカニズムについて説明します。

システムは、プロセッサをウェイクアップするデバッグ操作など、偽のウェイクアップ·イベントを生成できます。したがって、ソフトウェアが、そのようなイベントの後にプロセッサを SLEEP モードに戻すことができる必要があります。たとえば、プロセッサを SLEEP モードに戻すアイドル·ループをプログラムに組み込むことができます。

#### 割込みを待機

ウェイクアップ条件が真の場合を除き、割込み待機命令 WFI により、ただちに SLEEP モードに移行します。50 ページのWFI または sleep-on-exit からのウェイクアップを参照してください。プロセッサは、WFI 命令を実行すると、命令の実行を停止して、SLEEP モードに移行します。詳細については、179 ページのWFI を参照してください。

#### イベント待機

イベント待機命令 WFE は、1 ビットのイベント·レジスタの値に応じて、SLEEP モードへの移行を発生させます。プロセッサは WFE 命令を実行すると、イベント·レジスタの値をチェックします。

0：プロセッサは命令の実行を停止して、SLEEP モードに移行します。

1：プロセッサはレジスタを 0 にクリアし、SLEEP モードに移行することなく、引き続き命令を実行します。

詳細については、178 ページのWFE を参照してください。

イベント·レジスタが 1 の場合、プロセッサは WFE 命令の実行時に SLEEP モードにすることはできません。通常、外部のイベント信号がアサートされるか、システム内のプロセッサにより SEV 命令が実行されているためです。177 ページのSEV を参照してください。ソフトウェアはこのレジスタに直接アクセスすることはできません。

#### Sleep-on-exit

SCR の SLEEPONEXIT ビットが 1 にセットされている場合、プロセッサはすべての例外ハンドラの実行を完了するとスレッド·モードに戻り、ただちに SLEEP モードに移行します。このメカニズムは、例外の発生時にプロセッサの実行のみが必要なアプリケーションで使用されます。

### 2.6.2 SLEEP モードからのウェイクアップ

プロセッサがウェイクアップする条件は、SLEEP モードへの移行を引き起こしたメカニズムによって異なります。

#### WFI または sleep-on-exit からのウェイクアップ

通常プロセッサは、例外の開始を引き起こすのに十分な優先度の例外を検出した場合にのみウェイクアップします。一部の組込みシステムでは、プロセッサのウェイクアップ後、割込みハンドラの実行前に、システムの復元タスクを実行する必要がある場合があります。これを実現するには、PRIMASK ビットを 1 に、FAULTMASK ビットを 0 にセットします。有効で現在の例外優先度より高い優先度の割込みが入った場合、プロセッサはウェイクアップしますが、PRIMASK をゼロにセットするまで割込みハンドラを実行しません。PRIMASK と FAULTMASK の詳細については、25 ページの例外マスク·レジスタを参照してください。

### WFE からのウェイクアップ

プロセッサは次の場合にウェイクアップします。

- 例外の開始を引き起こすのに十分な優先度の例外を検出した場合。
- 外部イベント信号を検出した場合（外部イベント入力を参照）。
- マルチプロセッサ･システムで、システム内の別のプロセッサがSEV 命令を実行した場合。

また、SCR の SEVONPEND ビットが 1 にセットされている場合、新しい保留中の割込みは、その割込みが無効、あるいは例外の開始を引き起こすのに十分な優先度を持っていなくても、イベントをトリガし、プロセッサをウェイクアップします。SCR の詳細については、196 ページのシステム制御レジスタを参照してください。

## 2.6.3 外部イベント入力

プロセッサは外部イベント入力信号を供給します。ペリフェラルはこの信号を駆動でき、WFE からプロセッサをウェイクアップするか、内部の WFE イベント･レジスタを 1 にセットして後の WFE 命令でプロセッサが SLEEP モードに移行してはいけないことを示します。詳細については、50 ページのイベント待機 を参照してください。

## 2.6.4 電源管理に関するプログラミングのヒント

ISO/IEC C は直接 WFI および WFE 命令を生成できません。これらの命令に対して、CMSIS では次の関数を用意しています。

```
void __WFE(void) // Wait for Event
void __WFI(void) // Wait for Interrupt
```

# 3 Cortex-M7 命令セット

## 3.1 命令セットの概要

プロセッサは、Armv7-M 命令セットと、Armv7E-M アーキテクチャ・プロファイルによって提供される機能を実装します。表 23 にサポートされる命令を示します。

表 23：

- 山かっこ（<>）は、オペランドの代替形式を囲みます。
- 中かっこ（{}）は、任意のオペランドを囲みます。
- オペランド列はすべてが示されているわけではありません。
- Op2は、フレキシブルな第 2 オペランドで、レジスタにも定数にもできます。
- ほとんどの命令では、任意の条件コード接尾文字を使用できます。

命令およびオペランドの詳細については、該当する命令の説明を参照してください。

**表 23. Cortex®-M7 命令**

| ニーモニック | オペランド | 概要 | フラグ | ページ |
|---|---|---|---|---|
| ADC、ADCS | {Rd}、Rn、Op2 | キャリー付き加算 | N、Z、C、V | 86ページの3.5.1 |
| ADD、ADDS | {Rd}、Rn、Op2 | 加算 | N、Z、C、V | 86ページの3.5.1 |
| ADD、ADDW | {Rd}、Rn、#imm12 | 加算 | - | 86ページの3.5.1 |
| ADR | Rd、label | PC 相対アドレスのロード | - | 72ページの3.4.1 |
| AND、ANDS | {Rd}、Rn、Op2 | 論理積 | N、Z、C | 88ページの3.5.2 |
| ASR、ASRS | Rd、Rm、<Rs\|#n> | 算術右シフト | N、Z、C | 89ページの3.5.3 |
| B | label | 分岐 | - | 142ページの3.10.1 |
| BFC | Rd、#lsb、#width | ビット･フィールドのクリア | - | 139ページの3.9.1 |
| BFI | Rd、Rn、#lsb、#width | ビット･フィールドの挿入 | - | 139ページの3.9.1 |
| BIC、BICS | {Rd}、Rn、Op2 | ビット･クリア | N、Z、C | 88ページの3.5.2 |
| BKPT | #imm | ブレークポイント | - | 172ページの3.12.1 |
| BL | label | リンク付き分岐 | - | 142ページの3.10.1 |
| BLX | Rm | リンク付き間接分岐 | - | 142ページの3.10.1 |
| BX | Rm | 間接分岐 | - | 142ページの3.10.1 |
| CBNZ | Rn、label | ゼロとの比較と分岐（ゼロでない場合に分岐） | - | 144ページの3.10.2 |
| CBZ | Rn、label | ゼロとの比較と分岐（ゼロの場合に分岐） | - | 144ページの3.10.2 |
| CLREX | – | 排他をクリア | - | 83ページの3.4.10 |
| CLZ | Rd、Rm | 先行ゼロ･カウント | - | 90ページの3.5.4 |
| CMN | Rn、Op2 | 比較否定 | N、Z、C、V | 91ページの3.5.5 |
| CMP | Rn、Op2 | 比較 | N、Z、C、V | 91ページの3.5.5 |

表 23. Cortex®-M7 命令 （続き）

| ニーモニック | オペランド | 概要 | フラグ | ページ |
|---|---|---|---|---|
| CPSID | i | プロセッサ状態の変更、割込みを無効にする | - | 173ページの3.12.2 |
| CPSIE | i | プロセッサ状態の変更、割込みを有効にする | - | 173ページの3.12.2 |
| DMB | - | データ・メモリ・バリア | - | 174ページの3.12.3 |
| DSB | - | データ同期バリア | - | 174ページの3.12.4 |
| EOR、EORS | {Rd}、Rn、Op2 | 排他的論理和 | N、Z、C | 88ページの3.5.2 |
| ISB | - | 命令同期バリア | - | 175ページの3.12.5 |
| IT | - | If-Then 条件ブロック | - | 145ページの3.10.3 |
| LDM | Rn{!}、reglist | 多重レジスタ・ロード、ポスト・インクリメント | - | 78ページの3.4.6 |
| LDMDB、LDMEA | Rn{!}、reglist | 多重レジスタ・ロード、プレ・デクリメント | - | 78ページの3.4.6 |
| LDMFD、LDMIA | Rn{!}、reglist | 多重レジスタ・ロード、ポスト・インクリメント | - | 78ページの3.4.6 |
| LDR | Rt、[Rn、#offset] | レジスタ・ロード（ワード） | - | 72ページの3.4.2 |
| LDRB、LDRBT | Rt、[Rn、#offset] | レジスタ・ロード（バイト） | - | 72ページの3.4.2 |
| LDRD | Rt、Rt2、[Rn、#offset] | レジスタ・ロード（2 バイト） | - | 76ページの3.4.4 |
| LDREX | Rt、[Rn、#offset] | 排他的レジスタ・ロード | - | 82ページの3.4.9 |
| LDREXB | Rt、[Rn] | 排他的レジスタ・ロード（バイト） | - | 82ページの3.4.9 |
| LDREXH | Rt、[Rn] | 排他的レジスタ・ロード（ハーフワード） | - | 82ページの3.4.9 |
| LDRH、LDRHT | Rt、[Rn、#offset] | レジスタ・ロード（ハーフワード） | - | 72ページの3.4.2 |
| LDRSB、LDRSBT | Rt、[Rn、#offset] | レジスタ・ロード（符号付きバイト） | - | 72ページの3.4.2 |
| LDRSH、LDRSHT | Rt、[Rn、#offset] | レジスタ・ロード（符号付きハーフワード） | - | 72ページの3.4.2 |
| LDRT | Rt、[Rn、#offset] | レジスタ・ロード（ワード） | - | 72ページの3.4.2 |
| LSL、LSLS | Rd、Rm、<Rs\|#n> | 論理左シフト | N、Z、C | 89ページの3.5.3 |
| LSR、LSRS | Rd、Rm、<Rs\|#n> | 論理右シフト | N、Z、C | 89ページの3.5.3 |
| MLA | Rd、Rn、Rm、Ra | 積和、32 ビットの結果 | - | 111ページの3.6.1 |
| MLS | Rd、Rn、Rm、Ra | 積差、32 ビットの結果 | - | 111ページの3.6.1 |
| MOV、MOVS | Rd、Op2 | 転送 | N、Z、C | 92ページの3.5.6 |
| MOVT | Rd、#imm16 | 上位に転送 | - | 93ページの3.5.7 |
| MOVW、MOV | Rd、#imm16 | 16 ビット定数を転送 | N、Z、C | 92ページの3.5.6 |
| MRS | Rd、spec_reg | 特殊レジスタの内容の汎用レジスタへの転送 | - | 175ページの3.12.6 |

表 23. Cortex®-M7 命令 （続き）

| ニーモニック | オペランド | 概要 | フラグ | ページ |
|---|---|---|---|---|
| MSR | spec_reg、Rm | 汎用レジスタの内容の特殊レジスタへの転送 | N、Z、C、V | 176ページの3.12.7 |
| MUL、MULS | {Rd}、Rn、Rm | 乗算、32 ビットの結果 | N、Z | 111ページの3.6.1 |
| MVN、MVNS | Rd、Op2 | 転送して否定 | N、Z、C | 92ページの3.5.6 |
| NOP | - | 何もしない | - | 177ページの3.12.8 |
| ORN、ORNS | {Rd}、Rn、Op2 | 否定論理和 | N、Z、C | 88ページの3.5.2 |
| ORR、ORRS | {Rd}、Rn、Op2 | 論理和 | N、Z、C | 88ページの3.5.2 |
| PKHTB、PKHBT | {Rd}、Rn、Rm、Op2 | ハーフワード･パック | - | 135ページの3.8.1 |
| PLD | [Rn, #offset] | プリロード・データ | - | 80ページの3.4.7 |
| POP | reglist | レジスタをスタックからポップ | - | 81ページの3.4.8 |
| PUSH | reglist | レジスタをスタックにプッシュ | - | 81ページの3.4.8 |
| QADD | {Rd}、Rn、Rm | 飽和倍演算および加算 | Q | 129ページの3.7.3 |
| QADD16 | {Rd}、Rn、Rm | 16 ビット飽和加算 | - | 129ページの3.7.3 |
| QADD8 | {Rd}、Rn、Rm | 8 ビット飽和加算 | - | 129ページの3.7.3 |
| QASX | {Rd}、Rn、Rm | 飽和加算および減算、交換付き | - | 130ページの3.7.4 |
| QDADD | {Rd}、Rn、Rm | 飽和加算 | Q | 131ページの3.7.5 |
| QDSUB | {Rd}、Rn、Rm | 飽和倍演算および減算 | Q | 129ページの3.7.3 |
| QSAX | {Rd}、Rn、Rm | 飽和減算および加算、交換付き | - | 130ページの3.7.4 |
| QSUB | {Rd}、Rn、Rm | 飽和減算 | Q | 129ページの3.7.3 |
| QSUB16 | {Rd}、Rn、Rm | 16 ビット飽和減算 | - | 129ページの3.7.3 |
| QSUB8 | {Rd}、Rn、Rm | 8 ビット飽和減算 | - | 129ページの3.7.3 |
| RBIT | Rd、Rn | ビット順序反転 | - | 94ページの3.5.8 |
| REV | Rd、Rn | ワード内のバイト順序反転 | - | 94ページの3.5.8 |
| REV16 | Rd、Rn | 各ハーフワード内のバイト順序反転 | - | 94ページの3.5.8 |
| REVSH | Rd、Rn | 下位ハーフワード内のバイト順序を反転させ、符号拡張 | - | 94ページの3.5.8 |
| ROR、RORS | Rd、Rm、<Rs\|#n> | 右ローテート | N、Z、C | 89ページの3.5.3 |
| RRX、RRXS | Rd、Rm | 右ローテート、拡張付き | N、Z、C | 89ページの3.5.3 |
| RSB、RSBS | {Rd}、Rn、Op2 | 反転減算 | N、Z、C、V | 86ページの3.5.1 |
| SADD16 | {Rd}、Rn、Rm | 16 ビット符号付き加算 | GE | 95ページの3.5.9 |
| SADD8 | {Rd}、Rn、Rm | 8 ビット符号付き加算 | GE | 95ページの3.5.9 |
| SASX | {Rd}、Rn、Rm | 符号付き加算および減算、交換付き | GE | 100ページの3.5.14 |
| SBC、SBCS | {Rd}、Rn、Op2 | キャリー付き減算 | N、Z、C、V | 86ページの3.5.1 |
| SBFX | Rd、Rn、#lsb、#width | 符号付きビット･フィールド拡張 | - | 140ページの3.9.2 |

表 23. Cortex®-M7 命令 （続き）

| ニーモニック | オペランド | 概要 | フラグ | ページ |
|---|---|---|---|---|
| SDIV | {Rd}、Rn、Rm | 符号付き除算 | - | 125ページの3.6.12 |
| SEL | {Rd}、Rn、Rm | バイトの選択 | - | 107ページの3.5.21 |
| SEV | – | イベント送信 | - | 177ページの3.12.9 |
| SHADD16 | {Rd}、Rn、Rm | 結果を半減する 16 ビット符号付き加算 | - | 96ページの3.5.10 |
| SHADD8 | {Rd}、Rn、Rm | 結果を半減する 8 ビット符号付き加算 | - | 96ページの3.5.10 |
| SHASX | {Rd}、Rn、Rm | 結果を半減する符号付き加算および減算、交換付き | - | 97ページの3.5.11 |
| SHSAX | {Rd}、Rn、Rm | 結果を半減する符号付き減算および加算、交換付き | - | 97ページの3.5.11 |
| SHSUB16 | {Rd}、Rn、Rm | 結果を半減する 16 ビット符号付き減算 | - | 98ページの3.5.12 |
| SHSUB8 | {Rd}、Rn、Rm | 結果を半減する 8 ビット符号付き減算 | - | 98ページの3.5.12 |
| SMLABB、SMLABT、SMLATB、SMLATT | Rd、Rn、Rm、Ra | 符号付き Long 積和（ハーフワード） | Q | 113ページの3.6.3 |
| SMLAD、SMLADX | Rd、Rn、Rm、Ra | デュアル符号付き積和 | Q | 114ページの3.6.4 |
| SMLAL | RdLo、RdHi、Rn、Rm | 符号付き積和（32 x 32 + 64）、64 ビットの結果 | - | 115ページの3.6.5 |
| SMLALBB、SMLALBT、SMLALTB、SMLALTT | RdLo、RdHi、Rn、Rm | 符号付き Long 積和、ハーフワード | - | 115ページの3.6.5 |
| SMLALD、SMLALDX | RdLo、RdHi、Rn、Rm | デュアル符号付き Long 積和 | - | 115ページの3.6.5 |
| SMLAWB、SMLAWT | Rd、Rn、Rm、Ra | 符号付き積和、ワード×ハーフワード | Q | 113ページの3.6.3 |
| SMLSD | Rd、Rn、Rm、Ra | デュアル符号付き積差 | Q | 117ページの3.6.6 |
| SMLSLD | RdLo、RdHi、Rn、Rm | デュアル符号付き Long 積差 | | 117ページの3.6.6 |
| SMMLA | Rd、Rn、Rm、Ra | 符号付き上位ワード積和 | - | 119ページの3.6.7 |
| SMMLS、SMMLR | Rd、Rn、Rm、Ra | 符号付き上位ワード積差 | - | 119ページの3.6.7 |
| SMMUL、SMMULR | {Rd}、Rn、Rm | 符号付き上位ワード乗算 | - | 120ページの3.6.8 |
| SMUAD | {Rd}、Rn、Rm | デュアル符号付き乗加算 | Q | 121ページの3.6.9 |
| SMULBB、SMULBT、SMULTB、SMULTT | {Rd}、Rn、Rm | 符号付き乗算（ハーフワード） | - | 122ページの3.6.10 |
| SMULL | RdLo、RdHi、Rn、Rm | 符号付き乗算（32 x 32）、64 ビットの結果 | - | 124ページの3.6.11 |
| SMULWB、SMULWT | {Rd}、Rn、Rm | 符号付き乗算、ワード×ハーフワード | - | 122ページの3.6.10 |
| SMUSD、SMUSDX | {Rd}、Rn、Rm | デュアル符号付き積差 | - | 121ページの3.6.9 |

表 23. Cortex®-M7 命令 （続き）

| ニーモニック | オペランド | 概要 | フラグ | ページ |
|---|---|---|---|---|
| SSAT | Rd、#n、Rm、{shift #s} | 符号付き飽和 | Q | 127ページの3.7.1 |
| SSAT16 | Rd、#n、Rm | 16 ビット符号付き飽和 | Q | 128ページの3.7.2 |
| SSAX | {Rd}、Rn、Rm | 符号付き減算および加算、交換付き | GE | 100ページの3.5.14 |
| SSUB16 | {Rd}、Rn、Rm | 16 ビット符号付き減算 | - | 99ページの3.5.13 |
| SSUB8 | {Rd}、Rn、Rm | 8 ビット符号付き減算 | - | 99ページの3.5.13 |
| STM | Rn{!}、reglist | 複数レジスタのストア、ポスト･インクリメント | - | 78ページの3.4.6 |
| STMDB、STMEA | Rn{!}、reglist | 複数レジスタのストア、プレ･デクリメント | - | 78ページの3.4.6 |
| STMFD、STMIA | Rn{!}、reglist | 複数レジスタのストア、ポスト･インクリメント | - | 78ページの3.4.6 |
| STR | Rt、[Rn、#offset] | レジスタ･ストア（ワード） | - | 72ページの3.4.2 |
| STRB、STRBT | Rt、[Rn、#offset] | レジスタ･ストア（バイト） | - | 72ページの3.4.2 |
| STRD | Rt、Rt2、[Rn、#offset] | レジスタ･ストア（ダブルワード） | - | 72ページの3.4.2 |
| STREX | Rd、Rt、[Rn、#offset] | 排他的レジスタ･ストア | - | 82ページの3.4.9 |
| STREXB | Rd、Rt、[Rn] | 排他的レジスタ･ストア（バイト） | - | 82ページの3.4.9 |
| STREXH | Rd、Rt、[Rn] | 排他的レジスタ･ストア（ハーフワード） | - | 82ページの3.4.9 |
| STRH、STRHT | Rt、[Rn、#offset] | レジスタ･ストア（ハーフワード） | - | 72ページの3.4.2 |
| STRT | Rt、[Rn、#offset] | レジスタ･ストア（ワード） | - | 72ページの3.4.2 |
| SUB、SUBS | {Rd}、Rn、Op2 | 減算 | N、Z、C、V | 86ページの3.5.1 |
| SUB、SUBW | {Rd}、Rn、#imm12 | 減算 | - | 86ページの3.5.1 |
| SVC | #imm | スーパバイザ･コール | - | 86ページの3.5.1 |
| SXTAB | {Rd}、Rn、Rm、{ROR #} | 8 ビット値を 32 ビット値に拡張して加算 | - | 137ページの3.8.3 |
| SXTAB16 | {Rd}、Rn、Rm、{ROR #} | デュアルで 8 ビット値を 16 ビット値に拡張して加算 | - | 137ページの3.8.3 |
| SXTAH | {Rd}、Rn、Rm、{ROR #} | 16 ビット値を 32 ビット値に拡張して加算 | - | 137ページの3.8.3 |
| SXTB16 | {Rd}、Rm、{ROR #n} | 16 ビット値に符号拡張（バイト） | - | 141ページの3.9.3 |
| SXTB | {Rd}、Rm、{ROR #n} | 符号拡張（バイト） | - | 141ページの3.9.3 |
| SXTH | {Rd}、Rm、{ROR #n} | 符号拡張（ハーフワード） | - | 141ページの3.9.3 |
| TBB | [Rn、Rm] | テーブル分岐（バイト） | - | 147ページの3.10.4 |
| TBH | [Rn、Rm、LSL #1] | テーブル分岐（ハーフワード） | - | 147ページの3.10.4 |
| TEQ | Rn、Op2 | 等価テスト | N、Z、C | 101ページの3.5.15 |

表 23. Cortex®-M7 命令 （続き）

| ニーモニック | オペランド | 概要 | フラグ | ページ |
|---|---|---|---|---|
| TST | Rn、Op2 | テスト | N、Z、C | 101ページの3.5.15 |
| UADD16 | {Rd}、Rn、Rm | 16 ビット符号なし加算 | GE | 102ページの3.5.16 |
| UADD8 | {Rd}、Rn、Rm | 8 ビット符号なし加算 | GE | 102ページの3.5.16 |
| USAX | {Rd}、Rn、Rm | 符号なし減算および加算、交換付き | GE | 103ページの3.5.17 |
| UHADD16 | {Rd}、Rn、Rm | 結果を半減する 16 ビット符号なし加算 | - | 104ページの3.5.18 |
| UHADD8 | {Rd}、Rn、Rm | 結果を半減する 8 ビット符号なし加算 | - | 104ページの3.5.18 |
| UHASX | {Rd}、Rn、Rm | 結果を半減する符号なし加算および減算、交換付き | - | 105ページの3.5.19 |
| UHSAX | {Rd}、Rn、Rm | 結果を半減する符号なし減算および加算、交換付き | - | 105ページの3.5.19 |
| UHSUB16 | {Rd}、Rn、Rm | 結果を半減する 16 ビット符号なし減算 | - | 106ページの3.5.20 |
| UHSUB8 | {Rd}、Rn、Rm | 結果を半減する 8 ビット符号なし減算 | - | 106ページの3.5.20 |
| UBFX | Rd、Rn、#lsb、#width | 符号なしビット･フィールド拡張 | - | 140ページの3.9.2 |
| UDIV | {Rd}、Rn、Rm | 符号なし除算 | - | 125ページの3.6.12 |
| UMAAL | RdLo、RdHi、Rn、Rm | 符号なし Long 積和（32 x 32 + 32 +32）、64 ビットの結果 | - | 112ページの3.6.2 |
| UMLAL | RdLo、RdHi、Rn、Rm | 符号なし積和（32 x 32 + 64）､64 ビットの結果 | - | 112ページの3.6.2 |
| UMULL | RdLo、RdHi、Rn、Rm | 符号なし乗算（32 x 32）、64 ビットの結果 | - | 112ページの3.6.2 |
| UQADD16 | {Rd}、Rn、Rm | 16 ビット符号なし飽和加算 | - | 133ページの3.7.7 |
| UQADD8 | {Rd}、Rn、Rm | 8 ビット符号なし飽和加算 | - | 133ページの3.7.7 |
| UQASX | {Rd}、Rn、Rm | 符号なし飽和加算および減算、交換付き | - | 132ページの3.7.6 |
| UQSAX | {Rd}、Rn、Rm | 符号なし飽和減算および加算、交換付き | - | 132ページの3.7.6 |
| UQSUB16 | {Rd}、Rn、Rm | 16 ビット符号なし飽和減算 | - | 133ページの3.7.7 |
| UQSUB8 | {Rd}、Rn、Rm | 8 ビット符号なし飽和減算 | - | 133ページの3.7.7 |
| USAD8 | {Rd}、Rn、Rm | 符号なし絶対差の和 | - | 107ページの3.5.22 |
| USADA8 | {Rd}、Rn、Rm、Ra | 絶対差の符号なし和の累算。 | - | 108ページの3.5.23 |
| USAT | Rd、#n、Rm、{shift #s} | 符号なし飽和 | Q | 127ページの3.7.1 |
| USAT16 | Rd、#n、Rm | 16 ビット符号なし飽和 | Q | 128ページの3.7.2 |
| UASX | {Rd}、Rn、Rm | 符号なし加算および減算、交換付き | GE | 103ページの3.5.17 |

表 23. Cortex®-M7 命令 （続き）

| ニーモニック | オペランド | 概要 | フラグ | ページ |
|---|---|---|---|---|
| USUB16 | {Rd}、Rn、Rm | 16 ビット符号なし減算 | GE | 109ページの3.5.24 |
| USUB8 | {Rd}、Rn、Rm | 8 ビット符号なし減算 | GE | 109ページの3.5.24 |
| UXTAB | {Rd}、Rn、Rm、{ROR #} | ローテートし、8 ビット値を 32 ビット値に拡張して加算 | - | 137ページの3.8.3 |
| UXTAB16 | {Rd}、Rn、Rm、{ROR #} | ローテートし、デュアルで 8 ビット値を 16 ビット値に拡張して加算 | - | 137ページの3.8.3 |
| UXTAH | {Rd}、Rn、Rm、{ROR #} | ローテートし、16 ビット値を 32 ビット値に符号なし拡張して加算 | - | 137ページの3.8.3 |
| UXTB | {Rd}、Rm、{ROR #n} | ゼロ拡張（バイト） | - | 141ページの3.9.3 |
| UXTB16 | {Rd}、Rm、{ROR #n} | 16 ビット値に符号なし拡張（バイト） | - | 141ページの3.9.3 |
| UXTH | {Rd}、Rm、{ROR #n} | ゼロ拡張（ハーフワード） | - | 141ページの3.9.3 |
| VABS.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 浮動小数点の絶対値 | - | 150ページの3.11.1 |
| VADD.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 浮動小数点加算 | - | 150ページの3.11.2 |
| VCMP.F<32\|64> | <Sd\|Dd>、<<Sm\|Dm> \| #0.0> | 2 つの浮動小数点レジスタ、または 1 つの浮動小数点レジスタとゼロを比較 | FPSCR | 151ページの3.11.3 |
| VCMPE.F<32\|64> | <Sd\|Dd>、<<Sm\|Dm> \| #0.0> | 無効演算チェックで、2 つの浮動小数点レジスタ、または 1 つの浮動小数点レジスタとゼロを比較 | FPSCR | 151ページの3.11.3 |
| VCVT | F<32\|64>.<S\|U>、<16\|32> | 浮動小数点数から固定小数点数への変換 | - | 153ページの3.11.5 |
| VCVT | <S\|U>、<16\|32>.F<32\|64> | 固定小数点数から浮動小数点数への変換 | | 153ページの3.11.5 |
| VCVT.S32.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 浮動小数点数から整数への変換 | - | 152ページの3.11.4 |
| VCVT<B\|T>.F<32\|64>.F16 | <Sd\|Dd>、Sm | 半精度値から単精度値または倍精度値への変換 | - | 154ページの3.11.6 |
| VCVTA.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 最近傍点への方向丸めを使用した浮動小数点から整数への変換 | - | 170ページの3.11.33 |
| VCVTM.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 負の無限大への方向丸めを使用した浮動小数点から整数への変換 | - | 170ページの3.11.33 |
| VCVTN.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 一番近い偶数への方向丸めを使用した浮動小数点から整数への変換 | - | 170ページの3.11.33 |
| VCVTP.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 正の無限大への方向丸めを使用した浮動小数点から整数への変換 | - | 170ページの3.11.33 |
| VCVTR.S32.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 丸めを使った浮動小数点数と整数との変換。 | FPSCR | 152ページの3.11.4 |

表 23. Cortex®-M7 命令 （続き）

| ニーモニック | オペランド | 概要 | フラグ | ページ |
|---|---|---|---|---|
| VCVT<B\|T>.F16.F<32\|64> | Sd、<Sm\|Dm> | 単精度または倍精度レジスタの半精度レジスタへの変換 | - | 153ページの3.11.5 |
| VDIV.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 浮動小数点除算 | - | 154ページの3.11.7 |
| VFMA.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 結合浮動小数点積和 | - | 155ページの3.11.8 |
| VFMS.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 浮動小数点結合積差 | - | 155ページの3.11.8 |
| VFNMA.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 浮動小数点結合積和の結果を符号反転 | - | 156ページの3.11.9 |
| VFNMS.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 浮動小数点結合乗算を減算した結果を符号反転 | - | 156ページの3.11.9 |
| VLDM.F<32\|64> | Rn{!}、list | 拡張レジスタの多重ロード | - | 156ページの3.11.10 |
| VLDR.F<32\|64> | <Sd\|Dd>、[<Rn>{,#+/-<imm>}] | メモリからの拡張レジスタ・ロード | - | 157ページの3.11.11 |
| VLDR.F<32\|64> | <Sd\|Dd>、<label> | メモリからの拡張レジスタ・ロード | - | 157ページの3.11.11 |
| VLDR.F<32\|64> | <Sd\|Dd>、[PC,#-0] | メモリからの拡張レジスタ・ロード | - | 157ページの3.11.11 |
| VMLA.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 浮動小数点積和 | - | 158ページの3.11.12 |
| VMLS.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 浮動小数点積差 | - | 158ページの3.11.12 |
| VMAXNM.F<32\|64> | <Sd\|Dd>、<Sn\|Dn>、<Sm\|Dm> | IEEE754-2008 NaN 処理での 2 つの浮動小数点数の最大 | - | 169ページの3.11.32 |
| VMINNM.F<32\|64> | <Sd\|Dd>、<Sn\|Dn>、<Sm\|Dm> | IEEE754-2008 NaN 処理での 2 つの浮動小数点数の最小 | - | 169ページの3.11.32 |
| VMOV | <Sd\|Dd>、<Sm\|Dm> | 浮動小数点レジスタ間の転送 | - | 162ページの3.11.19 |
| VMOV | <Sn\|Dn>、Rt | Arm コア・レジスタ値を単精度レジスタにコピー | - | 162ページの3.11.19 |
| VMOV | <Sm\|Dm>、<Sm\|Dm>1、Rt、Rt2 | 2 つの Arm コア・レジスタ値を 2 つの単精度レジスタにコピー | - | 162ページの3.11.19 |
| VMOV | Dd[x]、Rt | Arm コア・レジスタ値をスカラにコピー | - | 162ページの3.11.19 |
| VMOV | Rt、Dn[x] | スカラ値を Arm コア・レジスタにコピー | - | 162ページの3.11.19 |
| VMOV.F<32\|64> | <Sd\|Dd>、#imm | 浮動小数点のイミディエート値の転送 | - | 162ページの3.11.19 |
| VMUL.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 浮動小数点乗算 | - | 163ページの3.11.22 |
| VNEG.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 浮動小数点符号反転 | - | 164ページの3.11.23 |

表 23. Cortex®-M7 命令 （続き）

| ニーモニック | オペランド | 概要 | フラグ | ページ |
|---|---|---|---|---|
| VNMLA.F<32\|64> | <Sd\|Dd>、<Sn\|Dn>、<Sm\|Dm> | 浮動小数点積和 | - | 164ページの3.11.24 |
| VNMLS.F<32\|64> | <Sd\|Dd>、<Sn\|Dn>、<Sm\|Dm> | 浮動小数点積差 | - | 164ページの3.11.24 |
| VNMUL.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 浮動小数点乗算 | - | 164ページの3.11.24 |
| VRINTA.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 方向丸めを使用した浮動小数点フォーマット変換での浮動小数点から整数への変換 | - | 171ページの3.11.35 |
| VRINTM.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 方向丸めを使用した浮動小数点フォーマット変換での浮動小数点から整数への変換 | - | 171ページの3.11.35 |
| VRINTN.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 方向丸めを使用した浮動小数点フォーマット変換での浮動小数点から整数への変換 | - | 171ページの3.11.35 |
| VRINTP.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 方向丸めを使用した浮動小数点フォーマット変換での浮動小数点から整数への変換 | - | 171ページの3.11.35 |
| VRINTR.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 浮動小数点フォーマット変換での浮動小数点から整数への変換 | - | 170ページの3.11.34 |
| VRINTX.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 浮動小数点フォーマット変換での浮動小数点から整数への変換 | - | 170ページの3.11.34 |
| VRINTZ.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 浮動小数点フォーマット変換での浮動小数点から整数への変換 | - | 171ページの3.11.35 |
| VSEL.F<32\|64> | <Sd\|Dd>、<Sn\|Dn>、<Sm\|Dm> | 条件付き VMOV レジスタの対に代わるレジスタの選択 | - | 169ページの3.11.31 |
| VSQRT.F<32\|64> | <Sd\|Dd>、<Sm\|Dm> | 浮動小数点平方根 | - | 166ページの3.11.27 |
| VSTR.F<32\|64> | <Sd\|Dd>、[Rn] | 拡張レジスタをメモリにストア | - | 167ページの3.11.29 |
| VSUB.F<32\|64> | {<Sd\|Dd>,} <Sn\|Dn>、<Sm\|Dm> | 浮動小数点減算 | - | 168ページの3.11.30 |
| WFE | – | イベント待機 | - | 178ページの3.12.11 |
| WFI | – | 割込み待機 | - | 179ページの3.12.12 |

### 3.1.1 他の Cortex プロセッサとのバイナリ互換性

このプロセッサは、Armv7-M 命令セットおよび Armv7-M アーキテクチャ・プロファイルによって提供される機能を実装し、他の Cortex®-M プロファイル・プロセッサに実装されている命令セットおよび機能とバイナリ互換です。Cortex®-M7 プロセッサから次のプロセッサへのソフトウェアの移行はできません。

- Cortex®-M3 プロセッサ（浮動小数点演算または DSP 拡張が含まれている場合）。
- Cortex®-M4 プロセッサ（倍精度浮動小数点演算が含まれている場合）。
- Cortex®-M0 または Cortex®-M0+ プロセッサ（Armv6-M アーキテクチャの実装のため）。

他の Cortex®-M プロセッサ用に設計されたコードは、ビットバンディングに依存しない限り Cortex®-M7 と互換性があります。

円滑な移行を保証するために、Arm 社では、他の Cortex®-M プロファイル・プロセッサ・アーキテクチャで動作するように設計されたコードが次のルールに従い、Configuration and Control Register（設定および制御レジスタ）（CCR）を適切に設定するよう推奨しています。

- ワード転送を、NVIC と System Control Space（システム制御空間）（SCS）のレジスタへのアクセスのみに使用する。
- プロセッサ上で使用されていない SCS レジスタとレジスタ・フィールドはすべて、変更禁止として扱う。
- CCR レジスタの次のフィールドを設定する。
  - STKALIGN ビットを 1 にセット。
  - UNALIGN_TRP ビットを 1 にセット。
  - CCR レジスタのその他すべてのビットは元の値のままにしておく。

## 3.2 CMSIS 関数

ISO/IEC C コードは、一部の Cortex®-M7 プロセッサ命令には直接アクセスできません。このセクションでは、これらの命令を生成できる組込み関数について説明します。これらは、CMSIS から提供されますが、C コンパイラから提供される場合もあります。C コンパイラが適切な組込み関数をサポートしていない場合は、インライン·アセンブラを使用して一部の命令にアクセスする必要があることがあります。

CMSIS には、ISO/IEC C コードでは直接アクセスできない命令を生成するため、次の組み込み関数が用意されています。

**表 24. 一部の Cortex®-M7 プロセッサ命令を生成するための CMSIS 関数**

| 命令 | CMSIS 関数 |
|---|---|
| CPSIE I | `void __enable_irq(void)` |
| CPSID I | `void __disable_irq(void)` |
| CPSIE F | `void __enable_fault_irq(void)` |
| CPSID F | `void __disable_fault_irq(void)` |
| ISB | `void __ISB(void)` |
| DSB | `void __DSB(void)` |
| DMB | `void __DMB(void)` |
| REV | `uint32_t __REV(uint32_t int value)` |
| REV16 | `uint32_t __REV16(uint32_t int value)` |
| REVSH | `uint32_t __REVSH(uint32_t int value)` |
| RBIT | `uint32_t __RBIT(uint32_t int value)` |
| SEV | `void __SEV(void)` |
| WFE | `void __WFE(void)` |
| WFI | `void __WFI(void)` |

CMSIS では、MRS および MSR 命令を使用して特殊レジスタにアクセスするための関数も次のように多数用意しています。

**表 25. 特殊レジスタにアクセスするための CMSIS 関数**

| 特殊レジスタ | アクセス | CMSIS 関数 |
|---|---|---|
| PRIMASK | 読出し | `uint32_t __get_PRIMASK (void)` |
| | 書込み | `void __set_PRIMASK (uint32_t value)` |
| FAULTMASK | 読出し | `uint32_t __get_FAULTMASK (void)` |
| | 書込み | `void __set_FAULTMASK (uint32_t value)` |
| BASEPRI | 読出し | `uint32_t __get_BASEPRI (void)` |
| | 書込み | `void __set_BASEPRI (uint32_t value)` |
| CONTROL | 読出し | `uint32_t __get_CONTROL (void)` |
| | 書込み | `void __set_CONTROL (uint32_t value)` |
| MSP | 読出し | `uint32_t __get_MSP (void)` |
| | 書込み | `void __set_MSP (uint32_t TopOfMainStack)` |
| PSP | 読出し | `uint32_t __get_PSP (void)` |
| | 書込み | `void __set_PSP (uint32_t TopOfProcStack)` |

## 3.3 命令の説明について

以下に示すセクションでは、命令の使用方法について説明します。

- 62 ページのオペランドを参照してください。
- 63 ページのPC または SP を使用した場合の制限事項を参照してください。
- 63 ページのフレキシブル第 2 オペランドを参照してください。
- 64 ページのシフト演算を参照してください。
- 67 ページのアドレスのアライメントを参照してください。
- 67 ページのPC 相対式を参照してください。
- 68 ページの条件付き実行を参照してください。
- 70 ページの命令の幅の選択を参照してください。

### 3.3.1 オペランド

命令のオペランドには、Arm レジスタ、定数、またはその他の命令固有パラメータを指定できます。命令はオペランドに作用し、多くの場合、その結果をデスティネーション·レジスタに格納します。その命令にデスティネーション·レジスタが存在する場合、これは通常オペランドの前に指定されます。

一部の命令のオペランドは柔軟性が高く、レジスタにすることも、定数にすることもできます。63 ページのフレキシブル第 2 オペランドを参照してください。

### 3.3.2 PC または SP を使用した場合の制限事項

多くの命令には、オペランドまたはデスティネーション・レジスタにProgram Counter（プログラム・カウンタ）（PC）とStack Pointer（スタック・ポインタ）（SP）のどちらを使用できるかに関しての制約があります。詳細については、命令の説明を参照してください。

注： **BX、BLX、LDM、LDR、または POP の命令では、正常に実行するためには、PC に書き込まれるすべてのアドレスのビット [0] を 1 にする必要があります。これは、このビットが必要な命令セットを示しており、Cortex®-M7 プロセッサでは Thumb 命令のみがサポートされているためです。**

### 3.3.3 フレキシブル第 2 オペランド

多くの汎用データ処理命令にはフレキシブル第 2 オペランドがあります。これは、各命令の構文の説明において、*Operand2* と表記されています。

*Operand2* は次のいずれかです。

- 定数を参照してください。
- 64 ページのレジスタと任意に指定できるシフトを参照してください。

#### 定数

Operand2 定数は次の形式で指定します。

*#constant*

ここで *constant* は次のものが可能です。

- 32 ビットのワード内で 8 ビットの値を左に任意のビット数シフトして得られる定数
- 0x00XY00XY 形式の任意の定数。
- 0xXY00XY00 形式の任意の定数。
- 0xXYXYXYXY 形式の任意の定数。

注： **上記の定数の X と Y は 16 進数の値です。**

また、少数の命令では、*constant* により広い範囲の値を取ることができます。詳細については、各命令の説明を参照して下さい。

MOVS、MVNS、ANDS、ORRS、ORNS、EORS、BICS、TEQ、または TST 命令で Operand2 定数を使用するときに、その定数が 255 よりも大きく、8 ビット値のシフトによって得られる場合は、キャリー・フラグは定数のビット [31] に更新されます。Operand2 がそれ以外の定数の場合、これらの命令はキャリー・フラグに影響しません。

#### 命令置換

許可されていない constant が指定される場合、アセンブラは等価な命令を生成できることがあります。たとえば、CMP *Rd*, #0xFFFFFFFE という命令を CMN *Rd*, #0x2 という等価の命令としてアセンブルすることがあります。

### レジスタと任意に指定できるシフト

Operand2 レジスタは次の形式で指定します。

*Rm {, shift}*

ここで、

*Rm* これは、第 2 オペランドのデータを保持するレジスタです。

*shift* これは*Rm* に適用されるオプションのシフト量です。これは、次のいずれかです。

- ASR #*n* *n* ビット算術右シフト、1 ≤ *n* ≤ 32。
- LSL #*n* *n* ビット論理左シフト、1 ≤ *n* ≤ 31。
- LSR #*n* *n* ビット論理右シフト、1 ≤ *n* ≤ 32.。
- ROR #*n* *n* ビット右ローテート、1 ≤ *n* ≤ 31。
- RRX 拡張付き 1 ビット右ローテート。
- – 省略した場合、シフトは行われず、LSL #0 を指定した場合と同等になります。

シフトを省略するか、LSL #0 を指定した場合、*Rm* の値が命令で使用されます。

シフトを指定した場合、*Rm* の値に適用され、結果の 32 ビット値が命令で使用されます。ただし、レジスタ *Rm* の値は変更されません。また、シフトとともにレジスタを指定した場合は、特定の命令での使用時にキャリー・フラグが更新されます。シフト演算について、およびそれらがキャリー・フラグに与える影響については、シフト演算を参照してください。

## 3.3.4 シフト演算

レジスタのシフト演算では、レジスタ内のビットが指定のビット数（シフト長）だけ左または右に転送します。レジスタのシフトは以下の方法で実行できます。

- ASR、LSR、LSL、ROR、および RRX 命令によって直接実行し、結果を転送先レジスタに書き込みます。
- シフトを実行するレジスタとして第 2 オペランドを指定する命令によって、*Operand2* の計算中に実行できます（63 ページのフレキシブル第 2 オペランドを参照）。結果は命令によって使用されます。

指定可能なシフト長はシフトの種類と命令によって異なります（各命令の説明または 63 ページのフレキシブル第 2 オペランドを参照）。シフト長が 0 の場合、シフトは行われません。レジスタのシフト演算では、指定したシフト長が 0 の場合を除き、キャリー・フラグが更新されます。以下のサブセクションでは、さまざまなシフト演算と、それらの演算がキャリー・フラグに与える影響について説明します。ここでは、Rm はシフトされる値を保持するレジスタを、n はシフト長を表します。

### ASR

*n* ビットの算術右シフトです。レジスタ *Rm* の左側 32-*n* ビットを右に *n* 桁転送し、結果の右側 32-*n* ビットが得られます。さらに、レジスタの元のビット [31] は結果の左側 *n* ビットにコピーされます。65 ページの図 12 を参照してください。

ASR *#n* 演算を使用すると、レジスタ *Rm* の値を $2^n$ で除算し、その結果を負の無限大に丸めることができます。

命令が ASRS の場合、あるいは MOVS、MVNS、ANDS、ORRS、ORNS、EORS、BICS、TEQ、または TST 命令とともに ASR *#n* を *Operand2* で使用した場合、キャリー・フラグは、レジスタ *Rm* からシフトアウトされた最後のビット（ビット [*n*-1]）に更新されます。

- *n* が 32 以上の場合、結果のすべてのビットが *Rm* のビット [31] の値にセットされます。
- *n* が 32 以上でキャリー・フラグが更新される場合、*Rm* のビット [31] の値に更新されます。

図 12. ASR


## LSR

*n* ビット論理右シフトです。レジスタ *Rm* の左側 32-*n* ビットを右に *n* 桁転送し、結果の右側 32-*n* ビットが得られます。さらに、結果の左 *n* ビットを 0 にセットします図 13 を参照してください。

値が符号なしの整数と解釈される場合、LSR *#n* 演算を使用して、レジスタ *Rm* の値を $2^n$ で除算することができます。

命令が LSRS の場合、あるいは MOVS、MVNS、ANDS、ORRS、ORNS、EORS、BICS、TEQ、または TST 命令とともに LSR *#n* を *Operand2* で使用した場合、キャリー･フラグは、レジスタ *Rm* からシフトアウトされた最後のビット（ビット [*n*-1]）に更新されます。

- *n* が 32 以上の場合、結果のすべてのビットが 0 にクリアされます。
- *n* が 33 以上でキャリー･フラグが更新される場合、0 に更新されます。

図 13. LSR


## LSL

*n* ビットの論理左シフトです。レジスタ *Rm* の右側 32-*n* ビットを左に *n* 桁転送し、結果の左側 32-*n* ビットが得られます。さらに、結果の右側 *n* ビットを 0 にセットします 66 ページの図 14 を参照してください。

値が符号なしの整数または 2 の補数となる符号付き整数と解釈される場合、LSL *#n* 演算を使用して、レジスタ *Rm* の値を $2^n$ 倍することができます。このとき警告なしでオーバーフローが発生する場合があります。

命令が LSLS の場合、あるいは MOVS、MVNS、ANDS、ORRS、ORNS、EORS、BICS、TEQ、または TST 命令とともに LSL *#n*（*n* はゼロではない）を *Operand2* で使用した場合、キャリー･フラグは、レジスタ *Rm* からシフトアウトされた最後のビット（ビット [32-*n*]）に更新されます。これらの命令を LSL #0 とともに使用した場合、キャリー･フラグへの影響はありません。

- *n* が 32 以上の場合、結果のすべてのビットが 0 にクリアされます。
- *n* が 33 以上でキャリー･フラグが更新される場合、0 に更新されます。

図 14. LSL

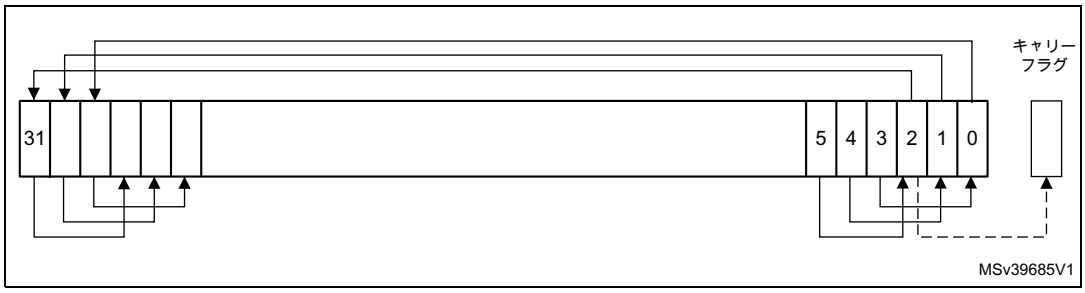

## ROR

*n* ビットの右ローテートです。レジスタ *Rm* の左側 32-*n* ビットを右に *n* 桁転送し、結果の右側 32-*n* ビットが得られます。さらに、レジスタの右 *n* ビットを結果の左 *n* ビットに転送します図 15 を参照してください。

命令が RORS の場合、あるいは MOVS、MVNS、ANDS、ORRS、ORNS、EORS、BICS、TEQ、または TST 命令とともに ROR *#n* を *Operand2* で使用した場合、キャリー･フラグは、レジスタ *Rm* からローテーションされた最後のビット（ビット [*n*-1]）に更新されます。

- *n* が 32 の場合、結果の値は *Rm* の値と同じになります。また、キャリー･フラグが更新される場合は、*Rm* のビット [31] に更新されます。
- 32 を超えるシフト長 *n* と ROR を指定した場合、シフト長 *n*-32 と ROR を指定した場合と動作は同じになります。

図 15. ROR

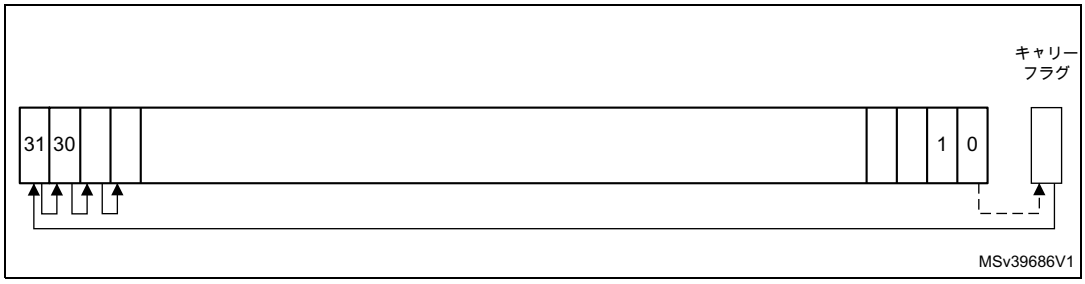



## RRX

拡張付き右ローテートです。レジスタ *Rm* のビットを 1 ビット分右に転送します。さらに、キャリー･フラグを結果のビット [31] にコピーします。67 ページの図 16 を参照してください。

命令が RRXS の場合、あるいは MOVS、MVNS、ANDS、ORRS、ORNS、EORS、BICS、TEQ、または TST 命令とともに RRX を *Operand2* で使用した場合、キャリー･フラグは、レジスタ *Rm* のビット [0] に更新されます。

図 16. RRX




### 3.3.5 アドレスのアライメント

アラインド･アクセスとは、ワード境界で整列されたアドレスがワード、デュアルワード、またはマルチワード･アクセスに使用される、あるいはハーフワード境界で整列されたアドレスがハーフワード･アクセスに使用される操作です。バイト･アクセスは常にアラインされます。

Cortex®-M7 プロセッサでは、次の命令でのみ、アンアラインド･アクセスがサポートされます。

- LDR、LDRT
- LDRH、LDRHT
- LDRSH、LDRSHT
- STR、STRT
- STRH、STRHT

その他すべてのロードおよびストア命令では、アンアラインド･アクセスを実行すると、用法フォールト例外が生成されるため、アラインされたアドレスでアクセスする必要があります。用法フォールトの詳細については、46 ページのフォールト処理を参照してください。

アンアラインド･アクセスは通常アラインド･アクセスより遅くなります。また、一部のメモリ領域では、アンアラインド･アクセスをサポートしていない場合があります。したがって、Arm 社では、プログラミングの際にアラインド･アクセスであることを確認することを推奨しています。予期せずにアンアラインド･アクセスが生成されるのをトラップするには、設定で UNALIGN_TRP ビットを使用し、レジスタを制御します（197 ページの設定および制御レジスタを参照）。

### 3.3.6 PC 相対式

PC 相対式、すなわちラベルは、命令またはリテラル･データのアドレスを表すシンボルです。命令内では、PC 値に対して数値オフセットを加算または減算した値として表現されます。アセンブラでは、ラベルおよび現在の命令のアドレスから必要なオフセットを計算します。オフセットが大きすぎる場合、アセンブラによってエラーが生成されます。

- B、BL、CBNZ、および CBZ 命令では、PC の値は現在の命令のアドレスに 4 バイトを加算した値になります。
- ラベルを使用するその他すべての命令では、PC の値は現在の命令のアドレスに 4 バイトを加算し、ワード境界で整列させるために、その結果のビット [1] を 0 にクリアした値になります。
- アセンブラでは、ラベルにある数値を加算または減算したものや、[PC, #number] の形式の表現など、他の構文が PC 相対式として許容される場合があります。

### 3.3.7 条件付き実行

ほとんどのデータ処理命令では、必要に応じ、操作の結果に従って、*Application Program Status register*（アプリケーション･プログラム･ステータス･レジスタ）（APSR）の条件フラグを更新できます（23 ページのアプリケーション･プログラム･ステータス･レジスタを参照）。すべてのフラグを更新する命令もあれば、サブセットのみを更新する命令もあります。フラグが更新されない場合は、元の値が保持されます。命令の影響を受けるフラグについては、該当する命令の説明を参照してください。

別の命令によってセットされた条件フラグに基づいて、以下のいずれかの時点で命令を条件付きで実行できます。

- フラグを更新した命令の直後。
- フラグを更新していない任意の数の命令の後。

条件付き実行は、条件分岐を使用するか、命令に条件コードの接尾文字を追加することによって可能です。条件付き命令を作成するために命令に追加する接尾文字のリストは、69 ページの表 26 を参照してください。条件コードの接尾文字を使用すると、プロセッサはフラグに基づいて条件をテストすることができます。条件付き命令の条件テストが失敗すると、命令は次のようになります。

- 実行されない。
- デスティネーション･レジスタに値を書き込まない。
- フラグに影響を及ぼさない。
- 例外を生成しない。

条件分岐を除く条件付き命令は、If-then 命令ブロックの内側に配置する必要があります。詳細および IT 命令使用時の制限事項については、145 ページのIT を参照してください。ベンダによっては、IT ブロックの外側に条件付き命令がある場合、アセンブラによって自動的に IT 命令が挿入されることもあります。

CBZ および CBNZ 命令を使用して、レジスタの値をゼロと比較し、その結果に基づいて分岐することもできます。

このセクションでは以下の内容について説明します。

- 68 ページの条件フラグを参照してください。
- 69 ページの条件コードの接尾文字を参照してください。

#### 条件フラグ

APSR には次の条件フラグが含まれます。

**N** 演算結果が負の場合は、1 にセットされます。それ以外の場合は、0 にクリアされます。

**Z** 演算結果がゼロの場合は、1 にセットされます。それ以外の場合は、0 にクリアされます。

**C** 演算の結果としてキャリーが発生した場合は、1 にセットされます。それ以外の場合は、0 にクリアされます。

**V** 演算がオーバーフローを引き起こした場合は、1 にセットされます。それ以外の場合は、0 にクリアされます。

APSR の詳細については、22 ページのプログラム･ステータス･レジスタを参照してください。

C 条件フラグは、次の 4 つの方法のいずれかでセットされます。

- 比較命令 CMN を含む加算では、加算によってキャリー（すなわち、符号なしオーバーフロー）が発生した場合、C は 1 にセットされ、それ以外の場合は 0 にセットされます。
- 比較命令 CMP を含む減算の場合、減算によってボロー（すなわち、符号なしアンダーフロー）が発生した場合、C は 0 にセットされ、それ以外の場合は 1 にセットされます。
- シフト演算が組み込まれた加算以外、または減算の場合、C は、シフタによって値からシフトアウトされた最後のビットにセットされます。

- 他の加算以外、または減算の場合、通常、C は変更されません。特殊なケースについては、個別の命令の説明を参照してください。

オーバーフローは、ビット [31] の結果の符号が、無限精度で演算が実行された場合の結果の符号と一致しない場合に発生します。以下に例を示します。

- 2 つの負の値の加算結果が正の値になる場合。
- 2 つの正の値の加算結果が負の値になる場合。
- 負の値から正の値を減算した結果が正の値になる場合。
- 正の値から負の値を減算した結果が負の値になる場合。

比較演算は、結果が破棄されることを除き、CMP の場合は減算と、CMN の場合は加算と同じです。詳細については、該当する命令の説明を参照してください。

ほとんどの命令では、ステータス·フラグは、S の接尾文字が指定されている場合にのみ更新されます。詳細については、該当する命令の説明を参照してください。

## 条件コードの接尾文字

条件付きにできる命令には、任意で条件コードを指定できます。構文の説明では、条件コードを {*cond*} と表記しています。条件付き実行では、先行する IT 命令が必要です。条件コードが指定されている命令は、APSR の条件コード·フラグが指定した条件を満たしている場合にのみ実行されます。表 26 に、使用できる条件コードを示します。

IT 命令とともに条件付き実行を使用して、コードの分岐命令の数を減らすことができます。

表 26 では、条件コードの接尾文字と、N、Z、C、および V の各フラグとの関係も示します。

**表 26. 条件コードの接尾文字**

| 接尾文字 | フラグ | 意味 |
|---|---|---|
| EQ | Z = 1 | 等しい |
| NE | Z = 0 | 等しくない |
| CS または HS | C = 1 | 以上（符号なし ≥） |
| CC または LO | C = 0 | 未満、符号なし |
| MI | N = 1 | ネガティブ |
| PL | N = 0 | 正または 0 |
| VS | V = 1 | オーバーフロー |
| VC | V = 0 | オーバーフローなし |
| HI | C = 1 かつ Z = 0 | より大きい、符号なし |
| LS | C = 0 または Z = 1 | 以下（符号なし） |
| GE | N = V | 以上（符号付き） |
| LT | N != V | より小さい、符号付き |
| GT | Z = 0 かつ N = V | より大きい、符号付き |
| LE | Z = 1 かつ N != V | 以下（符号付き） |
| AL | すべて | 無条件。接尾文字が指定されていない場合は、これがデフォルトです。 |

例 3-1：絶対値に、数の絶対値を求める条件付き命令の使用方法を示します。R0 = abs(R1).

**例 3-1：絶対値**

```
MOVS    R0, R1          ; R0 = R1、フラグをセット。
IT      MI              ; 値が 0 または正の場合、
                        ; 次の命令にスキップ。
RSBMI   R0, R0, #0      ; 負の場合、R0 = -R0。
```

例 3-2：値の比較と更新に、符号付き値 R0 が R1 より大きく R2 が R3.より大きい場合の、条件付き命令を使用したR4の値の更新の方法を示します。

**例 3-2：値の比較と更新**

```
CMP     R0, R1          ; R0 と R1 を比較してフラグをセット。
ITT     GT              ; GT 条件が満足されない限り、
                        ; 次の 2 つの命令をスキップ。
CMPGT   R2, R3          ; 「greater than（より大きい）」場合、
                        ; R2 と R3 を比較してフラグをセット。
MOVGT   R4, R5          ; まだ「greater than（より大きい）」場合、R4 = R5 を実行。
```

### 3.3.8 命令の幅の選択

指定したオペランドとデスティネーション･レジスタに応じて、16 ビット･エンコーディングと 32 ビット･エンコーディングのどちらかを生成できる命令が数多くあります。これらの命令の一部では、命令幅の接尾文字を使用することで、強制的に特定の命令サイズを指定できます。.W 接尾文字では、32 ビット命令のエンコーディングが強制されます。.N 接尾文字では、16 ビット命令のエンコーディングが強制されます。

命令の幅の接尾文字が指定されたが、要求された幅の命令エンコーディングを生成できない場合、アセンブラはエラーを生成します。

分岐命令のケースのようにオペランドが命令またはリテラル･データのラベルである場合など、場合によっては .W 接尾文字を指定する必要があることもあります。これは、アセンブラが適切なサイズのエンコーディングを自動的に生成しないことがあるためです。

命令の幅の接尾文字を使用するには、この文字を命令ニーモニックおよび条件コード（存在する場合）の直後に配置します。例 3-3：命令幅の選択に、命令の幅の接尾文字を使用した命令を示します。

**例 3-3：命令幅の選択**

```
BCS.W  label        ; 短い分岐に対しても
                    ; 32 ビット命令を生成。

ADDS.W R0, R0, R1 ; 16 ビット命令で同じ操作を行うことができますが、
                    ; 32 ビット命令を作成。
```

## 3.4 メモリ·アクセス命令

表 27 に、メモリ·アクセス命令を示します。

**表 27. メモリ·アクセス命令**

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| ADR | PC 相対アドレスの生成 | 72 ページのADR |
| CLREX | 排他をクリア | 83 ページのCLREX |
| LDM{mode} | レジスタの多重ロード | 78 ページのLDM と STM |
| LDR{type} | イミディエート·オフセットを使ったレジスタ·ロード | 72 ページのLDR と STR （イミディエート·オフセット） |
| LDR{type} | レジスタ·オフセットを使ったレジスタ·ロード | 75 ページのLDR と STR（レジスタ·オフセット） |
| LDR{type}T | 非特権アクセスでのレジスタ·ロード | 76 ページのLDR と STR （非特権） |
| LDR | PC 相対アドレスを使ったレジスタ·ロード | 77 ページのLDR （PC 相対） |
| LDRD | レジスタ·ロード（デュアル） | 72 ページのLDR と STR （イミディエート·オフセット） |
| LDREX{type} | 排他的レジスタ·ロード | 82 ページのLDREX と STREX |
| PLD | プリロード・データ | 80 ページのPLD |
| POP | レジスタをスタックからポップ | 81 ページのPUSH と POP |
| PUSH | レジスタをスタックにプッシュ | 81 ページのPUSH と POP |
| STM{mode} | レジスタの多重ストア | 78 ページのLDM と STM |
| STR{type} | イミディエート·オフセットを使ったレジスタ·ストア | 72 ページのLDR と STR （イミディエート·オフセット） |
| STR{type} | レジスタ·オフセットを使ったレジスタ·ストア | 75 ページのLDR と STR（レジスタ·オフセット） |
| STR{type}T | 非特権アクセスでのレジスタ·ストア | 76 ページのLDR と STR （非特権） |
| STREX{type} | 排他的レジスタ·ストア | 82 ページのLDREX と STREX |

### 3.4.1 ADR

PC 相対アドレスの生成。

#### 構文

```
ADR{cond} Rd, label
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション･レジスタです。

*label* これは PC 相対式です。67 ページのPC 相対式を参照してください。

#### 動作

ADR は、PC にイミディエート値を加算することによってアドレスを生成し、結果をデスティネーション・レジスタに書き込みます。

アドレスは PC 相対アドレスなので、ADR では位置に依存しないコードを生成できる方法を提供します。

ADR を使用して BX または BLX 命令のターゲット･アドレスを生成する場合、正常に実行するには、生成するアドレスのビット [0] を 1 にセットする必要があります。

*label* の値は、PC のアドレスから -4095 ~ +4095 の範囲内とする必要があります。

最大のオフセット範囲を得るため、またはワード境界で整列されていないアドレスを生成するためには、.W 接尾文字を使用する必要がある場合があります70 ページの命令の幅の選択を参照してください。

#### 制限事項

*Rd* に SP または PC は使用できません。

#### 条件フラグ

この命令によるフラグの変更はありません。

#### 例

```
    ADR     R1, TextMessage   ; TextMessage というラベルの位置の
                              ; アドレス値を R1 に書き込む。
```

### 3.4.2 LDR と STR （イミディエート･オフセット）

イミディエート･オフセット、プレインデクス･イミディエート･オフセット、またはポストインデクス･イミディエート･オフセットを使ったロードとストア。

#### 構文

```
op{type}{cond} Rt, [Rn {, #offset}]        ; イミディエート･オフセット
op{type}{cond} Rt, [Rn, #offset]!          ; プレインデクス
op{type}{cond} Rt, [Rn], #offset           ; ポストインデクス
opD{cond} Rt, Rt2, [Rn {, #offset}]        ; イミディエート･オフセット、ダブルワード
opD{cond} Rt, Rt2, [Rn, #offset]!          ; プレインデクス、ダブルワード
opD{cond} Rt, Rt2, [Rn], #offset           ; ポストインデクス、ダブルワード
```

ここで、

*op* これは次のいずれかです。
- LDR ロード・レジスタ。
- STR ストア・レジスタ。

タイプ これは次のいずれかです。
- B 符号なしバイト（ロード時に 32 ビットにゼロ拡張）
- SB 符号付きバイト（LDR のみ。32 ビットに符号拡張）
- H 符号なしハーフワード（ロード時に 32 ビットにゼロ拡張）
- SH 符号付きハーフワード（LDR のみ。32 ビットに符号拡張）
- - 省略（ワード）

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rt* これはロードまたはストアするレジスタです。

*Rn* これはメモリ·アドレスのベースとなるレジスタです。

オフセット これは *Rn* からのオフセットです。*offset* が省略されると、アドレスは *Rn* の内容となります。

*Rt2* これはダブルワードのロード／ストアで使用する追加のレジスタです。

### 動作

LDR 命令は、メモリの値を 1 つまたは 2 つのレジスタにロードします。

STR 命令は、1 つまたは 2 つのレジスタの値をメモリにストアします。

イミディエート·オフセットを使用するロードおよびストアの命令では、次のアドレッシング·モードを使用できます。

### オフセット·アドレッシング

レジスタ *Rn* から取得したアドレスにオフセット値が加算されるか、このアドレスからオフセット値が減算されます。この結果は、メモリ·アクセスのアドレスとして使用されます。レジスタ *Rn* は不変です。このモードでのアセンブリ言語構文は、

`[Rn, #offset]` です。

### プレインデクス·アドレッシング

レジスタ *Rn* から取得したアドレスにオフセット値が加算されるか、このアドレスからオフセット値が減算されます。この結果は、メモリ·アクセスのアドレスとして使用され、レジスタ *Rn* に書き戻されます。このモードでのアセンブリ言語構文は、

`[Rn, #offset]!` です。

### ポストインデクス·アドレッシング

レジスタ *Rn* から取得したアドレスが、メモリ·アクセスのアドレスとして使用されます。このアドレスにオフセット値が加算されるか、このアドレスからオフセット値が減算され、レジスタ *Rn* に書き戻されます。このモードでのアセンブリ言語構文は、

`[Rn], #offset` です

ロードまたはストアする値は、バイト、ハーフワード、ワード、またはダブルワードにできます。バイトおよびハーフワードは、符号付きでも符号なしでも構いません。67 ページのアドレスのアライメントを参照してください。

表 28 に、イミディエート、プレインデクス、ポストインデクスの各形式のオフセットの範囲を示します。

**表 28. オフセット範囲**

| 命令タイプ | イミディエート オフセット | プレインデクス | ポストインデクス |
|---|---|---|---|
| ワード、ハーフワード、符号付きハーフワード、バイト、または符号付きバイト | -255 ~ 4095 | -255 ~ 255 | -255 ~ 255 |
| ダブルワード | -1020 ~ 1020 の範囲の 4 の倍数 | -1020 ~ 1020 の範囲の 4 の倍数 | -1020 ~ 1020 の範囲の 4 の倍数 |

## 制限事項

ロード命令の場合：

- *Rt* は、ワードでのロードの場合のみ、SP または PC にできます。
- *Rt* は、ダブルワードでのロードの場合、*Rt2* とは異なる値である必要があります。
- *Rn* は、プレインデクスまたはポストインデクスの形式では、*Rt* および *Rt2* とは異なる値である必要があります。

ワードでのロード命令で *Rt* が PC の場合：

- 正常に実行するには、ロードされる値のビット [0] は 1 である必要があります。
- ロードされる値のビット [0] を 0 に変更することにより、生成されたアドレスに分岐が発生します。
- 命令が条件付きである場合は、その命令が IT ブロックの最後の命令である必要があります。

ストア命令の場合：

- *Rt* は、ワードでのストアの場合のみ、SP にできます。
- *Rt* は PC にはできません。
- *Rn* は PC にはできません。
- *Rn* は、プレインデクスまたはポストインデクスの形式では、*Rt* および *Rt2* とは異なる値である必要があります。

## 条件フラグ

これらの命令によるフラグの変更はありません。

## 例

```
LDR     R8, [R10]               ; R10 のアドレスから R8 をロード。
LDRNE   R2, [R5, #960]!         ; R5 のアドレスの 960 バイト上のワードから
                                ; R2 を条件付きでロードし、R5 を
                                ; 960 バイト分インクリメント。
STR     R2, [R9,#const-struc]   ; const-struc は、0 ~ 4095 の範囲の定数に
                                ; 評価される式です。
STRH    R3, [R4], #4            ; R3 をハーフワード･データとして
                                ; R4 のアドレスに保存し、
                                ; R4 を 4 だけインクリメント。
LDRD    R8, R9, [R3, #0x20]     ; R3 のアドレスから 32 バイト上のワードから
                                ; R8 をロードし、R3 のアドレスから
                                ; 36 バイト上のワードから R9 をロード。
STRD    R0, R1, [R8], #-16      ; R0 を R8 のアドレスに格納し、
```

```
                                    ; R1 を R8 のアドレスから
                                    ; 4 バイト上のワードに格納し、
                                    ; R8 を 16 デクリメントします。
```

### 3.4.3 LDR と STR （レジスタ･オフセット）

レジスタ･オフセットを使ったロードとストア。

#### 構文

*op*{*type*}{*cond*} *Rt*, [*Rn*, *Rm* {, LSL #*n*}]

ここで、

*op* これは次のいずれかです。
- LDR ロード・レジスタ。
- STR ストア・レジスタ。

*タイプ* これは次のいずれかです。
- B 符号なしバイト（ロード時に 32 ビットにゼロ拡張）
- SB 符号付きバイト（LDR のみ。32 ビットに符号拡張）
- H 符号なしハーフワード（ロード時に 32 ビットにゼロ拡張）
- SH 符号付きハーフワード（LDR のみ。32 ビットに符号拡張）
- – 省略（ワード）

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rt* これはロードまたはストアするレジスタです。

*Rn* これはメモリ･アドレスのベースとなるレジスタです。

*Rm* これはオフセットとして使用される値を含むレジスタです。

LSL #*n* これは *n* （範囲は 0 ~ 3）ビットの任意のシフトです。

#### 動作

LDR 命令は、メモリの値をレジスタにロードします。

STR 命令は、レジスタの値をメモリにストアします。

ロード元またはストア先のメモリ･アドレスは、レジスタ *Rn* からのオフセット位置です。オフセットは *Rm* レジスタによって指定され、LSL を使用して最大 3 ビットまで左にシフトできます。

ロードまたはストアする値は、バイト、ハーフワード、またはワードにできます。ロード命令では、バイトおよびハーフワードは、符号付きでも符号なしでも構いません。67 ページのアドレスのアライメントを参照してください。

#### 制限事項

これらの命令の場合：

- *Rn* は PC にはできません。
- *Rm* に SP または PC は指定できません。
- *Rt* は、ワードでのロードおよびワードでのストアの場合のみ、SP にできます。
- *Rt* は、ワードでのロードの場合のみ、PC にできます。

ワードでのロード命令で *Rt* が PC の場合：

- 正常に実行するには、ロードされる値のビット [0] は 1 である必要があり、このハーフワード境界で整列されたアドレスには分岐が発生します。
- 命令が条件付きである場合は、その命令が IT ブロックの最後の命令である必要があります。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
STR     R0, [R5, R1]          ; R0 の値を R5 と R1 の合計に等しいアドレスに
                              ; 格納する。
LDRSB   R0, [R5, R1, LSL #1] ; R5 と R1 の 2 倍の合計に等しいアドレスからの
                              ; バイト値を読み出す。符号はワード値に拡張して
                              ; R0 に入れる。
STR     R0, [R1, R2, LSL #2] ; R0 を R1 と R2 の 4 倍の合計に
                              ; 等しいアドレスに格納する。
```

## 3.4.4 LDR と STR （非特権）

非特権アクセスを使ったロードとストア。

### 構文

```
op{type}T{cond} Rt, [Rn {, #offset}]       ; イミディエート・オフセット
```

ここで、

*op* これは次のいずれかです。
- LDR ロード・レジスタ。
- STR ストア・レジスタ。

タイプ これは次のいずれかです。
- B 符号なしバイト（ロード時に 32 ビットにゼロ拡張）
- SB 符号付きバイト（LDR のみ。32 ビットに符号拡張）
- H 符号なしハーフワード（ロード時に 32 ビットにゼロ拡張）
- SH 符号付きハーフワード（LDR のみ。32 ビットに符号拡張）
- – 省略（ワード）

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rt* これはロードまたはストアするレジスタです。

*Rn* これはメモリ・アドレスのベースとなるレジスタです。

オフセット これは *Rn* からのオフセットで、0 ~ 255 の値を指定できます。*offset* が省略されると、アドレスは *Rn* の値となります。

### 動作

このロードとストアの命令は、イミディエート・オフセットを使用したメモリ・アクセス命令（72 ページのLDR と STR（イミディエート・オフセット）を参照）と同じ機能を実行します。異なる点は、特権のあるソフトウェアで使用している場合でも、この命令では非特権アクセスしかできないということです。

特権のないソフトウェアで使用する場合、この命令はイミディエート・オフセットを使用した通常のメモリ・アクセス命令とまったく同じ動作となります。

### 制限事項

これらの命令の場合：

- *R*n は PC にはできません。
- *Rt* に SP または PC は使用できません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
STRBTEQ  R4, [R7]        ;非特権アクセスで、R4 の最下位バイトを条件付きで
                         ;R7 のアドレスに格納します。
LDRHT    R2, [R2, #8]    ;非特権アクセスで、R2 と 8 の合計に等しいアドレスの
                         ; ハーフワード値を R2 にロードします。
```

## 3.4.5 LDR （PC 相対）

メモリからのレジスタ·ロード。

### 構文

```
LDR{type}{cond} Rt, label
LDRD{cond} Rt, Rt2, label          ; ダブルワードをロード
```

ここで、

タイプ　これは次のいずれかです。

| | |
|---|---|
| B | 符号なしバイト（32 ビットにゼロ拡張） |
| SB | 符号付きバイト（32 ビットに符号拡張） |
| H | 符号なしハーフワード（32 ビットにゼロ拡張） |
| SH | 符号付きハーフワード（32 ビットに符号拡張） |
| - | 省略（ワード） |

*cond*　これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rt*　これはロードまたはストアするレジスタです。

*Rt2*　これは 2 番目にロードまたはストアするレジスタです。

*label*　これは PC 相対式です。67 ページのPC 相対式を参照してください。

### 動作

LDR は、PC 相対メモリ·アドレスの値をレジスタにロードします。メモリ·アドレスは、ラベルまたは PC からのオフセットによって指定されます。

ロードまたはストアする値は、バイト、ハーフワード、またはワードにできます。ロード命令では、バイトおよびハーフワードは、符号付きでも符号なしでも構いません。67 ページのアドレスのアライメントを参照してください。

label は現在の命令の限定された範囲内である必要があります。表 29 に *label* と PC 間で可能なオフセットを示します

表 29. オフセット範囲

| 命令タイプ | オフセット範囲 |
|---|---|
| ワード、ハーフワード、符号付きハーフワード、バイト、符号付きバイト | –4095 ~ 4095 |
| ダブルワード | –1020 ~ 1020 |

注： **最大のオフセット範囲を得るため、.W 接尾文字を使用する必要がある場合があります。70 ページの命令の幅の選択を参照してください。**

### 制限事項

これらの命令の場合：

- *Rt* は、ワードでのロードの場合のみ、SP または PC にできます。
- *Rt2* に SP または PC は使用できません。
- *Rt* は *Rt2* とは異なる値である必要があります。

ワードでのロード命令で *Rt* が PC の場合：

- 正常に実行するには、ロードされる値のビット [0] は 1 である必要があり、このハーフワード境界で整列されたアドレスには分岐が発生します。
- 命令が条件付きである場合は、その命令が IT ブロックの最後の命令である必要があります。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
LDR     R0, LookUpTable    ; LookUpTable とラベル付けされたアドレスからの
                           ; ワード·データで R0 をロード。
LDRSB   R7, localdata      ; localdata というラベルのアドレスからバイト値を
                           ; ロードし、ワード値に符号拡張して、R7 に格納します。
```

## 3.4.6 LDM と STM

複数レジスタのロードとストア。

### 構文

*op*{*addr_mode*}{*cond*} *Rn*{!}, *reglist*

ここで、

*op* これは次のいずれかです。
- LDM レジスタの多重ロード。
- STM レジスタの多重ストア。

*addr_mode* これは次のいずれかです。
- IA 各アクセスの後にアドレスをインクリメントします。これがデフォルトです。
- DB 各アクセスの前にアドレスをデクリメントします。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rn* これはメモリ·アドレスのベースとなるレジスタです。

`!` これは任意に指定できるライトバック接尾文字です。! を指定すると、ロード元またはストア先の最終アドレスが *Rn* にライトバックされます。

*reglist* これは、ロードまたはストアするレジスタのリストを中かっこで囲んで指定します。レジスタ範囲も指定できます。複数のレジスタまたはレジスタ範囲を指定する場合は、コンマで区切る必要があります（80 ページの例を参照）。

LDM と LDMFD は LDMIA の同義語です。LDMFD を使用することで、完全降順スタックからデータをポップできます。

LDMEA は LDMDB の同義語で、これを使用することで、空き昇順スタックからデータをポップできます。

STM と STMEA は STMIA の同義語です。STMEA を使用することで、空き昇順スタックへデータをプッシュできます。

STMFD は STMDB の同義語で、これを使用することで、完全降順スタックへデータをプッシュできます。

### 動作

LDM 命令は、*Rn* に基づくメモリ·アドレスからワード値を *reglist* のレジスタにロードします。

STM 命令は、*reglist* のレジスタのワード値を Rn に基づくメモリ·アドレスにストアします。

LDM、LDMIA、LDMFD、STM、STMIA、および STMEA の場合、アクセスに使用されるメモリ·アドレスは、*Rn* ~ *Rn* + 4 * (*n*-1) の範囲（*n* は *reglist* 内のレジスタ数）で 4 バイト間隔となります。アクセスはレジスタ番号の昇順に発生し、最小の番号のレジスタが最下位のメモリ·アドレスを使用し、最大の番号のレジスタが最上位のメモリ·アドレスを使用します。ライトバック接尾文字が指定されている場合、*Rn* + 4 * (*n*-1) の値が *Rn* にライトバックされます。

LDMDB、LDMEA、STMDB、および STMFD の場合、アクセスに使用されるメモリ·アドレスは、*Rn* ~ *Rn* - 4 * (*n*-1) の範囲（*n* は *reglist* 内のレジスタ数）で 4 バイト間隔となります。アクセスはレジスタ番号の降順に発生し、最大の番号のレジスタが最上位のメモリ·アドレスを使用し、最小の番号のレジスタが最下位のメモリ·アドレスを使用します。ライトバック接尾文字を指定した場合、*Rn* - 4 * (*n*-1) の値は *Rn* に書き戻されます。.

PUSH 命令と POP 命令は、この形式で表すことができます。詳細については、81 ページのPUSH と POP を参照してください。

### 制限事項

これらの命令の場合：

- *R*n は PC にはできません。
- *reglist* に SP を含めることはできません。
- STM 命令では、*reglist* に PC を含めることはできません。
- LDM 命令では、*reglist* に LR が含まれる場合、ここに PC を含めることはできません。
- ライトバック接尾文字が指定された場合、*reglist* は *Rn* を含むことはできません。

LDM 命令で *reglist* に PC が含まれている場合：

- 正常に実行するには、PC にロードされる値のビット [0] は 1 である必要があり、このハーフワード境界で整列されたアドレスには分岐が発生します。
- 命令が条件付きである場合は、その命令が IT ブロックの最後の命令である必要があります。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
LDM     R8,{R0,R2,R9}       ; LDMIA は LDM の同義語。
STMDB   R1!,{R3-R6,R11,R12}
```

### 誤用例

```
STM     R5!,{R5,R4,R9} ; R5 にストアされる値は予測不能。
LDM     R2, {}         ; リストには少なくとも 1 つのレジスタが必要。
```

## 3.4.7 PLD

プリロード・データ

### 構文

```
PLD [<Rn>,#<imm12>]

PLD  [<Rn>, <Rm> {, LSL #<shift>}]

PLD  <label>
```

ここで、

`<Rn>` これはベース・レジスタです。

`<imm>` これはアドレス形成に使用するイミディエート・オフセットです。

`<Rm>` オプションでシフトされるオフセットレジスタです。

`<shift>` `<Rm>`（0 から 3 の範囲）から読み出された値に適用するシフトを指定します。このオプションを省略した場合は、シフトは 0 とみなされます。

`<label>` 近い将来アクセスされる可能性のあるリテラル項目のラベル。

### 動作

PLD が指定されたアドレスからのデータ・メモリへのアクセスが近いうちに行われる可能性が高いことを、メモリ・システムに通知します。アドレスがキャッシュ可能な場合、メモリ・システムは指定されたアドレスを含むキャッシュラインをデータ・キャッシュにプリロードすることで応答します。アドレスがキャッシュ不可またはデータ・キャッシュが無効の場合、この命令は無操作として動作します。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.4.8 PUSH と POP

完全降順スタックへレジスタをプッシュ、および完全降順スタックからレジスタをポップ。

#### 構文

`PUSH{cond} reglist`

`POP{cond} reglist`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*reglist* これは空でないレジスタのリストを、中かっこで囲んで指定します。レジスタ範囲も指定できます。複数のレジスタまたはレジスタ範囲を指定する場合は、コンマで区切る必要があります。

PUSH と POP は、アクセスに使用されるメモリ·アドレスが SP に基づき、アクセスに使用される最終アドレスが SP にライトバックされる STMDB と LDM （またはLDMIA）と同義です。このような場合、推奨されるニーモニックは PUSH と POP です。

#### 動作

PUSH は、最小の番号のレジスタが最下位のメモリ·アドレスを使用して、最大の番号のレジスタが最上位のメモリ·アドレスを使用してレジスタをスタックに格納します。

POP は、最小の番号のレジスタが最下位のメモリ·アドレスを使用して、最大の番号のレジスタが最上位のメモリ·アドレスを使用してレジスタをスタックからロードします。

完全降順スタックを実装し、PUSH は最上位のメモリ·アドレスとして SP レジスタの値から 4 を引いた値を使用して、POP は最下位のメモリ·アドレスとして SP レジスタの値を使用します。完了時に、PUSH は最下位のストア値の位置をポイントするように SP レジスタを更新し、POP はロードされた最上位の位置の上の位置をポイントするように SP レジスタを更新します。

POP 命令で reglist に PC が含まれている場合、この POP 命令の完了時にこの位置への分岐を発生させます。PC で読み出した値のビット [0] は、APSR の T ビットの更新に使用されます。正常な操作を保証するには、このビットを 1 にする必要があります。

詳細については、78 ページのLDM と STM を参照してください。

#### 制限事項

これらの命令の場合：

- *reglist* に SP を含めることはできません。
- PUSH 命令では、*reglist* に PC を含めることはできません。
- POP 命令では、*reglist* に LR が含まれる場合、ここに PC を含めることはできません。

POP 命令で *reglist* に PC が含まれている場合：

- 正常に実行するには、PC にロードされる値のビット [0] は 1 である必要があり、このハーフワード境界で整列されたアドレスには分岐が発生します。
- 命令が条件付きである場合は、その命令が IT ブロックの最後の命令である必要があります。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
PUSH {R0,R4-R7} ; R0、R4、R5、R6、R7 をスタックにプッシュする
PUSH {R2,LR}    ; R2 とリンクレジスタをスタックにプッシュする
POP  {R0,R6,PC} ; R0、R6 および PC をスタックからポップし、新しい PC に分岐する
```

## 3.4.9 LDREX と STREX

排他的なレジスタのロードとストア。

### 構文

`LDREX{cond} Rt, [Rn {, #offset}]`

`STREX{cond} Rd, Rt, [Rn {, #offset}]`

`LDREXB{cond} Rt, [Rn]`

`STREXB{cond} Rd, Rt, [Rn]`

`LDREXH{cond} Rt, [Rn]`

`STREXH{cond} Rd, Rt, [Rn]`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これは返されるステータスのデスティネーション・レジスタです。

*Rt* これはロードまたはストアするレジスタです。

*Rn* これはメモリ・アドレスのベースとなるレジスタです。

オフセット これは *Rn* の値に適用される任意のオフセットです。*offset* が省略されると、アドレスは *Rn* の値となります。

### 動作

LDREX、LDREXB、および LDREXH は、メモリ・アドレスからそれぞれワード、バイト、およびハーフワードをロードします。

STREX、STREXB、および STREXH は、メモリ・アドレスにそれぞれワード、バイト、およびハーフワードをストアします。排他的ストア命令で使用されるアドレスは、最近実行された排他的ロード命令のアドレスと同じである必要があります。排他的ストア命令によりストアされる値のデータ・サイズも、先行する排他的ロード命令によってロードされた値と同じである必要があります。つまり、ソフトウェアは、同期操作を実行するには、常に排他的ロード命令とそれに対応する排他的ストア命令を使用する必要があります（37 ページの同期プリミティブを参照）。

排他的ストア命令は、そのストアを実行すると、デスティネーション・レジスタに 0 を書き込みます。そのストアを実行しない場合は、デスティネーション・レジスタに 1 を書き込みます。排他的ストア命令がデスティネーション・レジスタに 0 を書き込むと、システム内の他のプロセスが、排他的ロード命令と排他的ストア命令の間のメモリ位置にアクセスしていないことが保証されます。

性能上の理由により、該当する排他的ロード命令と排他的ストア命令の間の命令の数は最小限に抑えてください。

排他的ストア命令を、先行する排他的ロード命令で使用したアドレスとは異なるアドレスに対して実行した場合、結果は予測できません。

### 制限事項

これらの命令の場合：

- PC は使用しないでください。
- *Rd* と *Rt* には SP を使用しないでください。
- STREX の場合、*Rd* は *Rt* と *Rn* のどちらとも異なる必要があります。
- *offset* の値は、0 ~ 1020 の範囲内の 4 の倍数である必要があります。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
MOV     R1, #0x1              ; ロック取得値 try を初期化する
LDREX   R0, [LockAddr]        ; ロック値をロードする
CMP     R0, #0                ; ロック・フリーかどうか
ITT     EQ                    ; STREXEQ と CMPEQ の IT 命令
STREXEQ R0, R1, [LockAddr]    ; ロックの取得を試みる
CMPEQ   R0, #0                ; 成功したかどうか
BNE     try                   ; 失敗した場合は再試行する
...                           ; 成功した場合はロック取得済み。
```

## 3.4.10 CLREX

排他をクリア。

### 構文

`CLREX{cond}`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

### 動作

CLREX を使用すると、次の STREX、STREXB、または STREXH 命令がそのデスティネーション・レジスタに 1 を書き込み、ストアの実行に失敗します。例外ハンドラ・コードでは、同期操作の排他的ロード命令と対応する排他的ストア命令との間で例外が発生した場合、排他的ストアの失敗を強制すると便利です。

詳細については、37 ページの同期プリミティブを参照してください。

### 条件フラグ

この命令によるフラグの変更はありません。

### 例

```
CLREX
```

## 3.5 汎用データ処理命令

表 30 にデータ処理命令を示します：

**表 30. データ処理命令**

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| ADC | キャリー付き加算 | 86 ページのADD、ADC、SUB、SBC、および RSB |
| ADD | 加算 | 86 ページのADD、ADC、SUB、SBC、および RSB |
| ADDW | 加算 | 86 ページのADD、ADC、SUB、SBC、および RSB |
| AND | 論理積 | 88 ページのAND、ORR、EOR、BIC、および ORN |
| ASR | 算術右シフト | 89 ページのASR、LSL、LSR、ROR、および RRX |
| BIC | ビット·クリア | 88 ページのAND、ORR、EOR、BIC、および ORN |
| CLZ | 先行ゼロ·カウント | 90 ページのCLZ |
| CMN | 比較否定 | 91 ページのCMP および CMN |
| CMP | 比較 | 91 ページのCMP および CMN |
| EOR | 排他的論理和 | 88 ページのAND、ORR、EOR、BIC、および ORN |
| LSL | 論理左シフト | 89 ページのASR、LSL、LSR、ROR、および RRX |
| LSR | 論理右シフト | 89 ページのASR、LSL、LSR、ROR、および RRX |
| MOV | 転送 | 92 ページのMOV および MVN |
| MOVT | 上位に転送 | 93 ページのMOVT |
| MOVW | 16 ビット定数を転送 | 92 ページのMOV および MVN |
| MVN | 転送して否定 | 92 ページのMOV および MVN |
| ORN | 否定論理和 | 88 ページのAND、ORR、EOR、BIC、および ORN |
| ORR | 論理和 | 88 ページのAND、ORR、EOR、BIC、および ORN |
| RBIT | ビット順序反転 | 94 ページのREV、REV16、REVSH、および RBIT |
| REV | ワード内のバイト順序反転 | 94 ページのREV、REV16、REVSH、および RBIT |
| REV16 | 各ハーフワード内のバイト順序反転 | 94 ページのREV、REV16、REVSH、および RBIT |
| REVSH | 下位ハーフワード内のバイト順序を反転させ、符号拡張 | 94 ページのREV、REV16、REVSH、および RBIT |
| ROR | 右ローテート | 89 ページのASR、LSL、LSR、ROR、および RRX |
| RRX | 右ローテート、拡張付き | 89 ページのASR、LSL、LSR、ROR、および RRX |
| RSB | 反転減算 | 86 ページのADD、ADC、SUB、SBC、および RSB |
| SADD16 | 16 ビット符号付き加算 | 95 ページのSADD16 および SADD8 |
| SADD8 | 8 ビット符号付き加算 | 95 ページのSADD16 および SADD8 |
| SASX | 符号付き加算および減算、交換付き | 100 ページのSASX および SSAX |
| SSAX | 符号付き減算および加算、交換付き | 100 ページのSASX および SSAX |
| SBC | キャリー付き減算 | 86 ページのADD、ADC、SUB、SBC、および RSB |
| SHADD16 | 結果を半減する 16 ビット符号付き加算 | 96 ページのSHADD16 および SHADD8 |
| SHADD8 | 結果を半減する 8 ビット符号付き加算 | 96 ページのSHADD16 および SHADD8 |
| SHASX | 結果を半減する符号付き加算および減算、交換付き | 97 ページのSHASX および SHSAX |
| SHSAX | 結果を半減する符号付き減算および加算、交換付き | 97 ページのSHASX および SHSAX |

表 30. データ処理命令 （続き）

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| SHSUB16 | 結果を半減する 16 ビット符号付き減算 | 98 ページのSHSUB16 および SHSUB8 |
| SHSUB8 | 結果を半減する 8 ビット符号付き減算 | 98 ページのSHSUB16 および SHSUB8 |
| SSUB16 | 16 ビット符号付き減算 | 99 ページのSSUB16 および SSUB8 |
| SSUB8 | 8 ビット符号付き減算 | 99 ページのSSUB16 および SSUB8 |
| SUB | 減算 | 86 ページのADD、ADC、SUB、SBC、および RSB |
| SUBW | 減算 | 86 ページのADD、ADC、SUB、SBC、および RSB |
| TEQ | 等価テスト | 101 ページのTST および TEQ |
| TST | テスト | 101 ページのTST および TEQ |
| UADD16 | 16 ビット符号なし加算 | 102 ページのUADD16 および UADD8 |
| UADD8 | 8 ビット符号なし加算 | 102 ページのUADD16 および UADD8 |
| UASX | 符号なし加算および減算、交換付き | 103 ページのUASX および USAX |
| USAX | 符号なし減算および加算、交換付き | 103 ページのUASX および USAX |
| UHADD16 | 結果を半減する 16 ビット符号なし加算 | 104 ページのUHADD16 および UHADD8 |
| UHADD8 | 結果を半減する 8 ビット符号なし加算 | 104 ページのUHADD16 および UHADD8 |
| UHASX | 結果を半減する符号なし加算および減算、交換付き | 105 ページのUHASX および UHSAX |
| UHSAX | 結果を半減する符号なし減算および加算、交換付き | 105 ページのUHASX および UHSAX |
| UHSUB16 | 結果を半減する 16 ビット符号なし減算 | 106 ページのUHSUB16 および UHSUB8 |
| UHSUB8 | 結果を半減する 8 ビット符号なし減算 | 106 ページのUHSUB16 および UHSUB8 |
| USAD8 | 符号なし絶対差の和 | 107 ページのUSAD8 |
| USADA8 | 絶対差の符号なし和の累算。 | 108 ページのUSADA8 |
| USUB16 | 16 ビット符号なし減算 | 109 ページのUSUB16 および USUB8 |
| USUB8 | 8 ビット符号なし減算 | 109 ページのUSUB16 および USUB8 |

### 3.5.1 ADD、ADC、SUB、SBC、および RSB

加算、キャリー付き加算、減算、キャリー付き減算、および反転減算。

#### 構文

```
op{S}{cond} {Rd,} Rn, Operand2

op{cond} {Rd,} Rn, #imm12                    ; ADD および SUB のみ
```

ここで、

*op* これは次のいずれかです。
- ADD アドレス
- ADC キャリー付き加算
- SUB 減算
- SBC キャリー付き減算
- RSB 反転減算

S これは任意に指定できる接尾文字です。S が指定されている場合は、演算結果に基づいて条件コード·フラグが更新されます（68 ページの条件付き実行を参照）。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。*Rd* が省略されている場合、デスティネーション·レジスタは *Rn* になります。

*Rn* これは第 1 オペランドを保持するレジスタです。

*Operand2* これはフレキシブル第 2 オペランドです。オプションの詳細については63 ページのフレキシブル第 2 オペランドを参照してください。

*imm12* これは 0 ~ 4095 の範囲の値です。

#### 動作

ADD 命令は、*Operand2* または *imm12* の値を *Rn* の値に加算します。

ADC 命令は、*Rn* の値と *Operand2* の値、そしてキャリー·フラグも含めて加算します。

SUB 命令は、*Operand2* または *imm12* の値を *Rn* の値から減算します。

SBC 命令は、*Operand2* の値を *Rn* の値から減算します。キャリー·フラグがクリアされている場合は、結果から 1 が引かれます。

RSB 命令は、*Rn* の値を *Operand2* の値から減算します。*Operand2* にはさまざまなオプションがあるので、この命令は便利です。

ADC および SBC を使用してマルチワード算術演算を構成できます（87 ページのマルチワード算術演算の例を参照）。

72 ページのADRも参照してください。

ADDW は *imm12* オペランドを使用する ADD 構文と同じです。SUBW は *imm12* オペランドを使用する SUB 構文と同じです。

### 制限事項

これらの命令の場合：

- *Operand2* に SP または PC は使用できません
- *Rd* は ADD および SUB で使用する場合にのみ SP にできます。ただし追加制限事項があります。
  - *Rn* も SP にする必要があります。
  - *Operand2* のシフトは LSL を使用して最大でも 3 ビットに制限する必要があります。
- *Rn* は ADD および SUB で使用する場合にのみ SP にできます
- *Rd* は ADD{*cond*} PC, PC, Rm 命令の場合にのみ PC にできます。この場合、以下の制限事項があります。
  - 接尾文字 S は指定しないでください。
  - *Rm* に PC または SP は指定できません。
  - 命令が条件付きである場合は、その命令が IT ブロックの最後の命令である必要があります。
- ADD{*cond*} PC, PC, Rm 命令を除き、*Rn* は ADD および SUB で使用する場合にのみ PC にできます。ただし、追加制限事項があります。
  - 接尾文字 S は指定しないでください。
  - 第 2 オペランドは 0 ~ 4095 の範囲の定数である必要があります。
  - 加算または減算に PC を使用すると、PC のビット [1:0] が 0b00 に丸められてから、計算が行われるため、計算のベース･アドレスがワード境界で整列されます。
  - 命令のアドレスを生成する場合は、PC の値に基づいて、定数を調整する必要があります。Arm 社では、ADD や SUB 命令で PC に等しい *Rn* を指定する代わりに、ADR 命令を使用することを推奨しています。これは、ADR 命令では、アセンブラによって、適切な定数が自動的に計算されるためです。

*Rd* が ADD{*cond*} PC, PC, Rm 命令の PC である場合：

- PC に書き込まれる値のビット [0] は無視されます。
- 値のビット [0] を 0 にすることにより、生成されたアドレスに分岐が発生します。

### 条件フラグ

`S` が指定されている場合、これらの命令は演算結果に基づいて N、Z、C、および V の各フラグを更新します。

### 例

```
ADD     R2, R1, R3
SUBS    R8, R6, #240        ; 結果にフラグをセット。
RSB     R4, R4, #1280       ; 1280 から R4 の内容を減算。
ADCHI   R11, R0, R3         ; C フラグがセットされ、
                            ; Z フラグがクリアされた場合のみ実行
```

### マルチワード算術演算の例

例 3-4 : 64 ビット加算は、R2 と R3 に含まれる 64 ビット整数を、R0 と R1 に含まれる別の 64 ビット整数に加算し、その結果を R4 と R5 に格納する 2 つの命令を示しています。

### 例 3-4：64 ビット加算

```
ADDS    R4, R0, R2    ; 最下位ワードを加算。
ADC     R5, R1, R3    ; 最上位ワードをキャリー付きで加算。
```

マルチワードの値には、連続するレジスタを使用する必要はありません。例 3-5：96 ビット減算は、R9、R1 および R11 に含まれる 96 ビット整数を、R6、R2 および R8 に含まれる別の整数から減算する命令を示しています。この例では、結果を R6、R9 および R2 に格納します。

### 例 3-5：96 ビット減算

```
SUBS    R6, R6, R9    ; 最下位ワードを減算。
SBCS    R9, R2, R1    ; 中位ワードをキャリー付きで減算。
SBC     R2, R8, R11   ; 最上位ワードをキャリー付きで減算。
```

## 3.5.2 AND、ORR、EOR、BIC、および ORN

論理積、論理和、排他的論理和、ビット･クリア、および否定論理和。

### 構文

*op*{S}{*cond*} {*Rd*,} *Rn*, *Operand2*

ここで、

*op* これは次のいずれかです。
- AND 論理積。
- ORR 論理和、またはビット･セット。
- EOR 排他的論理和。
- BIC 論理積否定、またはビット･クリア。
- ORN 否定論理和。

S これは任意に指定できる接尾文字です。S が指定されている場合は、演算結果に基づいて条件コード･フラグが更新されます（68 ページの条件付き実行を参照）。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション･レジスタです。

*Rn* これは第 1 オペランドを保持するレジスタです。

*Operand2* これはフレキシブル第 2 オペランドです。オプションの詳細については63 ページのフレキシブル第 2 オペランドを参照してください。

### 動作

AND、EOR、および ORR の各命令は、*Rn* および *Operand2* の値に対し、それぞれビット単位の論理積、排他的論理和、および論理和演算を実行します。

BIC 命令は、*Rn* 内のビットと、*Operand2* の値に含まれる対応する各ビットの補数との論理積演算を実行します。

ORN 命令は、*Rn* 内のビットと、*Operand2* の値に含まれる対応する各ビットの補数との論理和演算を実行します。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

S が指定されている場合、これらの命令では以下のようになります。

- 結果に応じて N フラグおよび Z フラグを更新します。
- *Operand2* の計算中に C フラグを更新できます（88 ページの例 3-5 : 96 ビット減算を参照）。
- V フラグに影響することはありません。

### 例

```
AND     R9, R2, #0xFF00
ORREQ   R2, R0, R5
ANDS    R9, R8, #0x19
EORS    R7, R11, #0x18181818
BIC     R0, R1, #0xab
ORN     R7, R11, R14, ROR #4
ORNS    R7, R11, R14, ASR #32
```

## 3.5.3 ASR、LSL、LSR、ROR、および RRX

算術右シフト、論理左シフト、論理右シフト、右ローテート、および拡張付き右ローテート。

### 構文

*op*{S}{*cond*} *Rd*, *Rm*, *Rs*

*op*{S}{*cond*} *Rd*, *Rm*, #*n*

RRX{S}{*cond*} *Rd*, *Rm*

ここで、

*op* これは次のいずれかです。
- ASR 算術右シフト。
- LSL 論理左シフト。
- LSR 論理右シフト。
- ROR 右ローテート

S これは任意に指定できる接尾文字です。S が指定されている場合は、演算結果に基づいて条件コード·フラグが更新されます（68 ページの条件付き実行を参照）。

*Rd* これはデスティネーション·レジスタです。

*Rm* これはシフトされる値を保持するレジスタです。

*Rs* これは *Rm* の値に適用されるシフト長を保持するレジスタです。最下位バイトのみが使用され、0 ~ 255 の範囲で指定できます。

*n* これはシフト長です。シフト長の範囲は、以下のように命令に応じて異なります。
- ASR 1 ~ 32 のシフト長
- LSL 0 ~ 31 のシフト長
- LSR 1 ~ 32 のシフト長
- ROR 1 ~ 31 のシフト長

LSLS Rd, Rm, #0 の代わりに MOVS Rd, Rm の構文を使用することを推奨します。

### 動作

ASR、LSL、LSR、および ROR は *Rm* レジスタのビットを、定数 *n* またはレジスタ *Rs* で指定されている桁数だけ、左または右に転送します。

RRX は *Rm* レジスタのビットを右に 1 つ転送します。

これらのすべての命令では、結果は *Rd* に書き込まれますが、*Rm* レジスタの値は変更されません。各種命令から生成される結果の詳細については、64 ページのシフト演算を参照してください。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

S が指定されている場合：

- これらの命令は、結果に応じて N フラグおよび Z フラグを更新します。
- C フラグは、シフト長が 0 の場合を除き、最後にシフトアウトされたビットに更新されます（64 ページのシフト演算を参照）。

### 例

```
ASR     R7, R8, #9  ; 9 ビット算術右シフト。
LSLS    R1, R2, #3  ; 3 ビット論理左シフトし、フラグを更新。
LSR     R4, R5, #6  ; 6 ビット論理右シフト。
ROR     R4, R5, R6  ; R6 の下位バイトの値だけ右ローテートします。
RRX     R4, R5      ; 右ローテートして拡張。
```

## 3.5.4 CLZ

先行ゼロ･カウント。

### 構文

`CLZ{cond} Rd, Rm`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*Rd* これはデスティネーション･レジスタです。
*Rm* これはオペランド･レジスタです。

### 動作

CLZ 命令は *Rm* の値の先行ゼロ・カウントの数をカウントし、結果を *Rd* に返します。結果の値はビットがセットされていない場合は 32、ビット [31] がセットされている場合はゼロです。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

この命令によるフラグの変更はありません。

#### 例

```
CLZ     R4,R9
CLZNE   R2,R3
```

### 3.5.5 CMP および CMN

比較および比較否定。

#### 構文

`CMP{cond} Rn, Operand2`

`CMN{cond} Rn, Operand2`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rn* これは第 1 オペランドを保持するレジスタです。

*Operand2* これはフレキシブル第 2 オペランドです。オプションの詳細については 63 ページのフレキシブル第 2 オペランドを参照してください。

#### 動作

これらの命令は、レジスタ内の値と *Operand2* を比較します。結果に基づいて条件フラグを更新しますが、結果はレジスタに書き込みません。

CMP 命令は、*Operand2* の値を *Rn* の値から減算します。これは、結果が破棄されることを除けば、SUBS 命令と同じです。

CMN 命令は、*Operand2* の値を *Rn* の値に加算します。これは、結果が破棄されることを除けば、ADDS 命令と同じです。

#### 制限事項

これらの命令の場合：

- PC は使用しないでください。
- *Operand2* は SP にしないでください。

#### 条件フラグ

これらの命令は、演算結果に基づいて N、Z、C、および V の各フラグを更新します。

#### 例

```
CMP     R2, R9
CMN     R0, #6400
CMPGT   SP, R7, LSL #2
```

### 3.5.6 MOV および MVN

転送および転送して否定。

#### 構文

`MOV{S}{cond} Rd, Operand2`

`MOV{cond} Rd, #imm16`

`MVN{S}{cond} Rd, Operand2`

ここで、

`S` これは任意に指定できる接尾文字です。`S` が指定されている場合は、演算結果に基づいて条件コード·フラグが更新されます（68 ページの条件付き実行を参照）。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。

*Operand2* これはフレキシブル第 2 オペランドです。オプションの詳細については63 ページのフレキシブル第 2 オペランドを参照してください。

*imm16* これは 0 ~ 65535 の範囲の値です。

#### 動作

MOV 命令は、*Operand2* の値を *Rd* にコピーします。

MOV 命令の *Operand2* が LSL #0 以外のシフトのレジスタの場合、代わりに対応する以下の各シフト命令を使用することを推奨します。

- `MOV{S}{cond} Rd, Rm, ASR #n` の代わりに `ASR{S}{cond} Rd, Rm, #n` の構文を推奨します。
- `MOV{S}{cond} Rd, Rm, LSL #n if n != 0` の代わりに `LSL{S}{cond} Rd, Rm, #n` の構文を推奨します。
- `MOV{S}{cond} Rd, Rm, LSR #n` の代わりに `LSR{S}{cond} Rd, Rm, #n` の構文を推奨します。
- `MOV{S}{cond} Rd, Rm, ROR #n` の代わりに `ROR{S}{cond} Rd, Rm, #n` の構文を推奨します。
- `MOV{S}{cond} Rd, Rm, RRX` の代わりに `RRX{S}{cond} Rd, Rm` の構文を推奨します。

また、MOV 命令は、シフト命令と同義で別の形式の *Operand2* を許可します。

- `MOV{S}{cond} Rd, Rm, ASR Rs` は `ASR{S}{cond} Rd, Rm, Rs` と同義です。
- `MOV{S}{cond} Rd, Rm, LSL Rs` は `LSL{S}{cond} Rd, Rm, Rs` と同義です。
- `MOV{S}{cond} Rd, Rm, LSR Rs` は `LSR{S}{cond} Rd, Rm, Rs` と同義です。
- `MOV{S}{cond} Rd, Rm, ROR Rs` は `ROR{S}{cond} Rd, Rm, Rs` と同義です。

89 ページのASR、LSL、LSR、ROR、および RRX を参照してください。

MVN 命令は、*Operand2* の値を取り、この値に対してビット単位の論理否定演算を実行し、結果を *Rd* に入れます。

MOVW 命令は MOV と同じ機能を提供しますが、*imm16* オペランドの使用のみに制限されています。

### 制限事項

MOV 命令でのみ SP と PC を使用できます。ただし、次の制限が付きます。

- 第 2 オペランドがシフトなしのレジスタである必要があります。
- 接尾文字 S は指定しないでください。

*Rd* が MOV 命令の PC の場合：

- PC に書き込まれる値のビット [0] は無視されます。
- 値のビット [0] を 0 にすることにより、生成されたアドレスに分岐が発生します。

MOV を分岐命令として使用することは可能ですが、Arm 命令セットへのソフトウェアの移植性を考えて、分岐には BX または BLX 命令を使用することが推奨されています。

### 条件フラグ

S が指定されている場合、これらの命令では以下のようになります。

- 結果に応じて N フラグおよび Z フラグを更新します。
- *Operand2* の計算中に C フラグを更新できます（63 ページのフレキシブル第 2 オペランドを参照）。
- V フラグに影響することはありません。

### 例

```
MOVS   R11, #0x000B     ; 0x000B の値を R11 に書き込む。フラグが更新される。
MOV    R1, #0xFA05      ; 0xFA05 の値を R1 に書き込む。フラグは更新されない。
MOVS   R10, R12         ; R12 の値を R10 に書き込む。フラグが更新される。
MOV    R3, #23          ; 23 の値を R3 に書き込む。
MOV    R8, SP           ; スタック･ポインタの値を R8 に書き込む。
MVNS   R2, #0xF         ;0xFFFFFFF0 の値（0xF のビット反転）を R2 に書き込み、
                        ; フラグを更新する。
```

## 3.5.7 MOVT

上位転送。

### 構文

`MOVT{cond} Rd, #imm16`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*Rd* これはデスティネーション･レジスタです。
*imm16* これは 16 ビットのイミディエート定数です。

### 動作

MOVT は 16 ビットのイミディエート値 *imm16* を、デスティネーション･レジスタの上位ハーフワードである *Rd*[31:16] に書き込みます。この書込みが *Rd*[15:0] に影響することはありません。

MOV と MOVT の命令対で、任意の 32 ビット･イミディエート定数を生成できます。

### 制限事項

*Rd* に SP または PC は使用できません。

### 条件フラグ

この命令によるフラグの変更はありません。

### 例

```
MOVT    R3, #0xF123 ; R3 の上位ハーフワードに 0xF123 を書き込みますが、
                    ; 下位ハーフワードと APSR は変更されません。
```

## 3.5.8 REV、REV16、REVSH、および RBIT

バイトおよびビットの反転。

### 構文

*op{cond} Rd, Rn*

ここで、

*op* これは次のいずれかです。

- REV ワード内のバイトの順序を反転させます。
- REV16 各ハーフワード内のバイト順序を独自に反転させます。
- REVSH 下位ハーフワード内のバイト順序を反転させ、それを 32 ビットに符号拡張します。
- RBIT 32 ビットワード内のビット順序を反転させます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn* これはオペランドを保持するレジスタです。

### 動作

これらの命令を使用して、データのエンディアン方式を変更します。

REV 次のいずれかを変換します。

- 32 ビットのビッグエンディアン・データをリトルエンディアン・データに変換。
- 32 ビットのリトルエンディアン・データをビッグエンディアン・データに変換。

REV16 次のいずれかを変換します。

- 16 ビットのビッグエンディアン・データをリトルエンディアン・データに変換。
- 16 ビットのリトルエンディアン・データをビッグエンディアン・データに変換。

REVSH 次のいずれかを変換します。

- 16 ビットの符号付きビッグエンディアン・データを 32 ビットの符号付きリトルエンディアン・データに変換。
- 16 ビットの符号付きリトルエンディアン・データを 32 ビットの符号付きビッグエンディアン・データに変換。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
REV    R3, R7   ; R7 の値のバイト順序を反転させ、それを R3 に書き込む。
REV16  R0, R0   ; R0 内の 16 ビットの各ハーフワードのバイト順序を反転させる。
REVSH  R0, R5   ; 符号付きハーフワードを反転させる。
REVHS  R3, R7   ; 「より大きい」または「同じ」の条件を適用して反転させる。
RBIT   R7, R8   ; R8 の値のビット順序を反転させ、その結果を R7 に書き込む。
```

## 3.5.9 SADD16 および SADD8

16 ビット符号付き加算、および8 ビット符号付き加算。

### 構文

*op*{*cond*}{*Rd*,} *Rn*, *Rm*

ここで、

*op* これは次のいずれかです。
- SADD16 2 つの 16 ビットの符号付き整数の加算を実行します。
- SADD8 4 つの 8 ビットの符号付き整数の加算を実行します。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。

*Rn* これはオペランドを保持する 1 番目のレジスタです。

*Rm* これはオペランドを保持する 2 番目のレジスタです。

### 動作

これらの命令を使用して、並列ハーフワードまたはバイト単位の加算を実行します。

SADD16 命令：

1. 第 1 オペランドの各ハーフワードを第 2 オペランドの対応するハーフワードに加算します。
2. その結果をデスティネーション·レジスタの対応するハーフワードに書き込みます。

SADD8 命令：

1. 第 1 オペランドの各バイトを第 2 オペランドの対応するバイトに加算します。
2. その結果をデスティネーション·レジスタの対応するバイトに書き込みます。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
SADD16 R1, R0      ; R0 のハーフワードを R1 の対応するハーフワードに加算し、
                   ; R1 の対応するハーフワードに書き込む。
SADD8  R4, R0, R5  ; R0 のバイトを R5 の対応するバイトに加算し、
                   ; R4 の対応するバイトに書き込む。
```

### 3.5.10 SHADD16 および SHADD8

結果を半減する 16 ビット符号付き加算、および結果を半減する 8 ビット符号付き加算。

#### 構文

*op*{*cond*}{*Rd*,} *Rn*, *Rm*

ここで、

*op* これは次のいずれかです。
- SHADD16 結果を半減する 16 ビット符号付き加算
- SHADD8 結果を半減する 8 ビット符号付き加算

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn* これは第 1 オペランド・レジスタです。

*Rm* これは第 2 オペランド・レジスタです。

#### 動作

これらの命令を使用して、16 ビットおよび 8 ビットのデータを加算し、結果を半減させて、その結果をデスティネーション・レジスタに書き込みます。

SHADD16 命令：

1. 第 1 オペランドの各ハーフワードを第 2 オペランドの対応するハーフワードに加算します。
2. 結果を 1 ビット右へシフトして、データを半減させます。
3. ハーフワードの結果をデスティネーション・レジスタに書き込みます。

SHADDB8 命令：

1. 第 1 オペランドの各バイトを第 2 オペランドの対応するバイトに加算します。
2. 結果を 1 ビット右へシフトして、データを半減させます。
3. バイトの結果をデスティネーション・レジスタに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

#### 例

```
SHADD16 R1, R0      ; R0 のハーフワードを R1 の対応するハーフワードに加算し
                    ; 半減させた結果を、R1 の対応するハーフワードに書き込む。
SHADD8  R4, R0, R5 ; R0 のバイトを R5 の対応するバイトに加算し半減させた結果を、
                    ; R4 の対応するバイトに書き込む。
```

### 3.5.11 SHASX および SHSAX

結果を半減する符号付き加算および減算（交換付き）、および結果を半減する符号付き減算および加算（交換付き）。

#### 構文

`op{cond} {Rd}, Rn, Rm`

ここで、

*op* これは次のいずれかです。
- SHASX 結果を半減する加算および減算（交換付き）。
- SHSAX 結果を半減する減算および加算（交換付き）。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

#### 動作

SHASX 命令：

1. 第 1 オペランドの上位ハーフワードを第 2 オペランドの下位ハーフワードに加算します。
2. ハーフワードの加算結果をデスティネーション・レジスタの上位ハーフワードに書き込み、1 ビット右にシフトして、2 で除算、つまり半減させます。
3. 第 2 オペランドの上位ハーフワードを第 1 オペランドの下位ハーフワードから減算します。
4. ハーフワードの除算結果をデスティネーション・レジスタの下位ハーフワードに書き込み、1 ビット右にシフトして、2 で除算、つまり半減させます。

SHSAX 命令：

1. 第 2 オペランドの下位ハーフワードを第 1 オペランドの上位ハーフワードから減算します。
2. ハーフワードの加算結果をデスティネーション・レジスタの上位ハーフワードに書き込み、1 ビット右にシフトして、2 で除算、つまり半減させます。
3. 第 1 オペランドの下位ハーフワードを第 2 オペランドの上位ハーフワードに加算します。
4. ハーフワードの除算結果をデスティネーション・レジスタの下位ハーフワードに書き込み、1 ビット右にシフトして、2 で除算、つまり半減させます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令は条件コード・フラグに影響しません。

#### 例

```
SHASX    R7, R4, R2      ; R4 の上位ハーフワードを R2 の下位ハーフワードに加算し、
                         ; 半減させた結果を R7 の上位ハーフワードに書き込む。
                         ; R2 の上位ハーフワードを R4 の下位ハーフワードから減算し、
                         ; 半減させた結果を R7 の下位ハーフワードに書き込む。
SHSAX    R0, R3, R5      ; R3 の上位ハーフワードから R5 の下位ハーフワードを
```

```
                ; 減算し、半減させた結果を R0 の上位ハーフワードに書き込む。
                ; R5 の上位ハーフワードを R3 の下位ハーフワードに加算し、
                ; 半減させた結果を R0 の下位ハーフワードに書き込む。
```

### 3.5.12 SHSUB16 および SHSUB8

結果を半減する 16 ビット符号付き減算、および結果を半減する 8 ビット符号付き減算。

#### 構文

*op*{*cond*}{*Rd*,} *Rn*, *Rm*

ここで、

*op* これは次のいずれかです。
- SHSUB16 結果を半減する 16 ビット符号付き減算。
- SHSUB8 結果を半減する 8 ビット符号付き減算。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。

*Rn* これは第 1 オペランド·レジスタです。

*Rm* これは第 2 オペランド·レジスタです

#### 動作

これらの命令を使用して、16 ビットおよび 8 ビットのデータを加算し、結果を半減させて、その結果をデスティネーション·レジスタに書き込みます。

SHSUB16 命令：

1. 第 2 オペランドの各ハーフワードを第 1 オペランドの対応するハーフワードから減算します。
2. 結果を 1 ビット右へシフトして、データを半減させます。
3. 半減させたハーフワードの結果をデスティネーション·レジスタに書き込みます。

SHSUBB8 命令：

1. 第 2 オペランドの各バイトを第 1 オペランドの対応するバイトから減算します。
2. 結果を 1 ビット右へシフトして、データを半減させます。
3. 対応する符号付きバイト結果をデスティネーション·レジスタに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

#### 例

```
SHSUB16 R1, R0      ; R1 の対応するハーフワードから R0 のハーフワードを減算し、
                    ; R1 の対応するハーフワードに書き込む。
SHSUB8  R4, R0, R5  ; R0 のバイトを R5 の対応するバイトから減算し、
                    ; R4 の対応するバイトに書き込む。
```

### 3.5.13 SSUB16 および SSUB8

16 ビット符号付き減算、および8 ビット符号付き減算。

#### 構文

*op*{*cond*}{*Rd*,} *Rn*, *Rm*

ここで、

*op* これは次のいずれかです。
- SSUB16 2 つの 16 ビットの符号付き整数の減算を実行します。
- SSUB8 4 つの 8 ビットの符号付き整数の減算を実行します。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn* これは第 1 オペランド・レジスタです。

*Rm* これは第 2 オペランド・レジスタです。

#### 動作

これらの命令を使用して、データのエンディアン方式を変更します。

SSUB16 命令：

1. 第 2 オペランドの各ハーフワードを第 1 オペランドの対応するハーフワードから減算します。
2. 2 つの符号付きハーフワードの減算結果をデスティネーション・レジスタの対応するハーフワードに書き込みます。

SSUB8 命令：

1. 第 2 オペランドの各バイトを第 1 オペランドの対応するバイトから減算します。
2. 4 つの符号付きバイトの減算結果をデスティネーション・レジスタの対応するバイトに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

#### 例

```
SSUB16 R1, R0        ; R1 の対応するハーフワードから R0 のハーフワードを減算し、
                     ; R1 の対応するハーフワードに書き込む。
SSUB8  R4, R0, R5    ; R5 のバイトを R0 の対応するバイトから減算し、
                     ; R4 の対応するバイトに書き込む。
```

### 3.5.14 SASX および SSAX

符号付き加算および減算（交換付き）、および符号付き減算および加算（交換付き）。

#### 構文

`op{cond} {Rd}, Rm, Rn`

ここで、

*op* これは次のいずれかです。
- SASX 符号付き加算および減算、交換付き。
- SSAX 符号付き減算および加算、交換付き。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

#### 動作

SASX 命令：

1. 第 1 オペランドの符号付き上位ハーフワードを第 2 オペランドの符号付き下位ハーフワードに加算します。
2. 符号付き加算結果をデスティネーション・レジスタの上位ハーフワードに書き込みます。
3. 第 2 オペランドの符号付き下位ハーフワードを第 1 オペランドの符号付き上位ハーフワードから減算します。
4. 符号付き減算結果をデスティネーション・レジスタの下位ハーフワードに書き込みます。

SSAX 命令：

1. 第 2 オペランドの符号付き下位ハーフワードを第 1 オペランドの符号付き上位ハーフワードから減算します。
2. 符号付き減算結果をデスティネーション・レジスタの上位ハーフワードに書き込みます。
3. 第 1 オペランドの符号付き上位ハーフワードを第 2 オペランドの符号付き下位ハーフワードに加算します。
4. 符号付き加算結果をデスティネーション・レジスタの下位ハーフワードに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令は条件コード・フラグに影響しません。

#### 例

```
SASX    R0, R4, R5      ; R4 の上位ハーフワードを R5 の下位ハーフワードに加算し、
                        ; R0 の上位ハーフワードに書き込む。
                        ; R4 の上位ハーフワードから R5 の下位ハーフワードを減算して、
                        ; R0 の下位ハーフワードに書き込む。
SSAX    R7, R3, R2      ; R3 の下位ハーフワードから R2 の上位ハーフワードを減算し、
                        ; R7 の下位ハーフワードに書き込む。
                        ; R3 の上位ハーフワードを R R2 の下位ハーフワードに加算し、
                        ; R7 の上位ハーフワードに書き込む。
```

### 3.5.15 TST および TEQ

ビット・テストおよび等価テスト。

#### 構文

```
TST{cond} Rn, Operand2
TEQ{cond} Rn, Operand2
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rn* これは第 1 オペランドを保持するレジスタです。

*Operand2* これはフレキシブル第 2 オペランドです。オプションの詳細については 63 ページのフレキシブル第 2 オペランドを参照してください。

#### 動作

これらの命令は、レジスタ内の値を *Operand2* に対してテストします。結果に基づいて条件フラグを更新しますが、結果はレジスタに書き込みません。

TST 命令は、*Rn* の値および *Operand2* の値に対してビット単位論理積演算を実行します。これは、結果が破棄されることを除けば、ANDS 命令と同じです。

*Rn* のビットが 0 または 1 のどちらなのかをテストするには、そのビットを 1 にセットし、その他のビットをすべて 0 にクリアした *Operand2* 定数を使用して、TST 命令を実行します。

TEQ 命令は、*Rn* の値および *Operand2* の値に対してビット単位の排他的論理和演算を実行します。これは、結果が破棄されることを除けば、EORS 命令と同じです。

TEQ 命令を使用して、V フラグや C フラグに影響を与えずに、2 つの値が等しいかどうかをテストします。

TEQ は値の符号をテストするのにも役立ちます。比較後の N フラグは、2 つのオペランドの符号ビットの排他的論理和になります。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令の場合：

- 結果に応じて N フラグおよび Z フラグを更新します。
- *Operand2* の計算中に C フラグを更新できます（63 ページのフレキシブル第 2 オペランドを参照）。
- V フラグに影響することはありません。

#### 例

```
TST     R0, #0x3F8  ; R0 の値と 0x3F8 のビット単位論理積演算を行い、
                    ; APSR を更新しますが、結果は破棄されます。
TEQEQ   R10, R9     ; R10 の値が R9 の値と等しいかどうかを条件付きでテストし、
                    ; APSR を更新しますが結果は破棄されます。
```

### 3.5.16 UADD16 および UADD8

16 ビット符号なし加算、および 8 ビット符号なし加算。

#### 構文

*op*{*cond*}{*Rd*,} *Rn*, *Rm*

ここで、

*op* これは次のいずれかです。
- UADD16 2 つの 16 ビットの符号なし整数を加算します。
- UADD8 4 つの 8 ビットの符号なし整数を加算します。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn* これはオペランドを保持する 1 番目のレジスタです。

*Rm* これはオペランドを保持する 2 番目のレジスタです。

#### 動作

これらの命令を使用して、16 ビットおよび 8 ビットの符号なしデータを加算します。

UADD16 命令：

1. 第 1 オペランドの各ハーフワードを第 2 オペランドの対応するハーフワードに加算します。
2. その符号なしの結果をデスティネーション・レジスタの対応するハーフワードに書き込みます。

UADD16 命令：

1. 第 1 オペランドの各バイトを第 2 オペランドの対応するバイトに加算します。
2. その符号なしの結果をデスティネーション・レジスタの対応するバイトに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

#### 例

```
UADD16 R1, R0       ; R1 の対応するハーフワードに R0 のハーフワードを加算し、
                    ; R1 の対応するハーフワードに書き込む。
UADD8  R4, R0, R5   ; R0 のバイトを R5 の対応するバイトに加算し、
                    ; R4 の対応するバイトに書き込む。
```

### 3.5.17 UASX および USAX

交換付き加算および減算、および交換付き減算および加算。

#### 構文

```
op{cond} {Rd}, Rn, Rm
```

ここで、

*op* これは次のいずれかです。
- UASX 交換付き加算および減算。
- USAX 交換付き減算および加算。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

#### 動作

UASX 命令：

1. 第 2 オペランドの上位ハーフワードを第 1 オペランドの下位ハーフワードから減算します。
2. 符号なし減算結果をデスティネーション·レジスタの下位ハーフワードに書き込みます。
3. 第 1 オペランドの上位ハーフワードを第 2 オペランドの下位ハーフワードに加算します。
4. 符号なし加算結果をデスティネーション·レジスタの上位ハーフワードに書き込みます。

USAX 命令：

1. 第 1 オペランドの下位ハーフワードを第 2 オペランドの上位ハーフワードに加算します。
2. 符号なし加算結果をデスティネーション·レジスタの下位ハーフワードに書き込みます。
3. 第 2 オペランドの下位ハーフワードを第 1 オペランドの上位ハーフワードから減算します。
4. 符号なし減算結果をデスティネーション·レジスタの上位ハーフワードに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令は条件コード·フラグに影響しません。

#### 例

```
UASX    R0, R4, R5      ; R4 の上位ハーフワードを R5 の下位ハーフワードに加算し、
                        ; R0 の上位ハーフワードに書き込む。
                        ; R0 の上位ハーフワードから R5 の下位ハーフワードを減算して、
                        ; R0 の下位ハーフワードに書き込む。
USAX    R7, R3, R2      ; R3 の下位ハーフワードから R2 の上位ハーフワードを減算し、
                        ; R7 の下位ハーフワードに書き込む。
                        ; R3 の上位ハーフワードを R2 の下位ハーフワードに加算し、
                        ; R7 の上位ハーフワードに書き込む。
```

### 3.5.18 UHADD16 および UHADD8

結果を半減する 16 ビット符号なし加算、および結果を半減する 8 ビット符号なし加算。

#### 構文

*op*{*cond*}{*Rd*,} *Rn*, *Rm*

ここで、

*op* これは次のいずれかです。
UHADD16 結果を半減する 16 ビット符号なし加算。
UHADD8 結果を半減する 8 ビット符号なし加算。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn* これは第 1 オペランドを保持するレジスタです。

*Rm* これは第 2 オペランドを保持するレジスタです。

#### 動作

これらの命令を使用して、16 ビットおよび 8 ビットのデータを加算し、結果を半減させて、その結果をデスティネーション・レジスタに書き込みます。

UHADD16 命令：

1. 第 1 オペランドの各ハーフワードを第 2 オペランドの対応するハーフワードに加算します。
2. ハーフワードの結果を 1 ビット右へシフトして、データを半減させます。
3. その符号なしの結果をデスティネーション・レジスタの対応するハーフワードに書き込みます。

UHADD8 命令：

1. 第 1 オペランドの各バイトを第 2 オペランドの対応するバイトに加算します。
2. バイトの結果を 1 ビット右へシフトして、データを半減させます。
3. その符号なしの結果をデスティネーション・レジスタの対応するバイトに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

#### 例

```
UHADD16 R7, R3       ; R7 のハーフワードを R3 の対応するハーフワードに加算し、
                     ; 半減させた結果を R7 の対応するハーフワードに書き込む。
UHADD8  R4, R0, R5   ; R0 のバイトを R5 の対応するバイトに加算し半減させた結果を、
                     ; R4 の対応するバイトに書き込む。
```

### 3.5.19 UHASX および UHSAX

結果を半減する符号なし加算および減算（交換付き）、および結果を半減する符号なし減算および加算（交換付き）。

#### 構文

```
op{cond} {Rd}, Rn, Rm
```

ここで、

*op* これは次のいずれかです。
- UHASX 結果を半減する加算および減算（交換付き）。
- UHSAX 結果を半減する減算および加算（交換付き）。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

#### 動作

UHASX 命令：

1. 第 1 オペランドの上位ハーフワードを第 2 オペランドの下位ハーフワードに加算します。
2. 結果を 1 ビット右にシフトして、2 で除算、つまり半減させます。
3. ハーフワードの加算結果をデスティネーション·レジスタの上位ハーフワードに書き込みます。
4. 第 2 オペランドの上位ハーフワードを第 1 オペランドの下位ハーフワードから減算します。
5. 結果を 1 ビット右にシフトして、2 で除算、つまり半減させます。
6. ハーフワードの除算結果をデスティネーション·レジスタの下位ハーフワードに書き込みます。

UHSAX 命令：

1. 第 2 オペランドの下位ハーフワードを第 1 オペランドの上位ハーフワードから減算します。
2. 結果を 1 ビット右にシフトして、2 で除算、つまり半減させます。
3. ハーフワードの減算結果をデスティネーション·レジスタの上位ハーフワードに書き込みます。
4. 第 1 オペランドの下位ハーフワードを第 2 オペランドの上位ハーフワードに加算します。
5. 結果を 1 ビット右にシフトして、2 で除算、つまり半減させます。
6. ハーフワードの加算結果をデスティネーション·レジスタの下位ハーフワードに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令は条件コード·フラグに影響しません。

#### 例

```
UHASX    R7, R4, R2      ; R4 の上位ハーフワードと R2 の下位ハーフワードを加算し、
                         ; 半減させた結果を R7 の上位ハーフワードに書き込む。
                         ; R2 の上位ハーフワードを R7 の下位ハーフワードから減算し、
                         ; 半減させた結果を R7 の下位ハーフワードに書き込む。
UHSAX    R0, R3, R5      ; R3 の上位ハーフワードから R5 の下位ハーフワードを減算し、
                         ; 半減させた結果を R0 の上位ハーフワードに書き込む。
                         ; R5 の上位ハーフワードを R3 の下位ハーフワードに加算し、
                         ; 半減させた結果を R0 の下位ハーフワードに書き込む。
```

### 3.5.20 UHSUB16 および UHSUB8

結果を半減する 16 ビット符号なし減算、および結果を半減する 8 ビット符号なし減算。

#### 構文

*op{cond}{Rd,} Rn, Rm*

ここで、

*op* これは次のいずれかです。
- UHSUB16 2 つの符号なし 16 ビットの整数を減算し、結果を半減させて、その結果をデスティネーション·レジスタに書き込みます。
- UHSUB8 4 つの符号なし 8 ビットの整数を減算し、結果を半減させて、その結果をデスティネーション·レジスタに書き込みます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。

*Rn* これはオペランドを保持する 1 番目のレジスタです。

*Rm* これはオペランドを保持する 2 番目のレジスタです。

#### 動作

これらの命令を使用して、16 ビットおよび 8 ビットのデータを加算し、結果を半減させて、その結果をデスティネーション·レジスタに書き込みます。

UHSUB16 命令：

1. 第 2 オペランドの各ハーフワードを第 1 オペランドの対応するハーフワードから減算します。
2. 各ハーフワード結果を 1 ビット右にシフトし、データを半減させます。
3. 各符号なしの結果をデスティネーション·レジスタの対応するハーフワードに書き込みます。

UHSUB8 命令：

1. 第 2 オペランドの各バイトを第 1 オペランドの対応するバイトから減算します。
2. バイトの各結果を 1 ビット右へシフトして、データを半減させます。
3. その符号なしのバイトの結果をデスティネーション·レジスタの対応するバイトに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

#### 例

```
UHSUB16  R1, R0      ; R1 の対応するハーフワードから R0 のハーフワードを減算し、
                     ; 半減させた結果を R1 の対応するハーフワードに書き込む。
UHSUB8   R4, R0, R5  ; R0 の対応するバイトから R5 のバイトを減算し、
                     ; 半減させた結果を R4 の対応するバイトに書き込む。
```

### 3.5.21 SEL

バイトの選択。GE フラグの値に基づいて、第 1 オペランドまたは第 2 オペランドいずれかから結果の各バイトを選択します。

#### 構文

```
SEL{<c>}{<q>} {<Rd>,} <Rn>, <Rm>
```

ここで、

<c>、<q> これらは標準アセンブラ構文フィールドです。
<Rd> これはデスティネーション・レジスタです。
<Rn> これは第 1 オペランド・レジスタです。
<Rm> これは第 2 オペランド・レジスタです。

#### 動作

SEL 命令：

1. APSR.GE の各ビットの値を読み取ります。
2. APSR.GE の値に応じて、デスティネーション・レジスタに第 1 または第 2 オペランドレジスタの値を割り当てます。

#### 制限事項

なし。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

#### 例

```
SADD16 R0, R1, R2      ; 結果に基づいて GE ビットをセット。
SEL R0, R0, R3         ; GE に基づいて、R0 または R3 からバイトを選択
```

### 3.5.22 USAD8

符号なし絶対差の和。

#### 構文

```
USAD8{cond}{Rd,} Rn, Rm
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*Rd* これはデスティネーション・レジスタです。
*Rn* これは第 1 オペランド・レジスタです。
*Rm* これは第 2 オペランド・レジスタです。

#### 動作

USAD8 命令：

1. 第 2 オペランド·レジスタの各バイトを第 1 オペランド·レジスタの対応するバイトから減算します。
2. 絶対差の値をすべて加算します。
3. 結果をデスティネーション·レジスタに書き込みます。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
USAD8 R1, R4, R0        ; R4 の対応するバイトから R0 の各バイトを減算し、
                        ; 差を加算して R1 に書き込む。
USAD8  R0, R5           ; R0 の対応するバイトから R5 のバイトを減算し、差を加算して
                        ; R0 に書き込む。
```

## 3.5.23 USADA8

絶対差の符号なし和の累算。

### 構文

`USADA8{cond}{Rd,} Rn, Rm, Ra`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。

*Rn* これは第 1 オペランド·レジスタです。

*Rm* これは第 2 オペランド·レジスタです。

*Ra* これは累算値を含むレジスタです。

### 動作

USADA8 命令：

1. 第 2 オペランド·レジスタの各バイトを第 1 オペランド·レジスタの対応するバイトから減算します。
2. 符号なし絶対差をすべて加算します。
3. 累算値を絶対差の和に加算します。
4. 結果をデスティネーション·レジスタに書き込みます。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
USADA8 R1, R0, R6       ; R1 の対応するハーフワードから R0 のバイトを減算し、
                        ; 差分を加算し、R6 の値を加算して、R1 に書き込む。
USADA8  R4, R0, R5, R2  ; R0 の対応するバイトから R5 のバイトを減算し、差分を加算し、
                        ; R2 の値を加算して R4 に書き込む。
```

## 3.5.24 USUB16 および USUB8

16 ビット符号なし減算および 8 ビット符号なし減算。

### 構文

*op*{*cond*}{*Rd*,} *Rn*, *Rm*

ここで、

*op* これは次のいずれかです。
- USUB16 16 ビット符号なし減算
- USUB8 8 ビット符号なし減算

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn* これは第 1 オペランド・レジスタです。

*Rm* これは第 2 オペランド・レジスタです。

### 動作

これらの命令を使用して、16 ビットおよび 8 ビットのデータを減算し、結果をデスティネーション・レジスタに書き込みます。

USUB16 命令：

1. 第 2 オペランド・レジスタの各ハーフワードを第 1 オペランド・レジスタの対応するハーフワードから減算します。
2. その符号なしの結果をデスティネーション・レジスタの対応するハーフワードに書き込みます。

USUB8 命令：

1. 第 2 オペランド・レジスタの各バイトを第 1 オペランド・レジスタの対応するバイトから減算します。
2. その符号なしのバイトの結果をデスティネーション・レジスタの対応するバイトに書き込みます。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
USUB16 R1, R0       ; R1 の対応するハーフワードから R0 のハーフワードを減算し、
                    ; R1 の対応するハーフワードに書き込む。
USUB8  R4, R0, R5   ; R0 の対応するバイトから R5 のバイトを減算し、
                    ; R4 の対応するバイトに書き込む。
```

## 3.6 乗算および除算命令

表 31 は乗算および除算命令を示しています：

**表 31. 乗算および除算命令**

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| MLA | 積和、32 ビットの結果 | 111 ページのMUL、MLA、および MLS |
| MLS | 積差、32 ビットの結果 | 111 ページのMUL、MLA、および MLS |
| MUL | 乗算、32 ビットの結果 | 111 ページのMUL、MLA、および MLS |
| SDIV | 符号付き除算 | 125 ページのSDIV および UDIV |
| SMLA[B,T] | 符号付き積和（ハーフワード） | 119 ページのSMMLA および SMMLS |
| SMLAD、SMLADX | デュアル符号付き積和 | 114 ページのSMLAD |
| SMLAL | 符号付き積和（32 x 32 + 64）、64 ビットの結果 | 124 ページのUMULL、UMLAL、SMULL および SMLAL |
| SMLAL[B,T] | 符号付き Long 積和（ハーフワード） | 115 ページのSMLAL および SMLALD |
| SMLALD、SMLALDX | デュアル符号付き Long 積和 | 115 ページのSMLAL および SMLALD |
| SMLAW[B\|T] | 符号付き積和（ワード×ハーフワード） | 113 ページのSMLA および SMLAW |
| SMLSD | デュアル符号付き積差 | 117 ページのSMLSD および SMLSLD |
| SMLSLD | デュアル符号付き Long 積差 | 117 ページのSMLSD および SMLSLD |
| SMMLA | 符号付き上位ワード積和 | 119 ページのSMMLA および SMMLS |
| SMMLS、SMMLSR | 符号付き上位ワード積差 | 119 ページのSMMLA および SMMLS |
| SMUAD、SMUADX | デュアル符号付き乗加算 | 121 ページのSMUAD および SMUSD |
| SMUL[B,T] | 符号付き乗算（ワード×ハーフワード） | 122 ページのSMUL および SMULW |
| SMMUL、SMMULR | 符号付き上位ワード乗算 | 120 ページのSMMUL |
| SMULL | 符号付き乗算（32 x 32）、64 ビットの結果 | 124 ページのUMULL、UMLAL、SMULL および SMLAL |
| SMULWB、SMULWT | 符号付き乗算（ワード×ハーフワード） | 122 ページのSMUL および SMULW |
| SMUSD、SMUSDX | デュアル符号付き積差 | 121 ページのSMUAD および SMUSD |
| UDIV | 符号なし除算 | 125 ページのSDIV および UDIV |
| UMAAL | 符号なし Long 積和累算（32 x 32 + 32 + 32）、64 ビットの結果 | 112 ページのUMULL、UMAAL、UMLAL |
| UMLAL | 符号なし積和（32 x 32 + 64）、64 ビットの結果 | 112 ページのUMULL、UMAAL、UMLAL |
| UMULL | 符号なし乗算（32 x 32）、64 ビットの結果 | 112 ページのUMULL、UMAAL、UMLAL |

### 3.6.1 MUL、MLA、および MLS

32 ビット·オペランドを使用して 32 ビットの結果を生成する、乗算、積和、および積差。

#### 構文

```
MUL{S}{cond} {Rd,} Rn, Rm ; 乗算
MLA{cond} Rd, Rn, Rm, Ra  ; 積和
MLS{cond} Rd, Rn, Rm, Ra  ; 積差
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

S これは任意に指定できる接尾文字です。S が指定されている場合は、演算結果に基づいて条件コード·フラグが更新されます（68 ページの条件付き実行を参照）。

*Rd* これはデスティネーション·レジスタです。*Rd* が省略されている場合、デスティネーション·レジスタは *Rn* になります。

*Rn*、*Rm* これらは乗算される値を保持するレジスタです。

*Ra* これは加算先または減算元の値を保持するレジスタです。

#### 動作

MUL 命令は、*Rn* と *Rm* の値を乗算し、結果の下位 32 ビットを *Rd* に格納します。

MLA 命令は、Rn と Rm の値を乗算し、*Ra* の値を加算して、結果の下位 32 ビットを Rd に格納します。

MLS 命令は、Rn と Rm の値を乗算し、その結果を *Ra* の値から減算して、その結果の下位 32 ビットを Rd に格納します。

これらの命令の結果は、オペランドの符号の有無には依存しません。

#### 制限事項

これらの命令では、SP および PC は使用しないでください。

接尾文字 S をMUL 命令で使用した場合は次のとおりです。

- *Rd*、*Rn*、および *Rm* はすべて R0 ~ R7 の範囲である必要があります。
- *Rd* は *Rm* と同じであることが必要です。
- *cond* 接尾文字は使用しないでください。

#### 条件フラグ

S が指定されている場合、MUL 命令では以下のようになります。

- 結果に応じて N フラグおよび Z フラグを更新します。
- C フラグと V フラグに影響することはありません。

#### 例

```
MUL     R10, R2, R5      ; 乗算、R10 = R2 x R5
MLA     R10, R2, R1, R5  ; 積和、R10 = (R2 x R1) + R5
MULS    R0, R2, R2       ; 乗算してフラグを更新、R0 = R2 x R2
MULLT   R2, R3, R2       ; 条件付き乗算、R2 = R3 x R2
MLS     R4, R5, R6, R7   ; 積差、R4 = R7 - (R5 x R6)
```

### 3.6.2 UMULL、UMAAL、UMLAL

32 ビット·オペランドを使用して、64 ビットの結果を生成する、符号なし Long 乗算と任意に指定できる累算。

#### 構文

`op{cond} RdLo, RdHi, Rn, Rm`

ここで、

*op* これは次のいずれかです。

- `UMULL` 符号なし Long 乗算。
- `UMAAL` 符号なし Long 積和累算。
- `UMLAL` 符号なし Long 積和。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*RdHi*、*RdLo* これらはデスティネーション・レジスタです。`UMAAL`、`UMLAL` および `UMLAL` の場合は、累算値も保持されます。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

#### 動作

これらの命令は *Rn* と *Rm* の値を符号なし 32 ビット整数として解釈します。
UMULL 命令：

- 第 1 オペランドと第 2 オペランドの 2 つの符号なし整数を乗算します。
- 結果の下位 32 ビットを *RdLo* に書き込みます。
- 結果の上位 32 ビットを *RdHi* に書き込みます。

UMAAL 命令：

- 第 1 オペランドと第 2 オペランドの 2 つの符号なし 32 ビット整数を乗算します。
- *RdHi* の符号なし 32 ビット整数を 64 ビットの乗算結果に加算します。
- *RdLo* の符号なし 32 ビット整数を 64 ビットの加算結果に加算します。
- 結果の上位 32 ビットを *RdHi* に書き込みます。
- 結果の下位 32 ビットを *RdLo* に書き込みます。

UMLAL 命令：

- 第 1 オペランドと第 2 オペランドの 2 つの符号なし整数を乗算します。
- 64 ビットの結果を *RdHi* および *RdLo* に含まれる 64 ビットの符号なし整数に加算します。
- 結果を *RdHi* および *RdLo* にライトバックします。

#### 制限事項

これらの命令の場合：

- SP および PC は使用しないでください。
- *RdHi* と *RdLo* は異なるレジスタである必要があります。

#### 条件フラグ

これらの命令は条件コード·フラグに影響しません。

### 例

```
UMULL   R0, R4, R5, R6   ; R5 と R6 を乗算し、上位 32 ビットを R4 に、
                         ; 下位 32 ビットを R0 に書き込む。
UMAAL   R3, R6, R2, R7   ; R2 と R7 を乗算し、R6 を加算し、R3 を加算し、
                         ; 上位 32 ビットを R6 に、下位 32 ビットを R3 に書き込む。
UMLAL   R2, R1, R3, R5   ; R5 と R3 を乗算し、R1:R2 を加算して、
                         ; R1:R2 に書き込む
```

## 3.6.3 SMLA および SMLAW

符号付き積和（ハーフワード）。

### 構文

`op{XY}{cond} Rd, Rn, Rm`

`op{Y}{cond} Rd, Rn, Rm, Ra`

ここで、

`op` これは次のいずれかです。

- `SMLA` 符号付き Long 積和（ハーフワード）
  - *X* および *Y* は、ソース・レジスタ *Rn* と *Rm* の上位と下位のどちらの半分を第 1 および第 2 乗算オペランドとして使用するのかを指定します。
  - *X* が *B* の場合は、*Rn* の下位ハーフワード、ビット [15:0] が使用されます。
  - *X* が *T* の場合は、*Rn* の上位ハーフワード、ビット [31:16] が使用されます。
  - *Y* が *B* の場合は、*Rm* の下位ハーフワード、ビット [15:0] が使用されます。
  - *Y* が *T* の場合は、*Rm* の上位ハーフワード、ビット [31:16] が使用されます。
- `SMLAW` 符号付き積和（ワード×ハーフワード）
  - *Y* は、ソース・レジスタ *Rm* の上位と下位のどちらの半分を第 2 乗算オペランドとして使用するのかを指定します。
  - *Y* が T の場合は*Rm* の上位ハーフワード、ビット [31:16] が使用されます。
  - *Y* が B の場合は*Rm* の下位ハーフワード、ビット [15:0] が使用されます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。*Rd* が省略されている場合、デスティネーション・レジスタは *Rn* になります。

*Rn*、*Rm* これらは乗算される値を保持するレジスタです。

*Ra* これは加算先または減算元の値を保持するレジスタです。

### 動作

SMALBB、SMLABT、SMLATB、SMLATT 命令：

- *Rn* と *Rm* の指定された符号付き（上位または下位）ハーフワード値を乗算します。
- *Ra* の値を 32 ビットの乗算結果に加算します。
- 積和の結果を *Rd* に書き込みます。

ソース・レジスタの未指定のハーフワードは無視されます。

SMLAWB および SMLAWT 命令：

- *Rn* の 32 ビットの符号付き値を以下と乗算します。
  - *Rm* の符号付き上位ハーフワード（*T* 接尾文字付き）。
  - *Rm* の符号付き下位ハーフワード（*B* 接尾文字付き）。
- *Ra* の 32 ビット符号付き値を 48 ビットの乗算結果の上位 32 ビットに加算します。
- 積和の結果を *Rd* に書き込みます。

48 ビットの乗算結果の下位 16 ビットは無視されます。

累算値を加算する際にオーバーフローが発生する場合は、この命令によって APSR に Q フラグがセットされます。乗算中にはオーバーフローは発生しません。

### 制限事項

これらの命令では、SP および PC は使用しないでください。

### 条件フラグ

オーバーフローが検出された場合は、Q フラグがセットされます。

### 例

```
SMLABB  R5, R6, R4, R1  ; R6 と R4 の下位ハーフワード同士を乗算し、
                        ; R1 を加算して、R5 に書き込む。
SMLATB  R5, R6, R4, R1  ; R6 の上位ハーフワードと R4 の下位ハーフワードを乗算し、
                        ; R1 を加算して、R5 に書き込む。
SMLATT  R5, R6, R4, R1  ; R6 と R4 の上位ハーフワード同士を乗算し、R1 を加算して、
                        ; その合計を R5 に書き込む。
SMLABT  R5, R6, R4, R1  ; R6 の下位ハーフワードを R4 の上位ハーフワードと乗算し、
                        ; R1 を加算して、R5 に書き込む。
SMLABT  R4, R3, R2      ; R4 の下位ハーフワードと R3 の上位ハーフワードを乗算し、
                        ; R2 を加算して、R4 に書き込む。
SMLAWB  R10, R2, R5, R3 ; R2 と R5 の下位ハーフワードを乗算し、R3 を結果に加算して、
                        ; 上位 32 ビットを R10 に書き込む。
SMLAWT  R10, R2, R1, R5 ; R2 と R1 の上位ハーフワードを乗算し、R5 を加算して、
                        ; 上位 32 ビットを R10 に書き込む。
```

## 3.6.4 SMLAD

デュアル符号付き Long 積和。

### 構文

`op{X}{cond} Rd, Rn, Rm, Ra`

ここで、

`op` これは次のいずれかです。

`SMLAD` デュアル符号付き積和。

`SMLADX` デュアル符号付き積和（交換付き）。

*X* は、ソース・レジスタ *Rn* の上位と下位のどちらのハーフワードを乗算オペランドとして使用するのかを指定します。
*X* が省略されている場合、乗算は下位 x 下位および上位 x 上位で実行されます。
*X* が指定されている場合、乗算は下位 x 上位および上位 x 下位で実行されます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn* これは乗算される値を保持する第 1 オペランド・レジスタです。

*Rm* これは第 2 オペランド・レジスタです。

*Ra* これは累算値です。

### 動作

SMLAD および SMLADX 命令は、2 つのオペランドを 4 つのハーフワード 16 ビット値とみなします。SMLAD および SMLADX 命令：

- *X* が指定されていない場合、*Rn* の符号付き上位ハーフワード値と *Rm* の符号付き上位ハーフワード値を、*Rn* の符号付き下位ハーフワード値と *Rm* の符号付き下位ハーフワード値を乗算します。
- あるいは、*X* が指定されている場合、*Rn* の符号付き上位ハーフワード値と *Rm* の符号付き下位ハーフワード値を、*Rn* の符号付き下位ハーフワード値と *Rm* の符号付き上位ハーフワード値を乗算します。
- 両方の乗算結果を *Ra* の符号付き 32 ビット値に加算します。
- 32 ビットの符号付き積和の結果を *Rd* に書き込みます。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
SMLAD   R10, R2, R1, R5 ; R2 の 2 つのハーフワード値を R1 の対応する
                        ; ハーフワード値で乗算し、R5 を加算して、R10 に書き込む。
SMLALDX R0, R2, R4, R6  ; R2 の上位ハーフワードと R4 の下位ハーフワードを乗算し、
                        ; R2 の下位ハーフワードと R4 の上位ハーフワードを乗算し、
                        ; R6 を加算して、R0 に書き込む。
```

## 3.6.5 SMLAL および SMLALD

符号付き Long 積和、符号付き Long 積和（ハーフワード）、およびデュアル符号付き Long 積和。

### 構文

*op*{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*

*op*{XY}{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*

*op*{*X*}{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*

ここで、

*op* これは次のいずれかです。

SMLAL 符号付き Long 積和。

SMLALXY 符号付き Long 積和（ハーフワード、X および Y）。
X および Y は、ソース・レジスタ *Rn* と *Rm* の上位と下位のどちらのハーフワードを第 1 および第 2 乗算オペランドとして使用するのかを指定します。

*X* が B の場合は、*Rn* の下位ハーフワード、ビット [15:0] が使用されます。
*X* が T の場合は、*Rn* の上位ハーフワード、ビット [31:16] が使用されます。
*Y* が B の場合は、*Rm* の下位ハーフワード、ビット [15:0] が使用されます。
*Y* が T の場合は、*Rm* の上位ハーフワード、ビット [31:16] が使用されます。

SMLALD デュアル符号付き Long 積和。

SMLALDX デュアル符号付き Long 積和（交換付き）。
*X* が省略されている場合、乗算は下位 x 下位および上位 x 上位で実行されます。
*X* が指定されている場合、乗算は下位 x 上位および上位 x 下位で実行されます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*RdHi*、*RdLo* これらはデスティネーション・レジスタです。
*RdLo* は 64 ビット整数の下位 32 ビット、*RdHi* は上位 32 ビットです。
SMLAL、SMLALBB、SMLALBT、SMLALTB、SMLALTT、SMLALD および SMLALDX の場合、これらには累算値も保持されます。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

## 動作

SMLAL 命令：

- *Rn* と *Rm* の 2 の補数となる符号付きワード値を乗算します。
- *RdLo* と *RdHi* の 64 ビット値を 64 ビットの乗算結果に加算します。
- 64 ビットの積和の結果を *RdLo* と *RdHi* に書き込みます。

SMLALBB、SMLALBT、SMLALTB および SMLALTT 命令：

- *Rn* と *Rm* の指定された符号付き（上位または下位）ハーフワード値を乗算します。
- 符号拡張された32 ビットの乗算結果を *RdLo* と *RdHi* の 64 ビット値に加算します。
- 64 ビットの積和の結果を *RdLo* と *RdHi* に書き込みます。

ソース・レジスタの未指定のハーフワードは無視されます。

SMLALD および SMLALDX 命令は、*Rn* と *Rm* の値を 4 つのハーフワードの 2 の補数となる符号付き 16 ビット整数値として解釈します。これらの命令の場合：

- *X* が指定されていない場合、*Rn* の符号付き上位ハーフワード値と *Rm* の符号付き上位ハーフワード値を、*Rn* の符号付き下位ハーフワード値と *Rm* の符号付き下位ハーフワード値を乗算します。
- あるいは、*X* が指定されている場合、*Rn* の符号付き上位ハーフワード値と *Rm* の符号付き下位ハーフワード値を、*Rn* の符号付き下位ハーフワード値と *Rm* の符号付き上位ハーフワード値を乗算します。
- 2 つの乗算結果を *RdLo* と *RdHi* の符号付き 64 ビット値に加算して、64 ビットの乗算結果を生成します。
- 64 ビットの乗算結果を *RdLo* および *RdHi* に書き込みます。

## 制限事項

これらの命令の場合：

- SP および PC は使用しないでください。
- *RdHi* と *RdLo* は異なるレジスタである必要があります。

## 条件フラグ

これらの命令は条件コード・フラグに影響しません。

### 例

```
SMLAL       R4, R5, R3, R8    ; R3 と R8 を乗算し、R5:R4 を加算して、
                              ; R5:R4 に書き込む。
SMLALBT     R2, R1, R6, R7    ; R6 の下位ハーフワードを R7 の上位ハーフワードで
                              ; 乗算し、32 ビットに符号拡張して R1:R2 を加算して、
                              ; R1:R2 に書き込む。
SMLALTB     R2, R1, R6, R7    ; R6 の上位ハーフワードと R7 の下位ハーフワードを
                              ; 乗算し、32 ビットに符号拡張して、R1:R2 を加算して、
                              ; R1:R2 に書き込む。
SMLALD      R6, R8, R5, R1    ; R5 と R1 の上位ハーフワードを乗算し、R5 と R1 の
                              ; 下位ハーフワードを乗算し、R8:R6 を加算して、
                              ; R8:R6 に書き込む。
SMLALDX     R6, R8, R5, R1    ; R5 の上位ハーフワードを R1 の下位ハーフワードと
                              ; 乗算し、R5 の下位ハーフワードと R1 の
                              ; 上位ハーフワードを乗算し、R8:R6 を加算して、
                              ; R8:R6 に書き込む。
```

## 3.6.6 SMLSD および SMLSLD

デュアル符号付き積差およびデュアル符号付き Long 積差。

### 構文

*op*{*X*}{*cond*} *Rd*, *Rn*, *Rm*, *Ra*

ここで、

*op* これは次のいずれかです。

- SMLSD デュアル符号付き積差。
- SMLSDX デュアル符号付き積差（交換付き）。
- SMLSLD デュアル符号付き Long 積差。
- SMLSLDX デュアル符号付き Long 積差（交換付き）。

*X* が指定されている場合、乗算は下位 x 上位および上位 x 下位で実行されます。
*X* が省略されている場合、乗算は下位 x 下位および上位 x 上位で実行されます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

*Ra* これは累算値を保持するレジスタです。

### 動作

SMLSD 命令は、第 1 オペランドと第 2 オペランドの値を 4 つの符号付きハーフワードとして解釈します。この命令の場合：

- 第 2 オペランドのハーフワードを任意にローテートできます。
- 符号付き 16 x 16 ビット・ハーフワードの乗算を 2 つ実行します。
- 上位ハーフワードの乗算結果を下位ハーフワードの乗算結果から減算します。
- 符号付き累算値を減算結果に加算します。
- 加算結果をデスティネーション・レジスタに書き込みます。

SMLSLD 命令は *Rn* と *Rm* の値を 4 つの符号付きハーフワードとして解釈します。
この命令の場合：

- 第 2 オペランドのハーフワードを任意にローテートできます。
- 符号付き 16 x 16 ビット・ハーフワードの乗算を 2 つ実行します。
- 上位ハーフワードの乗算結果を下位ハーフワードの乗算結果から減算します。
- *RdHi* と *RdLo* の 64 ビット値を減算結果に加算します。
- 64 ビットの加算結果を *RdHi* と *RdLo* に書き込みます。

### 制限事項

これらの命令の場合：

- SP および PC は使用しないでください。

### 条件フラグ

累算でオーバーフローが発生する場合は、この命令により、Q フラグがセットされます。乗算または減算ではオーバーフローは発生しません。

Thumb 命令セットの場合、これらの命令は条件コード・フラグに影響しません。

### 例

```
SMLSD       R0, R4, R5, R6    ; R4 の下位ハーフワードと
                              ; R5 の下位ハーフワードを乗算し、
                              ; R4 の上位ハーフワードと R5 の上位ハーフワードを
                              ; 乗算し、1 番目の値から 2 番目の値を減算し、
                              ; R6 を加算して R0 に書き込む。
SMLSDX      R1, R3, R2, R0    ; R3 の下位ハーフワードと R2 の上位ハーフワードを
                              ; 乗算し、R3 の上位ハーフワードと R2 の
                              ; 下位ハーフワードを乗算し、1 番目の値から
                              ; 2 番目の値を減算し、R0 を加算して、R1 に書き込む。
SMLSLD      R3, R6, R2, R7    ; R6 の下位ハーフワードと R2 の下位ハーフワードを
                              ; 乗算し、R6 の上位ハーフワードと R2 の
                              ; 上位ハーフワードを乗算し、1 番目の値から
                              ; 2 番目の値を減算し、R6:R3 を加算し、
                              ; R6:R3 に書き込む。
SMLSLDX     R3, R6, R2, R7    ; R6 の下位ハーフワードと R2 の上位ハーフワードを
                              ; 乗算し、R6 の上位ハーフワードと R2 の
                              ; 下位ハーフワードを乗算し、1 番目の値から
                              ; 2 番目の値を減算し、R6:R3 を加算して、
                              ; R6:R3 に書き込む。
```

### 3.6.7 SMMLA および SMMLS

符号付き上位ワードの積和および符号付き上位ワードの積差。

#### 構文

`op{R}{cond} Rd, Rn, Rm, Ra`

ここで、

*op* これは次のいずれかです。

- SMMLA 符号付き上位ワード積和。
- SMMLS 符号付き上位ワード積差。

R これは丸め誤差フラグです。*R* が指定されている場合は、演算結果が切り捨てられる代わりに丸められます。この場合、定数 0x80000000 が乗算結果に加算されてから、上位ワードが取り出されます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn*、*Rm* これらは第 1 乗算オペランドおよび第 2 乗算オペランドを保持するレジスタです。

*Ra* これは累算値を保持するレジスタです。

#### 動作

SMMLA 命令は *Rn* と *Rm* の値を符号付き 32 ビットワードとして解釈します。

SMMLA 命令：

- *Rn* と *Rm* の値を乗算します。
- 任意で、0x80000000 を加算して、値を丸めます。
- 結果の上位 32 ビットを抽出します。
- *Ra* の値を抽出した符号付き値に加算します。
- 加算結果を *Rd* に書き込みます。

SMMLS 命令は *Rn* と *Rm* の値を符号付き 32 ビットワードとして解釈します。

SMMLS 命令：

- *Rn* と *Rm* の値を乗算します。
- 任意で、0x80000000 を加算して、値を丸めます。
- 結果の上位 32 ビットを抽出します。
- 結果から抽出した値を *Ra* の値から減算します。
- 減算結果を *Rd* に書き込みます。

#### 制限事項

これらの命令の場合：

- SP および PC は使用しないでください。

#### 条件フラグ

これらの命令は条件コード・フラグに影響しません。

### 例

```
SMMLA      R0, R4, R5, R6   ; R4 と R5 を乗算し、上位 32 ビットを抽出し、
                            ; R6 を加算して、切り捨てて、R0 に書き込む。
SMMLAR     R6, R2, R1, R4   ; R2 と R1 を乗算し、上位 32 ビットを抽出し、
                            ; R4 を加算して丸め、R6 に書き込む。
SMMLSR     R3, R6, R2, R7   ; R6 と R2 を乗算し、上位 32 ビットを抽出し、
                            ; R7 を減算して、丸め、R3 に書き込む。
SMMLS      R4, R5, R3, R8   ; R5 と R3 を乗算し、上位 32 ビットを抽出し、
                            ; R8 を減算して、切り捨てて、R4 に書き込む。
```

## 3.6.8 SMMUL

符号付き上位ワード乗算。

### 構文

```
op{R}{cond} Rd, Rn, Rm
```

ここで、

`op` これは次のいずれかです。

`SMMUL` 符号付き上位ワード乗算

*R* これは丸め誤差フラグです。*R* が指定されている場合は、演算結果が切り捨てられる代わりに丸められます。この場合、定数 0x80000000 が乗算結果に加算されてから、上位ワードが取り出されます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

### 動作

SMMUL 命令は、*Rn* と *Rm* の値を 2 の補数の符号付き 32 ビット整数として解釈します。SMMUL 命令：

- *Rn* と *Rm* の値を乗算します。
- 任意で結果を丸めます。そうしない場合は、結果が切り捨てられます。
- 結果の符号付き上位 32 ビットを *Rd* に書き込みます。

### 制限事項

この命令の場合：

- SP および PC は使用しないでください。

### 条件フラグ

この命令は条件コード・フラグに影響しません。

### 例

```
SMULL      R0, R4, R5       ; R4 と R5 を乗算し、上位 32 ビットを切り捨てて、
                            ; R0 に書き込む。
SMULLR     R6, R2           ; R6 と R2 を乗算し、上位 32 ビットを丸めて、
                            ; R6 に書き込む。
```

### 3.6.9 SMUAD および SMUSD

デュアル符号付き積和、およびデュアル符号付き積差。

#### 構文

`op{X}{cond} Rd, Rn, Rm`

ここで、

*op* これは次のいずれかです。

- `SMUAD` デュアル符号付き乗加算。
- `SMUADX` デュアル符号付き積和（交換付き）。
- `SMUSD` デュアル符号付き積差。
- `SMUSDX` デュアル符号付き積差（交換付き）。

*X* が指定されている場合、乗算は下位 x 上位および上位 x 下位で実行されます。
*X* が省略されている場合、乗算は下位 x 下位および上位 x 上位で実行されます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

#### 動作

SMUAD 命令は、第 1 オペランドと第 2 オペランドの値を、各オペランドの 2 つの符号付きハーフワードとして解釈します。この命令の場合：

- 第 2 オペランドのハーフワードを任意にローテートできます。
- 符号付き 16 x 16 ビットの乗算を 2 つ実行します。
- 2 つの乗算結果を加算します。
- 加算結果をデスティネーション・レジスタに書き込みます。

SMUSD 命令は第 1 オペランドの値と第 2 オペランドの値を 2 の補数となる符号付き整数として解釈します。この命令の場合：

- 第 2 オペランドのハーフワードを任意にローテートできます。
- 符号付き 16 x 16 ビットの乗算を 2 つ実行します。
- 上位ハーフワードの乗算結果を下位ハーフワードの乗算結果から減算します。
- 減算結果をデスティネーション・レジスタに書き込みます。

#### 制限事項

これらの命令の場合：

- SP および PC は使用しないでください。

#### 条件フラグ

加算時にオーバーフローが発生する場合は、Q フラグをセットします。乗算ではオーバーフローすることはありません。

#### 例

```
SMUAD    R0, R4, R5        ; R4 の下位ハーフワードと R5 の下位ハーフワードを乗算し、
                           ; R4 の上位ハーフワードと R5 の上位ハーフワードの乗算を
                           ; 加算して、R0 に書き込む。
```

```
SMUADX   R3, R7, R4      ; R7 の下位ハーフワードと R4 の上位ハーフワードを乗算し、
                         ; R7 の上位ハーフワードと R4 の下位ハーフワードの乗算を
                         ; 加算して、R3 に書き込む。
SMUSD    R3, R6, R2      ; R4 の下位ハーフワードと R6 の下位ハーフワードを乗算し、
                         ; R6 の上位ハーフワードと R3 の上位ハーフワードの乗算を
                         ; 減算して、R3 に書き込む。
SMUSDX   R4, R5, R3      ; R5 の下位ハーフワードと R3 の上位ハーフワードを乗算し、
                         ; R5 の上位ハーフワードと R3 の下位ハーフワードの乗算を
                         ; 減算し、R4 に書き込む。
```

### 3.6.10 SMUL および SMULW

符号付き乗算（ハーフワード）および符号付き乗算（ワード×ハーフワード）。

#### 構文

*op*{*XY*}{*cond*} *Rd*,*Rn*, *Rm*

*op*{*Y*}{*cond*} *Rd*. *Rn*, *Rm*

*SMULXY* のみ：

*op* これは次のいずれかです。

SMUL{*XY*}：符号付き乗算（ハーフワード）。

*X* および *Y* は、ソース・レジスタ *Rn* と *Rm* の上位と下位のどちらのハーフワードを第 1 および第 2 乗算オペランドとして使用するのかを指定します。
*X* が B の場合は*Rn* の下位ハーフワード、ビット [15:0] が使用されます。
*X* が T の場合は、*Rn* の上位ハーフワード、ビット [31:16] が使用されます。*Y* が Bの場合は、*Rm* の下位ハーフワード、ビット [15:0] が使用されます。
*Y* が T の場合は、*Rm* の上位ハーフワード、ビット [31:16] が使用されます。

SMULW{*Y*}：符号付き乗算（ワード×ハーフワード）。

*Y* は、ソース・レジスタ *Rm* の上位と下位のどちらのハーフワードを第 2 乗算オペランドとして使用するのかを指定します。
*Y* が B の場合は *Rm* の下位ハーフワード（ビット [15:0]）が使用されます。
*Y* が T の場合は*Rm* の上位ハーフワード（ビット [31:16]）が使用されます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

#### 動作

SMULBB、SMULTB、SMULBT および SMULTT の各命令は、*Rn* と *Rm* の値を 4 つの符号付き 16 ビット整数として解釈します。
これらの命令の場合：

- *Rn* と *Rm* の指定された符号付き（上位または下位）ハーフワード値を乗算します。
- 32 ビットの乗算結果 *Rd* に書き込みます。

SMULWT および SMULWB 命令は、*Rn* の値を 32 ビットの符号付き整数として、*Rm* の値を 2 つのハーフワード（16 ビット）の符号付き整数として解釈します。これらの命令の場合：

- 第 1 オペランドと第 2 オペランドの上位ハーフワード（T 接尾文字付き）、または下位ハーフワード（B 接尾文字付き）を乗算します。
- 48 ビットの結果の符号付き上位 32 ビットをデスティネーション·レジスタに書き込みます。

### 制限事項

これらの命令の場合：

- SP および PC は使用しないでください。
- *RdHi* と *RdLo* は異なるレジスタである必要があります。

### 例

```
SMULBT      R0, R4, R5      ; R4 の下位ハーフワードと R5 の上位ハーフワードを
                            ; 乗算し、結果を乗算して、R0 に書き込む。
SMULBB      R0, R4, R5      ; R4 の下位ハーフワードと R5 の下位ハーフワードを
                            ; 乗算し、結果を乗算して、R0 に書き込む。
SMULTT      R0, R4, R5      ; R4 の上位ハーフワードと R5 の上位ハーフワードを
                            ; 乗算し、結果を乗算して、R0 に書き込む。
SMULTB      R0, R4, R5      ; R4 の上位ハーフワードと R5 の下位ハーフワードを
                            ; 乗算し、結果を乗算して、R0 に書き込む。
SMULWT      R4, R5, R3      ; R3 の上位ハーフワードと R5 を乗算し、
                            ; 上位 32 ビットを抽出して R4 に書き込む。
SMULWB      R4, R5, R3      ; R3 の下位ハーフワードと R5 を乗算し、
                            ; 上位 32 ビットを抽出して R4 に書き込む。
```

### 3.6.11 UMULL、UMLAL、SMULL および SMLAL

32 ビット·オペランドを使用して、64 ビットの結果を生成する、符号付きおよび符号なし Long 乗算と任意に指定できる累算。

#### 構文

*op*{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*

ここで、

*op* これは次のいずれかです。

- UMULL 符号なし Long 乗算。
- UMLAL 符号なし Long 積和。
- SMULL 符号付き Long 乗算。
- SMLAL 符号付き Long 積和。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*RdHi*、*RdLo* これらはデスティネーション・レジスタです。UMLAL および SMLAL の場合は、累算値も保持されます。

*Rn*、*Rm* これらはオペランドを保持するレジスタです。

#### 動作

UMULL 命令は *Rn* と *Rm* の値を符号なし整数として解釈します。これらの整数を乗算して、結果の下位 32 ビットを *RdLo* に格納し、上位 32 ビットを *RdHi* に格納します。

UMLAL 命令は *Rn* と *Rm* の値を符号なし整数として解釈します。これらの整数を乗算して得られた 64 ビットの結果を、*RdHi* と *RdLo* にある 64 ビットの符号なし整数に加算し、結果を *RdHi* と *RdLo* にライトバックします。

SMULL 命令は *Rn* と *Rm* の値を 2の補数の符号付き整数として解釈します。これらの整数を乗算して、結果の下位 32 ビットを *RdLo* に格納し、上位 32 ビットを *RdHi* に格納します。

SMLAL 命令は *Rn* と *Rm* の値を 2の補数の符号付き整数として解釈します。これらの整数を乗算して得られた 64 ビットの結果を、*RdHi* と *RdLo* にある 64 ビットの符号付き整数に加算し、結果を *RdHi* と *RdLo* にライトバックします。

#### 制限事項

これらの命令の場合：

- SP および PC は使用しないでください。
- *RdHi* と *RdLo* は異なるレジスタである必要があります。

#### 条件フラグ

これらの命令は条件コード·フラグに影響しません。

#### 例

```
UMULL       R0, R4, R5, R6   ; 符号なし(R4,R0) = R5 x R6
SMLAL       R4, R5, R3, R8   ; 符号付き (R5,R4) = (R5,R4) + R3 x R8
```

### 3.6.12 SDIV および UDIV

符号付き除算および符号なし除算。

#### 構文

`SDIV{cond} {Rd,} Rn, Rm`

`UDIV{cond} {Rd,} Rn, Rm`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。*Rd* が省略されている場合、デスティネーション・レジスタは *Rn* になります。

*Rn* これは除算される値を保持するレジスタです。

*Rm* これは除数を保持するレジスタです。

#### 動作

SDIV 命令は、*Rn* の値を *Rm* の値で除算する、符号付き整数除算を実行します。

UDIV 命令は、*Rn* の値を *Rm* の値で除算する、符号なし整数除算を実行します。

どちらの命令も、*Rn* の値が *Rm* の値で割り切れない場合は、結果はゼロ方向に丸められます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

#### 例

```
SDIV  R0, R2, R4  ; 符号付き除算、R0 = R2/R4
UDIV  R8, R8, R1  ; 符号なし除算、R8 = R8/R1
```

## 3.7 飽和命令

表 32 に、飽和命令を示します。

**表 32. 飽和命令**

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| SSAT | 符号付き飽和 | 127 ページのSSAT および USAT |
| SSAT16 | 符号付きハーフワード飽和 | 128 ページのSSAT16 および USAT16 |
| USAT | 符号なし飽和 | 127 ページのSSAT および USAT |
| USAT16 | 符号なしハーフワード飽和 | 128 ページのSSAT16 および USAT16 |
| QADD | 飽和加算 | 129 ページのQADD および QSUB |
| QSUB | 飽和減算 | 129 ページのQADD および QSUB |
| QSUB16 | 16 ビット飽和減算 | 129 ページのQADD および QSUB |
| QASX | 飽和加算および減算、交換付き | 130 ページのQASX および QSAX |
| QSAX | 飽和減算および加算、交換付き | 130 ページのQASX および QSAX |
| QDADD | 飽和倍演算および加算 | 131 ページのQDADD および QDSUB |
| QDSUB | 飽和倍演算および減算 | 131 ページのQDADD および QDSUB |
| UQADD16 | 16 ビット符号なし飽和加算 | 133 ページのUQADD および UQSUB |
| UQADD8 | 8 ビット符号なし飽和加算 | 133 ページのUQADD および UQSUB |
| UQASX | 符号なし飽和加算および減算、交換付き | 132 ページのUQASX および UQSAX |
| UQSAX | 符号なし飽和減算および加算、交換付き | 132 ページのUQASX および UQSAX |
| UQSUB16 | 16 ビット符号なし飽和減算 | 133 ページのUQADD および UQSUB |
| UQSUB8 | 8 ビット符号なし飽和減算 | 133 ページのUQADD および UQSUB |

符号付きの *n* ビット·サチュレーションの場合、これは以下を意味します。

- 飽和される値が $-2^{n-1}$ よりも小さい場合、返される結果は $-2^{n-1}$ になります。
- 飽和される値が $2^{n-1}-1$ よりも大きい場合、返される結果は $2^{n-1}-1$ になります。
- それ以外の場合、返される結果は飽和される値と同じになります。

符号なしの *n* ビット·サチュレーションの場合、これは以下を意味します。

- 飽和される値が 0 未満の場合、返される結果は 0 になります。
- 飽和される値が $2^n-1$ よりも大きい場合、返される結果は $2^n-1$ になります。
- それ以外の場合、返される結果は飽和される値と同じになります。

返される結果が飽和される値と異なる場合、これを *サチュレーション*と呼びます。サチュレーションが発生すると、これらの命令によって APSR で Q フラグが 1 にセットされます。そうでない場合、Q フラグは変更されません。Q フラグを 0 にクリアするには、MSR 命令を使用する必要があります。176 ページのMSR を参照してください。

Q フラグの状態を読み出すには、MRS 命令を使用します（175 ページのMRS を参照）。

### 3.7.1 SSAT および USAT

任意のビット位置に対する符号付き飽和および符号なし飽和。飽和の前に任意でシフトします。

#### 構文

*op*{*cond*} *Rd*, #*n*, *Rm* {, *shift* #*s*}

ここで、

*op* これは次のいずれかです。
- SSAT 符号付きの値を符号付き範囲内で飽和します。
- USAT 符号付きの値を符号なし範囲内で飽和します。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。

*n* 飽和させるビット位置を指定します。
- SSAT の場合、*n* の範囲は 1 ~ 32です。
- USAT の場合、*n* の範囲は 0 ~ 31です。

*Rm* これは飽和する値を保持するレジスタです。

*shift #s* これは、飽和の前に、*Rm* に任意で適用されるシフトです。以下のいずれかを指定します。
- ASR #*s* ここで *s* の範囲は 1 ~ 31 です。
- LSL #*s* ここで *s* の範囲は 0 ~ 31 です。

#### 動作

これらの命令は、符号付きまたは符号なし *n* ビット値に飽和します。

SSAT 命令は、指定されたシフトを適用してから、符号付き範囲に飽和させます。
$-2^{n-1} \le x \le 2^{n-1}-1$.

USAT 命令は、指定されたシフトを適用してから、符号なし範囲内 $0 \le x £ 2^{n}-1$ で飽和します。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令は条件コード·フラグに影響しません。

サチュレーションが発生すると、これらの命令によって Q フラグが 1 にセットされます。

#### 例

```
SSAT    R7, #16, R7, LSL #4  ; R7 の値を 4 だけ左に論理シフトし、
                             ; 符号付き 16 ビット値として飽和させて、
                             ; R7 に書き戻します。
USATNE  R0, #7, R5           ; R5 の値を符号なし 7 ビット値として条件付きで
                             ; 飽和させ、R0 に書き込む。
```

### 3.7.2 SSAT16 および USAT16

2 つのハーフワードの任意のビット位置に対する符号付き飽和と符号なし飽和。

#### 構文

*op*{*cond*} *Rd*, #*n*, *Rm*

ここで、

*op* これは次のいずれかです。
SSAT16 符号付きのハーフワード値を符号付き範囲内で飽和します。
USAT16 符号付きのハーフワード値を符号なし範囲内で飽和します。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。

*n* 飽和させるビット位置を指定します。
- SSAT の場合、*n* の範囲は 1 ~ 16です。
- USAT の場合、*n* の範囲は 0 ~ 15です。

*Rm* これは飽和する値を保持するレジスタです。

#### 動作

SSAT16 命令：

1. レジスタの 2 つの符号付き 16 ビット·ハーフワード値を、*n* のビット位置によって選択される値で飽和します。
2. 結果を 2 つの符号付き 16 ビット·ハーフワードとしてデスティネーション·レジスタに書き込みます。

USAT16 命令：

1. レジスタの 2 つの符号なし 16 ビット·ハーフワード値を、*n* のビット位置によって選択される値で飽和します。
2. 結果を 2 つの符号なしハーフワードとしてデスティネーション·レジスタに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令は条件コード·フラグに影響しません。

サチュレーションが発生すると、これらの命令によって Q フラグが 1 にセットされます。

#### 例

```
SSAT16    R7, #9, R2        ; R2 の上位および下位ハーフワードを
                            ; 9 ビット値として飽和させ、R7 の対応する
                            ; ハーフワードに書き込む。
USAT16NE  R0, #13, R5       ; R5 の上位ハーフワードと下位ハーフワードを
                            ; 13 ビット値として条件付きで飽和させ、
                            ; R0 の対応するハーフワードに書き込む。
```

### 3.7.3 QADD および QSUB

符号付き飽和加算および符号付き飽和減算。

#### 構文

`op{cond} {Rd}, Rn, Rm`

`op{cond} {Rd}, Rn, Rm`

ここで、

*op* これは次のいずれかです。

- `QADD` 32 ビットの飽和加算。
- `QADD8` 4 つの 8 ビット整数の飽和加算。
- `QADD16` 2 つの 16 ビット整数の飽和加算。
- `QSUB` 32 ビットの飽和減算。
- `QSUB8` 4 つの 8 ビット整数の飽和減算。
- `QSUB16` 2 つの 16 ビット整数の飽和減算。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

#### 動作

これらの命令は、第 1 および第 2 オペランドの 2、4、または 8 個の値を加算または減算し、飽和された符号付きの値をデスティネーション・レジスタに書き込みます。

QADD 命令と QSUB 命令は、指定されている加算および減算を適用し、その結果を符号付き範囲 $-2^{n-1} \le x \le 2^{n-1}-1$ で飽和します。ここで、*x* は、命令で適用されるビット数（32、16 または 8）です。

返される結果が飽和される値と異なる場合、これを*サチュレーション*と呼びます。サチュレーションが発生すると、QADD および QSUB 命令によって APSR で Q フラグが 1 にセットされます。そうでない場合、Q フラグは変更されません。8 ビットおよび 16 ビットの QADD 命令と QSUB 命令では、Q フラグは常に変更されないままになります。

Q フラグを 0 にクリアするには、MSR 命令を使用する必要があります。176 ページのMSR を参照してください。

Q フラグの状態を読み出すには、MRS 命令を使用します（175 ページのMRS を参照）。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令は条件コード・フラグに影響しません。

サチュレーションが発生すると、これらの命令によって Q フラグが 1 にセットされます。

#### 例

```
QADD16   R7, R4, R2  ; R4 のハーフワードと R2 の対応するハーフワードを加算し、
                     ; 16 ビットに飽和して、R7 の対応するハーフワードに書き込む。
QADD8    R3, R1, R6  ; R1 のバイトを R6 の対応するバイトに加算し、
```

```
                        ; 8 ビットに飽和して、R3 の対応するバイトに書き込む。
QSUB16   R4, R2, R3  ; R2 の対応するハーフワードから R3 のハーフワードを減算して、
                        ; 16 ビットに飽和し、R4 の対応するハーフワードに書き込む。
QSUB8 R4, R2, R5 ; R2 の対応するバイトから R5 のバイトを減算し、
                        ; 8 ビットに飽和して、R4 の対応するバイトに書き込む。
```

### 3.7.4 QASX および QSAX

符号付き飽和加算および減算（交換付き）、および符号付き飽和減算および加算（交換付き）。

#### 構文

`op{cond} {Rd}, Rm, Rn`

ここで、

`op` これは次のいずれかです。
- `QASX` 飽和加算および減算（交換付き）。
- `QSAX` 飽和減算および加算（交換付き）。

*`cond`* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*`Rd`* これはデスティネーション・レジスタです。

*`Rn`*、*`Rm`* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

#### 動作

QASX 命令：

1. ソース・オペランドの上位ハーフワードを第 2 オペランドの下位ハーフワードに加算します。
2. 第 2 オペランドの上位ハーフワードを第 1 オペランドの下位ハーフワードから減算します。
3. 減算結果を飽和し、$-2^{15} \le x \le 2^{15} - 1$ （*x* は 16）の範囲の 16 ビット符号付き整数をデスティネーション・レジスタの下位ハーフワードに書き込みます。
4. 加算結果を飽和し、$-2^{15} \le x \le 2^{15} - 1$ （*x* は 16）の範囲の 16 ビット符号付き整数をデスティネーション・レジスタの上位ハーフワードに書き込みます。

QSAX 命令：

1. 第 2 オペランドの下位ハーフワードを第 1 オペランドの上位ハーフワードから減算します。
2. ソース・オペランドの下位ハーフワードを第 2 オペランドの上位ハーフワードに加算します。
3. 加算結果を飽和し、$-2^{15} \le x \le 2^{15} - 1$ （*x* は 16）の範囲の 16 ビット符号付き整数をデスティネーション・レジスタの下位ハーフワードに書き込みます。
4. 減算結果を飽和し、$-2^{15} \le x \le 2^{15} - 1$ （*x* は 16）の範囲の 16 ビット符号付き整数をデスティネーション・レジスタの上位ハーフワードに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令は条件コード・フラグに影響しません。

### 例

```
QASX    R7, R4, R2      ; R4 の上位ハーフワードを R2 の下位ハーフワードに加算し、
                        ; 16 ビットに飽和して R7 の上位ハーフワードに書き込む。
                        ; R2 の上位ハーフワードを R4 の下位ハーフワードから減算し、
                        ; 16 ビットに飽和して R7 の下位ハーフワードに書き込む。
QSAX    R0, R3, R5      ; R3 の上位ハーフワードから R5 の下位ハーフワードを減算し、
                        ; 16 ビットに飽和して、R0 の上位ハーフワードに書き込む。
                        ; R3 の下位ハーフワードを R5 の上位ハーフワードに加算し、
                        ; 16 ビットに飽和して、R0 の下位ハーフワードに書き込む。
```

## 3.7.5 QDADD および QDSUB

符号付き飽和倍演算と加算、および符号付き飽和倍演算と減算。

### 構文

*op*{*cond*} {*Rd*}, *Rm*, *Rn*

ここで、

*op* これは次のいずれかです。
- QDADD 飽和倍演算および加算。
- QDSUB 飽和倍演算および減算。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rm*、*Rn* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

### 動作

QDADD 命令：

- 第 2 オペランドの値を 2 倍にします。
- 2 倍にした結果を第 1 オペランドの飽和された符号付きの値に加算します。
- 結果をデスティネーション・レジスタに書き込みます。

QDSUB 命令：

- 第 2 オペランドの値を 2 倍にします。
- 2 倍にした値を第 1 オペランドの飽和された符号付きの値から減算します。
- 結果をデスティネーション・レジスタに書き込みます。

倍演算と加算または倍演算と減算のどちらの場合も、結果は、$-2^{31} \le x \le 2^{31}-1$ の範囲の 32 ビットの符号付き整数に飽和されます。いずれかの演算でサチュレーションが発生すると、APSR で Q フラグがセットされます。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

サチュレーションが発生すると、これらの命令によって Q フラグが 1 にセットされます。

### 例

```
QDADD   R7, R4, R2          ; R4 を 2 倍にして 32 ビットに飽和し、R2 を加算する。
                            ; 32 ビットに飽和して、R7 に書き込む。
QDSUB   R0, R3, R5          ; R3 を 2 倍にして飽和し R5 から減算し、
                            ; 32 ビットに飽和して R0 に書き込む。
```

## 3.7.6 UQASX および UQSAX

符号なし飽和加算および減算（交換付き）、および符号なし飽和減算および加算（交換付き）。

### 構文

`op{cond} {Rd}, Rm, Rn`

ここで、

*タイプ* これは次のいずれかです。
- UQASX 飽和加算および減算（交換付き）。
- UQSAX 飽和減算および加算（交換付き）。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

### 動作

UQASX 命令：

1. ソース・オペランドの下位ハーフワードを第 2 オペランドの上位ハーフワードに加算します。
2. 第 2 オペランドの下位ハーフワードを第 1 オペランドの上位ハーフワードから減算します。
3. 加算結果を飽和し、$0 \le x \le 2^{16} - 1$ （$x$ は 16）の範囲の 16 ビット符号なし整数をデスティネーション・レジスタの上位ハーフワードに書き込みます。
4. 減算結果を飽和し、$0 \le x \le 2^{16} - 1$ （$x$ は 16）の範囲の 16 ビット符号なし整数をデスティネーション・レジスタの下位ハーフワードに書き込みます。

UQSAX 命令：

1. 第 2 オペランドの下位ハーフワードを第 1 オペランドの上位ハーフワードから減算します。
2. 第 1 オペランドの下位ハーフワードを第 2 オペランドの上位ハーフワードに加算します。
3. 減算結果を飽和し、$0 \le x \le 2^{16} - 1$ （$x$ は 16）の範囲の 16 ビット符号なし整数をデスティネーション・レジスタの上位ハーフワードに書き込みます。
4. 加算結果を飽和し、$0 \le x \le 2^{16} - 1$ （$x$ は 16）の範囲の 16 ビット符号なし整数をデスティネーション・レジスタの下位ハーフワードに書き込みます。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

これらの命令は条件コード・フラグに影響しません。

### 例

```
UQASX   R7, R4, R2  ; R4 の上位ハーフワードと R2 の下位ハーフワードを加算し、
                    ; 16 ビットに飽和して、R7 の上位ハーフワードに書き込む。
                    ; R2 の上位ハーフワードを R4 の下位ハーフワードから減算し、
                    ; 16 ビットに飽和して、R7 の下位ハーフワードに書き込む。
UQSAX   R0, R3, R5  ; R5 の下位ハーフワードを R3 の上位ハーフワードから減算し、
                    ; 16 ビットに飽和して、R0 の上位ハーフワードに書き込む。
                    ; R4 の下位ハーフワードを R5 の上位ハーフワードに加算し、
                    ; 16 ビットに飽和して、R0 の下位ハーフワードに書き込む
```

## 3.7.7 UQADD および UQSUB

符号なし飽和加算および符号なし飽和減算。

### 構文

`op{cond} {Rd}, Rn, Rm`

`op{cond} {Rd}, Rn, Rm`

ここで、

*op* これは次のいずれかです。

- `UQADD8` 4 つの 8 ビット符号なし整数の飽和加算。
- `UQADD16` 2 つの 16 ビット符号なし整数の飽和加算。
- `UDSUB8` 4 つの 8 ビット符号なし整数の飽和減算。
- `UQSUB16` 2 つの 16 ビット符号なし整数の飽和減算。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn*、*Rm* これらは第 1 オペランドおよび第 2 オペランドを保持するレジスタです。

### 動作

これらの命令は、2 つまたは 4 つの値を加算または減算し、符号なしの飽和された値をデスティネーション・レジスタに書き込みます。

UQADD16 命令：

- 第 1 オペランドと第 2 オペランドの上位ハーフワード同士および下位ハーフワード同士を加算します。
- デスティネーション・レジスタにある各ハーフワードの加算結果を符号なし範囲 $0 \le x \le 2^{16}-1$（$x$ は 16）に飽和します。

UQADD8 命令：

- 第 1 オペランドと第 2 オペランドのそれぞれ対応するバイトを加算します。
- デスティネーション・レジスタにある各バイトの加算結果を符号なし範囲 $0 \le x \le 2^{8}-1$ （$x$ は 8）に飽和します。

UQSUB16 命令：

- 第 2 オペランドの両ハーフワードを第 1 オペランドのそれぞれ対応するハーフワードから減算します。
- デスティネーション・レジスタにある減算結果を符号なし範囲 $0 \le x \le 2^{16}-1$ （$x$ は 16）に飽和します。

UQSUB8 命令：

- 第 2 オペランドのそれぞれのバイトを第 1 オペランドの対応するバイトから減算します。
- デスティネーション·レジスタ異なるある各バイトの減算結果を符号なし範囲 $0 \leq x \leq 2^8-1$ （$x$ は 8）に飽和します。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

これらの命令は条件コード·フラグに影響しません。

### 例

```
UQADD16   R7, R4, R2    ; R4 のハーフワードを R2 の対応するハーフワードに加算し、
                        ; 16 ビットに飽和して、R7 の対応するハーフワードに書き込む。
UQADD8    R4, R2, R5    ; R2 のバイトを R5 の対応するバイトに加算し、
                        ; 8 ビットに飽和して、R4 の対応するバイトに書き込む。
UQSUB16   R6, R3, R0    ; R3 の対応するハーフワードから R0 のハーフワードを減算し、
                        ; 16 ビットに飽和して、R6 の対応するハーフワードに書き込む。
UQSUB8    R1, R5, R6    ; R5 の対応するバイトから R6 のバイトを減算し、
                        ; 8 ビットに飽和して、R1 の対応するバイトに書き込む。
```

## 3.8 パック命令と展開命令

表 33 は、データのパックおよび展開で動作する命令を示しています。

**表 33. パック命令と展開命令**

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| PKH | ハーフワード·パック | 135 ページのPKHBT および PKHTB |
| SXTAB | 8 ビット値を 32 ビット値に拡張して加算 | 137 ページのSXTA および UXTA |
| SXTAB16 | デュアルで 8 ビット値を 16 ビット値に拡張して加算 | 137 ページのSXTA および UXTA |
| SXTAH | 16 ビット値を 32 ビット値に拡張して加算 | 137 ページのSXTA および UXTA |
| SXTB | 符号拡張（バイト） | 141 ページのSXT および UXT |
| SXTB16 | デュアルで 8 ビット値を 16 ビット値に拡張して加算 | 141 ページのSXT および UXT |
| SXTH | 符号拡張（ハーフワード） | 141 ページのSXT および UXT |
| UXTAB | 8 ビット値を 32 ビット値に拡張して加算 | 137 ページのSXTA および UXTA |
| UXTAB16 | デュアルで 8 ビット値を 16 ビット値に拡張して加算 | 137 ページのSXTA および UXTA |
| UXTAH | 16 ビット値を 32 ビット値に拡張して加算 | 137 ページのSXTA および UXTA |
| UXTB | ゼロ拡張（バイト） | 141 ページのSXT および UXT |
| UXTB16 | デュアルで 8 ビット値を 16 ビット値にゼロ拡張して加算 | 141 ページのSXT および UXT |
| UXTH | ゼロ拡張（ハーフワード） | 141 ページのSXT および UXT |

### 3.8.1 PKHBT および PKHTB

ハーフワード･パック。

#### 構文

`op{cond} {Rd}, Rn, Rm {, LSL #imm}`

`op{cond} {Rd}, Rn, Rm {, ASR #imm}`

ここで、

`op` これは次のいずれかです。

- `PKHBT` 下位ハーフワードと上位ハーフワードをパックします（シフト付き）。
- `PKHTB` 上位ハーフワードと下位ハーフワードをパックします（シフト付き）。

*`cond`* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*`Rd`* これはデスティネーション･レジスタです。

*`Rn`* これは第 1 オペランド･レジスタです。

*`Rm`* これは任意でシフトされる値を保持する第 2 オペランド･レジスタです。

*`imm`* これはシフト長です。シフト長のタイプは、以下のように命令に応じて異なります。

`PKHBT` の場合

- `LSL` シフト長が 1 ~ 31 の左シフト。0 はシフトなしを意味します。

`PKHTB` の場合：

- `ASR` 1 から 32 のシフト長の算術右シフト
  32 ビットのシフトは、`0b00000` とエンコードされます。

#### 動作

PKHBT 命令：

1. 第 1 オペランドの下位ハーフワードの値をデスティネーション･レジスタの下位ハーフワードに書き込みます。
2. シフトされる場合は、第 2 オペランドのシフトされた値がデスティネーション･レジスタの上位ハーフワードに書き込まれます。

PKHTB 命令：

1. 第 1 オペランドの上位ハーフワードの値をデスティネーション･レジスタの上位ハーフワードに書き込みます。
2. シフトされる場合は、第 2 オペランドのシフトされた値がデスティネーション･レジスタの下位ハーフワードに書き込まれます。

#### 制限事項

*Rd* に SP または PC は使用できません。

#### 条件フラグ

この命令によるフラグの変更はありません。

### 例

```
PKHBT   R3, R4, R5 LSL #0  ; R4 の下位ハーフワードを R3 の下位ハーフワードに書き込む。
                           ; R5 の上位ハーフワードをシフトしないで
                           ; R3 の上位ハーフワードに書き込む。
PKHTB   R4, R0, R2 ASR #1  ; R2 を 1 ビット右にシフトして R4 の下位ハーフワードに
                           ; 書き込み、R0 の上位ハーフワードを R4 の
                           ; 上位ハーフワードに書き込む。
```

## 3.8.2 SXT および UXT

符号付き拡張およびゼロ拡張。

### 構文

`op{cond} {Rd,} Rm {, ROR #n}`

`op{cond} {Rd}, Rm {, ROR #n}`

ここで、

*op* これは次のいずれかです。
- `SXTB` 8 ビット値を 32 ビット値に符号拡張。
- `SXTH` 16 ビット値を 32 ビット値に符号拡張。
- `SXTB16` 2 つの 8 ビット値を 2 つの 16 ビット値に符号拡張。
- `UXTB` 8 ビット値を 32 ビット値にゼロ拡張。
- `UXTH` 16 ビット値を 32 ビット値にゼロ拡張。
- `UXTB16` 2 つの 8 ビット値を 2 つの 16 ビット値にゼロ拡張。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rm* これは拡張する値を保持するレジスタです。

`ROR #n` これは次のいずれかです。
- `ROR #8` *Rm* の値が 8 ビット右にローテートされます。
- `ROR #16` *Rm* の値が 16 ビット右にローテートされます。
- `ROR #24` *Rm* の値が 24 ビット右にローテートされます。

`ROR #n` が省略されている場合、ローテートは行われません。

### 動作

これらの命令は以下の処理を行います。

1. *Rm* の値を 0、8、16 または 24 ビット右にローテートします。
2. 結果として得た値からビットを抽出します。
   - SXTB はビット [7:0] を抽出し、32 ビットに符号拡張します。
   - UXTB はビット [7:0] を抽出し、32 ビットにゼロ拡張します。
   - SXTH はビット [15:0] を抽出し、32 ビットに符号拡張します。
   - UXTH はビット [15:0] を抽出し、32 ビットにゼロ拡張します。
   - SXTB16 はビット [7:0] を抽出し、16 ビットに符号拡張します。
     さらに、ビット [23:16] を抽出し、16 ビットに符号拡張します。
   - UXTB16 はビット [7:0] を抽出し、16 ビットにゼロ拡張します。
     さらに、ビット [23:16] を抽出し、16 ビットにゼロ拡張します。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

これらの命令によるフラグへの影響はありません。

### 例

```
SXTH   R4, R6, ROR #16   ; R6 を 16 ビット右にローテートして結果の下位ハーフワードを
                         ;  取得し、32 ビットに符号拡張して R4 に書き込む。
UXTB   R3, R10           ; R10 の値の最下位バイトを抽出し、ゼロ拡張して、
                         ; R3 に書き込む。
```

## 3.8.3 SXTA および UXTA

符号拡張と加算、および符号なし拡張と加算。

### 構文

*op*{*cond*} {*Rd*,} *Rn*, *Rm* {, ROR #*n*}

*op*{*cond*} {*Rd*,} *Rn*, *Rm* {, ROR #*n*}

ここで、

*op* これは次のいずれかです。
- SXTAB 8 ビット値を 32 ビット値に符号拡張して加算。
- SXTAH 16 ビット値を 32 ビット値に符号拡張して加算。
- SXTAB16 2 つの 8 ビット値を 2 つの 16 ビット値に符号拡張して加算。
- UXTAB 8 ビット値を 32 ビット値にゼロ拡張して加算。
- UXTAH 16 ビット値を 32 ビット値にゼロ拡張して加算。
- UXTAB16 2 つの 8 ビット値を 2 つの 16 ビット値にゼロ拡張して加算。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rn* これは第 1 オペランド・レジスタです。

*Rm* これはローテートおよび拡張する値を保持するレジスタです。

ROR #*n* これは次のいずれかです。
- ROR #8 *Rm* の値が 8 ビット右にローテートされます。
- ROR #16 *Rm* の値が 16 ビット右にローテートされます。
- ROR #24 *Rm* の値が 24 ビット右にローテートされます。
- ROR #*n* が省略されている場合、ローテートは行われません。

### 動作

これらの命令は以下の処理を行います。

1. *Rm* の値を 0、8、16 または 24 ビット右にローテートします。
2. 結果として得た値からビットを抽出します。
   - SXTAB は *Rm* からビット [7:0] を抽出し、32 ビットに符号拡張します。
   - UXTAB は *Rm* からビット [7:0] を抽出し、32 ビットにゼロ拡張します。
   - SXTAH は *Rm* からビット [15:0] を抽出し、32 ビットに符号拡張します。

- UXTAH は *Rm* からビット [15:0] を抽出し、32 ビットにゼロ拡張します。
- SXTAB16 は *Rm* からビット [7:0] を抽出し、16 ビットに符号拡張します。
  さらに、*Rm* からビット [23:16] を抽出し、16 ビットに符号拡張します。
- UXTAB16 は *Rm* からビット [7:0] を抽出し、16 ビットにゼロ拡張します。
  さらに、*Rm* からビット [23:16] を抽出し、16 ビットにゼロ拡張します。

3. 符号拡張またはゼロ拡張した値を *Rn* のワードまたは対応するハーフワードに加算し、結果を *Rd* に書き込みます。

### 制限事項

SP および PC は使用しないでください。

### 条件フラグ

これらの命令によるフラグへの影響はありません。

#### 例

```
SXTAH  R4, R8, R6, ROR #16 ; R6 を右に 16 ビットローテートして
                           ; 下位ハーフワードを取得し、32 ビットに符号拡張して
                           ; R8 を加算し、R4 に書き込む。
UXTAB  R3, R4, R10         ; R10 の最下位バイトを抽出し、32 ビットにゼロ拡張し、
                           ; R4 を加算して R3 に書き込む。
```

## 3.9 ビット・フィールドの命令

表 34 は、レジスタまたはビット・フィールドの隣接するビット・セットで動作する命令を示しています。

**表 34. パック命令と展開命令**

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| BFC | ビット・フィールドのクリア | 139 ページのBFC および BFI |
| BFI | ビット・フィールドの挿入 | 139 ページのBFC および BFI |
| SBFX | 符号付きビット・フィールド拡張 | 140 ページのSBFX および UBFX |
| SXTB | 符号拡張（バイト） | 141 ページのSXT および UXT |
| SXTH | 符号拡張（ハーフワード） | 141 ページのSXT および UXT |
| UBFX | 符号なしビット・フィールド拡張 | 140 ページのSBFX および UBFX |
| UXTB | ゼロ拡張（バイト） | 141 ページのSXT および UXT |
| UXTH | ゼロ拡張（ハーフワード） | 141 ページのSXT および UXT |

### 3.9.1 BFC および BFI

ビット･フィールドのクリアおよびビット･フィールドの挿入。

#### 構文

`BFC{cond} Rd, #lsb, #width`

`BFI{cond} Rd, Rn, #lsb, #width`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション･レジスタです。

*Rn* これはソース･レジスタです。

*lsb* これはビット・フィールドの最下位ビットの位置です。*lsb* は0 ~ 31 の範囲内の値である必要があります。

*幅* これはビット･フィールドの幅で、1 ~ 32−*lsb* の範囲とする必要があります。

#### 動作

BFC はレジスタ内のビット･フィールドをクリアします。*Rd* の下位ビット位置 *lsb* から *width* ビット分のビットをクリアします。*Rd* 内の他のビットは変更されません。

BFI は、あるレジスタから別のレジスタにビット･フィールドをコピーします。*Rd* の下位ビット位置 *lsb* から始まる *width* 分のビットが、*Rn* のビット [0] から始まる *width* 分のビットによって置き換えられます。*Rd* 内の他のビットは変更されません。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令によるフラグへの影響はありません。

#### 例

```
BFC   R4, #8, #12        ; R4 のビット 8 ~ 19 (12 ビット) を 0 にクリア
BFI   R9, R2, #8, #12    ; R9 のビット 8 ~ 19 (12 ビット) を
                         ; R2 のビット 0 からビット 11 で置き換えます。
```

### 3.9.2 SBFX および UBFX

符号付きビット･フィールドの拡張および符号なしビット･フィールドの拡張。

#### 構文

`SBFX{cond} Rd, Rn, #lsb, #width`

`UBFX{cond} Rd, Rn, #lsb, #width`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション･レジスタです。

*Rn* これはソース･レジスタです。

*lsb* これはビット・フィールドの最下位ビットの位置です。*lsb* は0 ~ 31 の範囲内の値である必要があります。

*幅* これはビット･フィールドの幅で、1 ~ 32−*lsb* の範囲とする必要があります。

#### 動作

SBFX はあるレジスタからビット･フィールドを抽出し、32 ビットに符号拡張して、結果をデスティネーション･レジスタに書き込みます。

UBFX はあるレジスタからビット･フィールドを抽出し、32 ビットにゼロ拡張して、結果をデスティネーション･レジスタに書き込みます。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令によるフラグへの影響はありません。

#### 例

```
SBFX  R0, R1, #20, #4  ; R1 のビット 20 ~ ビット 23（4 ビット）を抽出し、
                       ; 32 ビットに符号拡張して、結果を R0 に書き込む。
UBFX  R8, R11, #9, #10 ; R11 のビット 9 からビット 18（10 ビット）を抽出し、
                       ; 32 ビットにゼロ拡張して、結果を R8 に書き込む。
```

### 3.9.3 SXT および UXT

符号付き拡張およびゼロ拡張。

#### 構文

`SXT`*extend*`{`*cond*`} {`*Rd*`,} `*Rm*` {, ROR #`*n*`}`

`UXT`*extend*`{`*cond*`} {`*Rd*`}, `*Rm*` {, ROR #`*n*`}`

ここで、

*extend* これは次のいずれかです。
- B 8 ビット値を 32 ビット値に拡張。
- H 16 ビット値を 32 ビット値に拡張。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション・レジスタです。

*Rm* これは拡張する値を保持するレジスタです。

ROR #*n* これは次のいずれかです。
- ROR #8 *Rm* の値が 8 ビット右にローテートされます。
- ROR #16 *Rm* の値が 16 ビット右にローテートされます。
- ROR #24 *Rm* の値が 24 ビット右にローテートされます。
- ROR #*n* が省略されている場合、ローテートは行われません。

#### 動作

これらの命令は以下の処理を行います。

1. *Rm* の値を 0、8、16 または 24 ビット右にローテートします。
2. 結果として得た値からビットを抽出します。
   - SXTB はビット [7:0] を抽出し、32 ビットに符号拡張します。
   - UXTB はビット [7:0] を抽出し、32 ビットにゼロ拡張します。
   - SXTH はビット [15:0] を抽出し、32 ビットに符号拡張します。
   - UXTH はビット [15:0] を抽出し、32 ビットにゼロ拡張します。

#### 制限事項

SP および PC は使用しないでください。

#### 条件フラグ

これらの命令によるフラグへの影響はありません。

#### 例

```
SXTH  R4, R6, ROR #16  ; R6 を 16 ビット右にローテートし、
                       ; 結果の下位ハーフワードを取得し、
                       ; 32 ビットに符号拡張して結果を R4 に書き込む。
UXTB  R3, R10          ; R10 の値の最下位バイトを抽出してゼロ拡張して、
                       ; 結果を R3 に書き込む。
```

## 3.10 分岐命令と制御命令

表 35 は分岐命令と制御命令を示しています。

**表 35. 分岐命令と制御命令**

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| B | 分岐 | 142 ページのB、BL、BX、および BLX |
| BL | リンク付き分岐 | 142 ページのB、BL、BX、および BLX |
| BLX | リンク付き間接分岐 | 142 ページのB、BL、BX、および BLX |
| BX | 間接分岐 | 142 ページのB、BL、BX、および BLX |
| CBNZ | ゼロとの比較と分岐（ゼロでない場合に分岐） | 144 ページのCBZ および CBNZ |
| CBZ | ゼロとの比較と分岐（ゼロの場合に分岐） | 144 ページのCBZ および CBNZ |
| IT | If-Then 命令 | 145 ページのIT |
| TBB | テーブル分岐（バイト） | 147 ページのTBB および TBH |
| TBH | テーブル分岐（ハーフワード） | 147 ページのTBB および TBH |

### 3.10.1 B、BL、BX、および BLX

分岐命令。

#### 構文

`B{cond} label`

`BL{cond} label`

`BX{cond} Rm`

`BLX{cond} Rm`

ここで、

`B` これは分岐（イミディエート）です。

`BL` これはリンク付き分岐（イミディエート）です。

`BX` これは間接分岐（レジスタ）です。

`BLX` これはリンク付き間接分岐（レジスタ）です。

*`cond`* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*`label`* これは PC 相対式です。67 ページのPC 相対式を参照してください。

*`Rm`* これは分岐先アドレスを示すレジスタです。*`Rm`* でこの値のビット [0] は 1 になっている必要があります。ただし、分岐先アドレスはビット [0] を 0 に変更することによって作成されます。

#### 動作

これらすべての命令は、*label* への分岐または *Rm* で示されているアドレスへの分岐を発生させます。さらに、以下の処理を行います。

- BL 命令と BLX 命令は、次の命令のアドレスを LR（R14：リンクレジスタ）に書き込みます。
- BX 命令と BLX 命令は、*Rm* のビット [0] が 0 の場合、UsageFault 例外を発生させます。

B*cond* label は、IT ブロックの内外どちらでも使用できる唯一の条件付き命令です。その他すべての分岐命令は、IT ブロック内で条件付き、IT ブロック外で無条件とする必要があります（145 ページの IT を参照）。

表 36 は、各種分岐命令の範囲を示しています。

**表 36. 分岐範囲**

| 命令 | 分岐範囲 |
|---|---|
| B label | –16MB ~ +16MB |
| B*cond* label （IT ブロック外） | –1MB ~ +1MB |
| B*cond* label （IT ブロック内） | –16MB ~ +16MB |
| BL{*cond*} label | –16MB ~ +16MB |
| BX{*cond*} Rm | レジスタ内の任意の値 |
| BLX{*cond*} Rm | レジスタ内の任意の値 |

最大の分岐範囲を得るため、.W 接尾文字を使用する必要がある場合があります。70 ページの命令の幅の選択を参照してください。

## 制限事項

次の制限事項があります。

- BLX 命令では PC は使用しないでください。
- BX および BLX の場合、正常に実行するには、*Rm* のビット [0] は 1 である必要がありますが、分岐は、ビット [0] を 0 に変更することによって作成されたターゲット・アドレスに発生します。
- これらの命令のいずれかが IT ブロック内にある場合、その命令は IT ブロック内の最後の命令である必要があります。

B*cond* は IT ブロック内にあることを必要としない唯一の条件付き命令です。ただし、IT ブロック内にある場合は、分岐の範囲が長くなります。

## 条件フラグ

これらの命令によるフラグの変更はありません。

## 例

```
B       loopA   ; loopA に分岐
BLE     ng      ; ラベル ng に条件付きで分岐
B.W     target  ; 16MB の範囲内でターゲットに分岐
BEQ     target  ; ターゲットに条件付きで分岐
BEQ.W   target  ; 1MB 内でターゲットに条件付きで分岐
BL      funC    ; 関数 funC にリンク付きで分岐し（呼出し）、
                ; LR に保存されているアドレスを返す。
BX      LR      ; 関数呼出しから戻る
BXNE    R0      ; R0 に保存されているアドレスに条件付きで分岐
BLX     R0      ; R0 に保存されているアドレスにリンク付きで分岐し、
                ; このアドレスに切替え。
```

### 3.10.2 CBZ および CBNZ

ゼロと比較し、ゼロで分岐、またはゼロでない場合に分岐。

#### 構文

```
CBZ Rn, label

CBNZ Rn, label
```

ここで、

*Rn* これはオペランドを保持するレジスタです。
*label* これは分岐のデスティネーションです。

#### 動作

CBZ 命令または CBNZ 命令を使用して、条件コード·フラグの変更を防止し、命令数を削減することができます。

`CBZ Rn, label` は、条件フラグは変更されませんが、それ以外については以下と同じ意味です。

```
    CMP     Rn, #0
    BEQ     label
```

`CBNZ Rn, label` は、条件フラグは変更されませんが、それ以外については以下と同じ意味です。

```
    CMP     Rn, #0
    BNE     label
```

#### 制限事項

次の制限事項があります。

- *Rn* は R0 ~ R7 の範囲とする必要があります。
- 分岐のデスティネーションは、命令後の 4 ~ 130 バイト内とする必要があります。
- これらの命令は IT ブロック内では使用できません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

#### 例

```
    CBZ     R5, target ; R5 がゼロの場合に前方分岐
    CBNZ    R0, target ; R0 がゼロ以外の場合に前方分岐
```

### 3.10.3 IT

If-Then 条件付き命令。

#### 構文

```
IT{x{y{z}}} cond
```

ここで、

*x* これは IT ブロックで 2 番目の命令の条件スイッチを指定します。
*y* これは IT ブロックで 3 番目の命令の条件スイッチを指定します。
*z* これは IT ブロックで 4 番目の命令の条件スイッチを指定します。
*cond* これは IT ブロックで最初の命令の条件を指定します。

IT ブロックで 2 ~ 4 番目の命令の条件スイッチは、以下のいずれかにすることができます。

T Then。条件 *cond* を命令に適用します。
E Else。*cond* の逆条件を命令に適用します。

IT 命令の *cond* には、AL （*always* 条件）を使用できます。使用する場合、IT ブロック内のすべての命令を無条件にし、*x*、*y*、および *z* にはそれぞれ、*T* を指定するか省略する必要があります（E にはしないこと）。

#### 動作

IT 命令は後続の最大 4 個の命令を条件付き命令にします。条件はすべて同じにすることも、論理的に逆の条件を混在させることも可能です。IT 命令の後続の条件付き命令は、*IT block* を形成します。

IT ブロック内の命令では、分岐も含めて、構文の {*cond*} 部分に条件を指定する必要があります。

IT 命令を自身で記述しなくてよいように、アセンブラが条件付き命令に必要な IT 命令を自動的に生成できる場合があります。詳細については、アセンブラに関するドキュメントを参照してください。

IT ブロック内の BKPT 命令は、条件が失敗しても、常に実行されます。

IT 命令とそれに対応する IT ブロックの間、または IT ブロックの内部で例外が発生する場合があります。例外が発生すると、適切な例外ハンドラが開始され、適切な復帰情報が LR と SPSR に格納されます。

例外からの復帰に使用する目的で設計された命令を通常どおり使用して、例外から復帰することができ、IT ブロックの実行が正常に再開されます。これは、プログラム･カウンタを変更する命令が、IT ブロック内の命令に分岐する唯一の方法です。

#### 制限事項

IT ブロック内では、以下の命令を使用できません。

- IT
- CBZ および CBNZ
- CPSID および CPSIE

IT ブロック使用時のその他の制限事項は、以下のとおりです。

- 分岐または PC を変更する命令は、IT ブロック外で指定されているか、IT ブロック内の最後の命令である必要があります。これらの命令は、以下のとおりです。
  - ADD PC, PC, Rm
  - MOV PC, Rm
  - B、BL、BX、BLX
  - PC に書込みを行う、任意の LDM、LDR、または POP 命令
  - TBB および TBH
- 例外ハンドラから復帰する場合を除き、IT ブロック内の命令には分岐しないでください。
- B*cond* 以外のすべての条件付き命令は、IT ブロック内に指定する必要があります。B*cond* は IT ブロックの内外のどちらでも使用できますが、ブロック内で使用する場合、分岐範囲が長くなります。
- IT ブロック内の各命令では、条件コード接尾文字を指定する必要があります。これは、ブロック内の他の命令と同じにすることも、論理的に逆にすることも可能です。

IT ブロック内でのアセンブラ·ディレクティブの使用禁止など、アセンブラが IT ブロックの使用に関する制限事項を追加で指定する場合もあります。

## 条件フラグ

この命令によるフラグの変更はありません。

## 例

```
ITTE   NE           ; 次の 3 つの命令は条件付き
ANDNE  R0, R0, R1    ; ANDNE は条件フラグを更新しない
ADDSNE R2, R2, #1    ; ADDSNE は条件フラグを更新する
MOVEQ  R2, R3        ; 条件付き転送


CMP    R0, #9        ; R0 16進値（0 ~ 15）を ASCII
                     ; ('0' ~ '9'、'A' ~ 'F'）に変換
ITE    GT            ; 次の 2 つの命令は条件付き
ADDGT  R1, R0, #55   ; 0xA -> 'A' に変換
ADDLE  R1, R0, #48   ; 0x0 -> '0' に変換


IT     GT            ; 1 つだけ条件付き命令を含む IT ブロック
ADDGT  R1, R1, #1    ; R1 を条件付きでインクリメント


ITTEE  EQ            ; 次の 4 つの命令は条件付き
MOVEQ  R0, R1        ; 条件付き転送
ADDEQ  R2, R2, #10   ; 条件付き加算
ANDNE  R3, R3, #1    ; 条件付き AND
BNE.W  dloop         ; 分岐命令は、IT ブロックの最後の命令でのみ使用できます。


IT     NE            ; 次の命令は条件付き
ADD    R0, R0, R1    ; 構文エラー。IT ブロックに条件コードが使用されていない
```

### 3.10.4 TBB および TBH

テーブル分岐（バイト）とテーブル分岐（ハーフワード）。

#### 構文

`TBB [Rn, Rm]`

`TBH [Rn, Rm, LSL #1]`

ここで、

*Rn* これは分岐長テーブルのアドレスを保持するレジスタです。

*Rn* が PC の場合は、テーブルのアドレスは `TBB` または `TBH` 命令の直後のバイトのアドレスになります。

*Rm* これはインデクス·レジスタです。これには、テーブル内のインデクスが保持されます。ハーフワード·テーブルの場合、`LSL #1` は *Rm* の値を 2 倍にし、テーブルに右オフセットを設定します。

#### 動作

これらの命令により、シングル·バイト·オフセット（TBB）またはハーフワード·オフセット（TBH）のテーブルを使用した PC 相対の前方分岐が発生します。*Rn* はテーブルへのポインタを提供し、*Rm* はテーブル内のインデクスを提供します。TBB の場合、分岐オフセットは、テーブルから返されたバイトの符号なし値の 2 倍になります。TBH の場合、分岐オフセットは、テーブルから返されたハーフワードの符号なし値の 2 倍になります。分岐は、TBB 命令または TBH 命令直後のバイトのアドレスからオフセットした位置でのアドレスに発生します。

#### 制限事項

次の制限事項があります。

- *Rn* は SP にはできません。
- *Rm* に SP または PC は指定できません。
- これらの命令のいずれかが IT ブロック内で使用される場合、その命令は IT ブロック内の最後の命令である必要があります。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

#### 例

```
    ADR.W  R0, BranchTable_Byte
    TBB    [R0, R1]          ; R1 は分岐テーブルのインデクス、R0 はベース·アドレス
                             ;
Case1
; 命令シーケンスが続く
Case2
; 命令シーケンスが続く
Case3
; 命令シーケンスが続く
BranchTable_Byte
    DCB    0                 ; Case1 オフセットの計算
    DCB    ((Case2-Case1)/2) ; Case2 オフセットの計算
    DCB    ((Case3-Case1)/2) ; Case3 オフセットの計算
```

```
    TBH    [PC, R1, LSL #1]    ; R1 は分岐テーブルのインデクス、
                               ; PC は分岐テーブルのベースとして使用。
BranchTable_H
    DCW    ((CaseA - BranchTable_H)/2)   ; CaseA オフセットの計算
    DCW    ((CaseB - BranchTable_H)/2)   ; CaseB オフセットの計算
    DCW    ((CaseC - BranchTable_H)/2)   ; CaseC オフセットの計算

CaseA
; 命令シーケンスが続く
CaseB
; 命令シーケンスが続く
CaseC
; 命令シーケンスが続く
```

## 3.11 浮動小数点命令

このセクションでは、単精度および倍精度 FPU が使用する命令セットについて説明します。

表 37 に、浮動小数点命令を示します。

これらの命令は、システムに FPU が含まれ、有効である場合にのみ使用できます。浮動小数点ユニットの有効化については、234 ページのFPU の有効化を参照してください。

**表 37. 浮動小数点命令**

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| VABS | 浮動小数点の絶対値 | 150 ページのVABS |
| VADD | 浮動小数点加算 | 150 ページのVADD |
| VCMP | 2 つの浮動小数点レジスタ、または 1 つの浮動小数点レジスタとゼロを比較。 | 151 ページのVCMP、VCMPE |
| VCMPE | 無効演算チェックで、2 つの浮動小数点レジスタ、または 1 つの浮動小数点レジスタとゼロを比較 | 151 ページのVCMP、VCMPE |
| VCVT | 浮動小数点数と整数との変換 | 153 ページのVCVT（浮動小数点数と固定小数点間） |
| VCVT | 浮動小数点数と固定小数点数との変換 | 153 ページのVCVT（浮動小数点数と固定小数点間） |
| VCVTR | 丸めを使った浮動小数点数と整数との変換 | 152 ページのVCVT、VCVTR（浮動小数点数と整数間） |
| VCVTB | 半精度値から単精度値への変換 | 154 ページのVCVTB、VCVTT |
| VCVTT | 単精度レジスタから半精度レジスタへの変換 | 154 ページのVCVTB、VCVTT |
| VDIV | 浮動小数点除算 | 154 ページのVDIV |
| VFMA | 結合浮動小数点積和 | 155 ページのVFMA、VFMS |
| VFNMA | 浮動小数点結合積和の結果を符号反転 | 156 ページのVFNMA、VFNMS |
| VFMS | 浮動小数点結合積差 | 155 ページのVFMA、VFMS |
| VFNMS | 浮動小数点結合乗算を減算した結果を符号反転 | 156 ページのVFNMA、VFNMS |
| VLDM | 拡張レジスタの多重ロード | 156 ページのVLDM |
| VLDR | メモリからの拡張レジスタ·ロード | 157 ページのVLDR |
| VMLA | 浮動小数点積和 | 158 ページのVMLA、VMLS |
| VMLS | 浮動小数点積差 | 158 ページのVMLA、VMLS |

表 37. 浮動小数点命令 （続き）

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| VMOV | 浮動小数点のイミディエート値の転送 | 159 ページのVMOV （イミディエート） |
| VMOV | 浮動小数点レジスタ間の転送 | 159 ページのVMOV レジスタ |
| VMOV | Arm コア･レジスタ値を単精度レジスタにコピー | 160 ページのVMOV（Arm コア･レジスタと単精度レジスタの間） |
| VMOV | 2 つのArm コア･レジスタ値を 2 つの単精度レジスタにコピー | 161 ページのVMOV（2 つの Arm コア･レジスタと 2 つの単精度レジスタの間） |
| VMOV | Arm コア･レジスタ値をスカラにコピー | 162 ページのVMOV（Arm コア･レジスタからスカラへ） |
| VMOV | スカラ値を Arm コア･レジスタにコピー | 160 ページのVMOV（スカラから Arm コア･レジスタへ） |
| VMRS | 浮動小数点システム･レジスタから Arm コア･レジスタに転送 | 162 ページのVMRS |
| VMSR | Arm コア･レジスタから 浮動小数点システム･レジスタに転送 | 163 ページのVMSR |
| VMUL | 浮動小数点乗算 | 163 ページのVMUL |
| VNEG | 浮動小数点数無効 | 164 ページのVNEG |
| VNMLA | 浮動小数点積和 | 164 ページのVNMLA、VNMLS、VNMUL |
| VNMLS | 浮動小数点積差 | 164 ページのVNMLA、VNMLS、VNMUL |
| VNMUL | 浮動小数点乗算 | 164 ページのVNMLA、VNMLS、VNMUL |
| VPOP | 拡張レジスタをポップ | 165 ページのVPOP |
| VPUSH | 拡張レジスタをプッシュ | 166 ページのVPUSH |
| VSQRT | 浮動小数点平方根 | 166 ページのVSQRT |
| VSTM | 拡張レジスタの多重ストア | 167 ページのVSTM |
| VSTR | 拡張レジスタをメモリにストア | 167 ページのVSTR |
| VSUB | 浮動小数点減算 | 168 ページのVSUB |
| VSEL | 条件付き VMOV ペアの代わりに、レジスタを選択します。 | 169 ページのVSEL |
| VMAXNM、VMINNM | IEEE754-2008 NaN 処理での最大値、最小値 | 169 ページのVMAXNM、VMINNM |
| VCVTA、VCVTN、VCVTP、VCVTM | 方向丸めを使用した浮動小数点数から整数への変換 | 170 ページのVCVTA、VCVTN、VCVTP、VCVTM |
| VRINTR、VRINTX | 浮動小数点から整数（浮動小数点フォーマット）への変換 | 170 ページのVRINTR、VRINTX |
| VRINTA、VRINTN、VRINTP、VRINTM | 方向丸めを使用した浮動小数点から整数（浮動小数点フォーマット）への変換 | 171 ページのVRINTA、VRINTN、VRINTP、VRINTM、VRINTZ |

### 3.11.1 VABS

浮動小数点の絶対値。

#### 構文

VABS{*cond*}.F<32|64> *<Sd|Dd>*, *<Sm|Dm>*

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>*、*<Sm|Dm>*
これらはデスティネーションの浮動小数点値、およびオペランドの浮動小数点値です。

#### 動作

この命令の場合：

1. オペランド浮動小数点レジスタの絶対値を取得します。
2. この結果をデスティネーションの浮動小数点レジスタに書き込みます。

#### 制限事項

制限事項はありません。

#### 条件フラグ

この命令によるフラグの変更はありません。

#### 例

```
VABS.F32 S4, S6
```

### 3.11.2 VADD

浮動小数点加算。

#### 構文

VADD{*cond*}.F<32|64> {*<Sd|Dd>*,} *<Sn|Dn>*, *<Sm|Dm>*

VADD{*cond*}.F64 {*Dd*,} *Dn*, *Dm*

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>* これはデスティネーションの浮動小数点値です。

*<Sn|Dn>*、*<Sm|Dm>*
これらはオペランドの浮動小数点値です。

#### 動作

この命令の場合：

1. 2 つの浮動小数点オペランド・レジスタの値を加算します。
2. この結果をデスティネーションの浮動小数点レジスタに書き込みます。

### 制限事項

制限事項はありません。

### 条件フラグ

この命令によるフラグの変更はありません。

### 例

```
VADD.F32 S4, S6, S7
```

## 3.11.3 VCMP、VCMPE

2 つの浮動小数点レジスタ、または 1 つの浮動小数点レジスタとゼロを比較。

### 構文

```
VCMP{E}{cond}.F<32|64> <Sd|Dd>, <Sm|Dm>
VCMP{E}{cond}.F<32|64> <Sd|Dd>, #0.0
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*E* これが指定されている場合は、任意の NaN オペランドによって無効演算例外が生成されます。それ以外の場合は、信号 NaN のみによって例外が生成されます。

*<Sd|Dd>* これは比較元の浮動小数点オペランドです。

*<Sm|Dm>* これは比較対象の浮動小数点オペランドです。

### 動作

この命令の場合：

1. 次のいずれかを比較します。
   - 2 つの浮動小数点レジスタ。
   - あるいは 1 つの浮動小数点レジスタとゼロ。
2. 結果を FPSCR フラグに書き込みます。

### 制限事項

いずれかのオペランドが何らかのタイプの NaN である場合は、この命令は無効演算例外を任意に生成する可能性があります。いずれかのオペランドが信号 NaN である場合は、常に無効演算例外を生成します。

### 条件フラグ

この命令が結果を FPSCR フラグに書き込むと、通常、後続の VMRS 命令によってその値が Arm フラグに転送されます（162 ページのVMRS を参照）。

### 例

```
VCMP.F32    S4, #0.0VCMP.F32    S4, S2
```

### 3.11.4 VCVT、VCVTR（浮動小数点数と整数間）

レジスタの値を浮動小数点値と 32 ビット整数値の間で相互に変換。

#### 構文

VCVT{*R*}{*cond*}.*Tm*.F<32|64> <*Sd*|*Dd*>, <*Sm*|*Dm*>

VCVT{*cond*}.F<32|64>.*Tm* <*Sd*|*Dd*>, <*Sm*|*Dm*>

ここで、

*R* *R* が指定されている場合、演算では、FPSCR によって指定されている丸めモードが使用されます。*R* が省略されている場合、演算では、「ゼロ方向への丸め」の丸めモードが使用されます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Tm* これはオペランドのデータ型です。以下のいずれかを指定する必要があります。

- S32 符号付き 32 ビット値。
- U32 符号なし 32 ビット値。

<*Sd*|*Dd*>、<*Sm*|*Dm*>

これらはデスティネーション·レジスタとオペランド·レジスタです。

#### 動作

これらの命令の場合：

1. 以下のいずれかを実行します。
   - レジスタの値を浮動小数点値から 32 ビット整数に変換します。
   - 32 ビット整数から浮動小数点値に変換します。
2. 結果を 2 番目のレジスタに書き込みます。

浮動小数点数から整数への演算では、通常、「ゼロ方向への丸め」の丸めモードを使用しますが、FPSCR で指定されている丸めモードを任意で使用することもできます。

整数から浮動小数点数への演算では、FPSCR で指定されている丸めモードを使用します。

#### 制限事項

制限事項はありません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.5 VCVT（浮動小数点数と固定小数点間）

レジスタの値を浮動小数点数と固定小数点数間で変換。

#### 構文

```
VCVT{cond}.Td.F<32|64> <Sd|Dd>, <Sd|Dd>, #fbits
VCVT{cond}.F<32|64>.Td <Sd|Dd>, <Sd|Dd>, #fbits
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Td* これは固定小数点数のデータ型です。以下のいずれかを指定する必要があります。

- S16 符号付き 16 ビット値。
- U16 符号なし 16 ビット値。
- S32 符号付き 32 ビット値。
- U32 符号なし 32 ビット値。

*<Sd|Dd>* これはデスティネーション·レジスタとオペランド·レジスタです。

*fbits* これは固定小数点数の小数部ビット数です。

- *Td* が S16 または U16 の場合は、*fbit* は 0 ~ 16 の範囲内とする必要があります。
- *Td* が S32 または U32 の場合は、*fbit* は 1 ~ 32 の範囲内とする必要があります。

#### 動作

この命令の場合：

1. 次のいずれかを実行します。
   - レジスタの値を浮動小数点から固定小数点に変換します。
   - レジスタの値を固定小数点から浮動小数点に変換します。
2. 結果を 2 番目のレジスタに書き込みます。

浮動小数点値は単精度または倍精度です。

固定小数点値は 16 ビットまたは 32 ビットにできます。固定小数点値からの変換では、ソース·レジスタの下位ビットからそのオペランドを取得し、残りのビットを無視します。

固定小数点値への符号付き変換では、結果の値をデスティネーション·レジスタ幅に符号拡張します。

固定小数点値への符号なし変換では、結果の値をデスティネーション·レジスタ幅にゼロ拡張します。

浮動小数点から固定小数点への変換では、「ゼロ方向への丸め」の丸めモードを使用します。固定小数点から浮動小数点への変換では、「最も近い値への丸め」の丸めモードを使用します。

#### 制限事項

制限事項はありません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.6 VCVTB、VCVTT

中間の丸めを行わずに、半精度と単精度または倍精度との間で変換を行います。

#### 構文

```
VCVT{y}{cond}.F<32|64>.F16 <Sd|Dd>, Sm
VCVT{y}{cond}.F16.F<32|64> Sd, <Sm|Dm>
```

ここで、

*y* これはオペランド・レジスタ *Sm* またはデスティネーション・レジスタ *Sd* の上位と下位のどちらのハーフワードをオペランドまたはデスティネーションに使用するのかを指定します。

- y が B の場合は、*Sm* または *Sd* の下位ハーフワード、ビット [15:0] が使用されます。
- *y* が T の場合は、*Sm* または *Sd* の上位ハーフワード、ビット [31:16] が使用されます。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>* これはデスティネーション・レジスタです。

*<Sm|Dm>* これはオペランド・レジスタです。

#### 動作

この命令に .F16.F<32|64> 接尾文字が指定されている場合：

1. 単精度レジスタの上位または下位半分の半精度値を単精度値または倍精度値に変換します。
2. 結果を単精度または倍精度レジスタに書き込みます。

この命令に .F<32|64>.F16 接尾文字が指定されている場合：

1. 倍精度または単精度レジスタの値を半精度値に変換します。
2. 結果を単精度レジスタの上位または下位半分に書き込み、ターゲット・レジスタの残り半分はそのまま保持します。

#### 制限事項

制限事項はありません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.7 VDIV

浮動小数点値を除算します。

#### 構文

```
VDIV{cond}.F<32|64> {<Sd|Dd>,} <Sn|Dn>, <Sm|Dm>
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>* これはデスティネーション・レジスタです。

*<Sn|Dn>*、*<Sm|Dm>* これらはオペランド・レジスタです。

### 動作

この命令の場合：

1. 浮動小数点値を別の浮動小数点値で除算します。
2. 結果を浮動小数点デスティネーション･レジスタに書き込みます。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.8 VFMA、VFMS

結合浮動小数点積和および積差。

### 構文

```
VFMA{cond}.F<32|64> {<Sd|Dd>,} <Sn|Dn>, <Sm|Dm>
VFMS{cond}.F<32|64> {<Sd|Dd>,} <Sn|Dn>, <Sm|Dm>
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>* これはデスティネーション･レジスタです。

*<Sn|Dn>*、*<Sm|Dm>*
これらはオペランド･レジスタです。

### 動作

VFMA 命令：

1. オペランド･レジスタの浮動小数点値を乗算します。
2. 結果をデスティネーション･レジスタに累積します。

累算の前に乗算の結果の丸めは行われません。

VFMS 命令：

1. 第 1 オペランド･レジスタを符号反転します。
2. 第 1 オペランド･レジスタと第 2 オペランド･レジスタの浮動小数点値を乗算します。
3. 乗算結果をデスティネーション･レジスタに加算します。
4. この結果をデスティネーション･レジスタに書き込みます。

加算の前に乗算の結果の丸めは行われません。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.9 VFNMA、VFNMS

符号反転を伴う浮動小数点結合積和および積差。

#### 構文

```
VFNMA{cond}.F<32|64> {<Sd|Dd>,} <Sn|Dn>, <Sm|Dm>
VFNMS{cond}.F<32|64> {<Sd|Dd>,} <Sn|Dn>, <Sm|Dm>
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>* これはデスティネーション・レジスタです。

*<Sn|Dn>*、*<Sm|Dm>* これらはオペランド・レジスタです。

#### 動作

VFNMA 命令：

1. 第 1 浮動小数点オペランド・レジスタを符号反転します。
2. 第 1 浮動小数点オペランドに第 2 浮動小数点オペランドを乗算します。
3. 乗算結果に浮動小数点デスティネーション・レジスタの符号反転値を加算します。
4. この結果をデスティネーション・レジスタに書き込みます。

加算の前に乗算の結果の丸めは行われません。

VFNMS 命令：

1. 第 1 浮動小数点オペランドに第 2 浮動小数点オペランドを乗算します。
2. 乗算結果にデスティネーション・レジスタの浮動小数点値の符号反転値を加算します。
3. この結果をデスティネーション・レジスタに書き込みます。

加算の前に乗算の結果の丸めは行われません。

#### 制限事項

制限事項はありません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.10 VLDM

浮動小数点多重ロード。

#### 構文

```
VLDM{mode}{cond}{.size} Rn{!}, list
```

ここで、

*モード* これは次のいずれかのアドレッシング・モードです。

*IA* ロード後にインクリメントします。連続するアドレスは *Rn* で指定されているアドレスから開始されます。

*DB* ロード前にデクリメントします。連続するアドレスは直前で終了します。*Rn* で指定されたアドレス。

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*size* これは任意のデータ･サイズ指定子です。
*Rn* これはベース･レジスタです。SP を使用できます。
*!* これは変更した値を *Rn* にライトバックする命令を実行するためのコマンドです。mode == DB の場合はこれを指定する必要があります。mode == IA の場合は任意です。
*list* これはロードする拡張レジスタのリストです。連続する番号のダブルワードまたはシングルワードのレジスタをコンマで区切り、中かっこで囲んだリストとして指定します。

### 動作

この命令は、Arm コア･レジスタのアドレスをベース･アドレスとして使用して、連続するメモリ位置から複数の拡張レジスタをロードします。

### 制限事項

次の制限事項があります。

- サイズが存在する場合、リストのレジスタのビットのサイズ（32 または 64）に等しい必要があります。
- ベース･アドレスの場合、SP を使用できます。Arm 命令セットでは、*!* が指定されなければ、PC を使用できます。
- リストは少なくとも 1 つのレジスタを含む必要があります。リストがダブルワードのレジスタを含む場合、含まれるレジスタ数は 16 以下である必要があります。
- DB アドレッシング･モードを使用する場合、ライトバック･フラグ *!* をベース･レジスタ指定に追加する必要があります。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 例

```
VLDMIA.F64 r1, {d3,d4,d5}
```

## 3.11.11 VLDR

メモリから単一の拡張レジスタをロードします。

### 構文

```
VLDR{cond}{.F<32|64>} <sd|Dd>, [Rn{#imm}]

{cond}{.F<32|64> }<Sd|Dd>, <Sm|Dm>

VLDR{cond}{.F<32|64>} <sd|Dd>, [PC, #imm}]
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*32*、*64* これらは任意のデータ･サイズ指定子です。
*Dd* これはダブルワード･ロードのデスティネーション･レジスタです。
*Sd* これはシングルワード･ロードのデスティネーション･レジスタです。
*Rn* これはベース･レジスタです。SP を使用できます。
*imm* これはアドレス形成に使用する + または - のイミディエート･オフセットです。許容されるアドレス値は、0 ~ 1020 の範囲の 4 の倍数です。
*label* これはロードするリテラル･データ項目のラベルです。

### 動作

この命令は、Arm コア·レジスタからのベース·アドレスおよび任意のオフセットを使用して、単一の拡張レジスタをメモリからロードします。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.12 VMLA、VMLS

2 つの浮動小数点値を乗算して、結果を累算または減算します。

### 構文

`VMLA{cond}.F<32|64> <Sd|Dd>, <Sn|Dn>, <Sm|Dm>`

`VMLS{cond}.F<32|64> <Sd|Dd>, <Sn|Dn>, <Sm|Dm>`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>* これはデスティネーションの浮動小数点値です。

*<Sn|Dn>*、*<Sm|Dm>*
これらはオペランドの浮動小数点値です。

### 動作

浮動小数点積和命令：

1. 2 つの浮動小数点値を乗算します。
2. 結果をデスティネーションの浮動小数点値に加算します。

浮動小数点積差命令：

1. 2 つの浮動小数点値を乗算します。
2. 結果をデスティネーションの浮動小数点値から減算します。
3. この結果をデスティネーション·レジスタに書き込みます。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.13 VMOV （イミディエート）

浮動小数点のイミディエートを転送します。

#### 構文

```
VMOV{cond}.F<32|64> <Sd|Dd>, #imm
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*<Sd|Dd>* これはデスティネーション·レジスタです。
*imm* これは浮動小数点の定数です。

#### 動作

この命令は、定数値を浮動小数点レジスタにコピーします。

#### 制限事項

制限事項はありません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.14 VMOV レジスタ

レジスタの内容を別のレジスタにコピーします。

#### 構文

```
VMOV{cond}.F<32|64> <Sd|Dd>, <Sm|Dm>
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*Dd* これはダブルワード演算のデスティネーション·レジスタです。
*Dm* これはダブルワード演算のソース·レジスタです。
*Sd* これはシングルワード演算のデスティネーション·レジスタです。
*Sm* これはシングルワード演算のソース·レジスタです。

#### 動作

この命令は、浮動小数点レジスタの内容を別の浮動小数点レジスタにコピーします。

#### 制限事項

制限事項はありません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.15 VMOV（スカラから Arm コア･レジスタへ）

ダブルワード浮動小数点レジスタの 1 ワードを Arm コア･レジスタに転送します。

#### 構文

```
VMOV{cond} Rt, Dn[x]
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*Rt* これはデスティネーション Arm コア･レジスタです。
*Dn* これは 64 ビット･ダブルワード･レジスタです。
*x* ダブルワード･レジスタのどちらの半分を使用するかを指定します。

- *x* 0 であれば、ダブルワード・レジスタの下位半分を使用します。
- *x* 1 の場合は、ダブルワード・レジスタの上位半分を使用します。

#### 動作

この命令は、ダブルワード浮動小数点レジスタの上位半分または下位半分の 1 ワードを Arm コア･レジスタに転送します。

#### 制限事項

*Rt* は PC または SP のいずれにもできません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.16 VMOV（Arm コア･レジスタと単精度レジスタの間）

単精度レジスタと Arm コア･レジスタの間で転送します。

#### 構文

```
VMOV{cond} Sn, Rt
```

```
VMOV{cond} Rt, Sn
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*<Sn>* これは単精度浮動小数点レジスタです。
*Rt* これは Arm コア･レジスタです。

#### 動作

この命令は次の転送を行います。

- 単精度レジスタの内容を Arm コア･レジスタに転送します。
- Arm コア･レジスタの内容を単精度レジスタに転送します。

#### 制限事項

*Rt* は PC または SP のいずれにもできません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.17 VMOV（2 つの Arm コア･レジスタと 2 つの単精度レジスタの間）

2 つの連続する番号の単精度レジスタと 2 つの Arm コア･レジスタの間で転送します。

#### 構文

```
VMOV{cond} Sm, Sm1, Rt, Rt2
VMOV{cond} Rt, Rt2, Sm, Sm1
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*Sm* これは 1 番目の単精度レジスタです。
*Sm1* これは 2 番目の単精度レジスタです。これは *<Sm>* の次の単精度レジスタです。
*Rt* これは *<Sm>* との間で転送を行う Arm コア･レジスタです。
*Rt2* これは *<Sm1>* との間で転送を行う Arm コア･レジスタです。

#### 動作

この命令は次の転送を行います。

- 2 つの連続する番号の単精度レジスタの内容を 2 つの Arm コア･レジスタに転送します。
- 2 つの Arm コア･レジスタの内容を 1 組の単精度レジスタに転送します。

#### 制限事項

次の制限事項があります。

- 2 つの浮動小数点レジスタは連続している必要があります。
- Arm コア･レジスタは連続している必要はありません。
- *Rt* は PC または SP のいずれにもできません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.18 VMOV（2 つの Arm コア・レジスタと 1 つの倍精度レジスタ）

2 つの Arm コア・レジスタから 1 つのダブルワード・レジスタに、または 1 つのダブルワード・レジスタから 2 つの Arm コア・レジスタに 2 ワードを転送します。

#### 構文

```
VMOV{cond} Dm, Rt, Rt2
VMOV{cond} Rt, Rt2, Dm
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*Dm* これは倍精度レジスタです。
*Rt*、*Rt2* これらは2 つの Arm コア・レジスタです。

#### 動作

この命令の場合：

- 2 つの Arm コア・レジスタからの 2 ワードをダブルワード・レジスタに転送します。
- 1 つのダブルワード・レジスタを 2 つの Arm コア・レジスタに転送します。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.19 VMOV（Arm コア・レジスタからスカラへ）

Arm コア・レジスタの 1 ワードを浮動小数点レジスタに転送します。

### 構文

```
VMOV{cond}{.32} Dd[x], Rt
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*32* これは任意のデータ・サイズ指定子です。

*Dd[x]* これはデスティネーションです。ここで、[x] はダブルワードのどちらの半分を転送するかを次のように定義します。

- *x* が 0 の場合は下位半分を抽出します。
- *x* が 1 の場合は上位半分を抽出します。

*Rt* これはソース Arm コア・レジスタです。

### 動作

この命令は、Arm コア・レジスタの 1 ワードをダブルワード浮動小数点レジスタの上位半分または下位半分に転送します。

### 制限事項

*Rt* は PC または SP のいずれにもできません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.20 VMRS

浮動小数点システム・レジスタから Arm コア・レジスタに転送。

### 構文

```
VMRS{cond} Rt, FPSCR

VMRS{cond} APSR_nzcv, FPSCR
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rt* これはデスティネーション Arm コア・レジスタです。このレジスタは、R0 ~ R14 のいずれかです。

*APSR_nzcv* これは浮動小数点フラグを APSR フラグに転送します。

### 動作

この命令は、次のいずれかのアクションを実行します。

- FPSCR の値を汎用レジスタにコピーします。
- FPSCR フラグ･ビットの値を APSR の N、Z、C、および V の各フラグにコピーします。

### 制限事項

*Rt* は PC または SP のいずれにもできません。

### 条件フラグ

これらの命令により必要に応じて N、Z、C、および V の各フラグが変更されます。

## 3.11.21 VMSR

Arm コア･レジスタから 浮動小数点システム･レジスタに転送。

### 構文

```
VMSR{cond} FPSCR, Rt
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rt* これは FPSCR に転送される汎用レジスタです。

### 動作

この命令は、汎用レジスタの値を FPSCR に転送します。詳細については、232 ページの浮動小数点ステータス制御レジスタを参照してください。

### 制限事項

*Rt* は PC または SP のいずれにもできません。

### 条件フラグ

この命令は、FPSCR を更新します。

## 3.11.22 VMUL

浮動小数点乗算。

### 構文

```
VMUL{cond}.F<32|64> {<Sd|Dd>,} <Sn|Dn>, <Sm|Dm>
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>* これはデスティネーションの浮動小数点値です。

*<Sn|Dn>*、*<Sm|Dm>*
これらはオペランドの浮動小数点値です。

### 動作

この命令の場合：

1. 2 つの浮動小数点値を乗算します。
2. この結果をデスティネーション・レジスタに書き込みます。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.23 VNEG

浮動小数点符号反転。

### 構文

`VNEG{cond}.F<32|64> <Sd|Dd>, <Sm|Dm>`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*<Sd|Dd>* これはデスティネーションの浮動小数点値です。
*<Sm|Dm>* これはオペランドの浮動小数点値です。

### 動作

この命令の場合：

1. 浮動小数点値を符号反転します。
2. この結果を 2 番目の浮動小数点レジスタに書き込みます。

この浮動小数点命令で符号ビットを反転します。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.24 VNMLA、VNMLS、VNMUL

浮動小数点を乗算し、その結果を符合反転した後、加算または減算を行います。

### 構文

`VNMLA{cond}.F<32|64> <Sd|Dd>, <Sn|Dn>, <Sm|Dm>`

`VNMLS{cond}.F<32|64> <Sd|Dd>, <Sn|Dn>, <Sm|Dm>`

`VNMUL{cond}.F<32|64> {<Sd|Dd>,} <Sn|Dn>, <Sm|Dm>`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。
*<Sd|Dd>* これはデスティネーションの浮動小数点レジスタです。
*<Sn|Dn>*、*<Sm|Dm>* はオペランドの浮動小数点レジスタです。

### 動作

VNMLA 命令：

1. 2 つの浮動小数点レジスタの値を乗算します。
2. 乗算結果の符合反転値にデスティネーション·レジスタの浮動小数点値の符号反転値を加算します。
3. 結果をデスティネーション·レジスタにライトバックします。

VNMLS 命令：

1. 2 つの浮動小数点レジスタの値を乗算します。
2. 乗算結果にデスティネーション·レジスタの浮動小数点値の符号反転値を加算します。
3. 結果をデスティネーション·レジスタにライトバックします。

VNMUL 命令：

1. 2 つの浮動小数点レジスタの値を乗算します。
2. 結果の符号反転値をデスティネーション·レジスタに書き込みます。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.25 VPOP

浮動小数点拡張レジスタのポップ。

### 構文

```
VPOP{cond}{.size} list
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*size* これは任意のデータ·サイズ指定子です。サイズが存在する場合、リストのレジスタのビットのサイズ（32 または 64）に等しい必要があります。

*list* これはロードする拡張レジスタのリストです。連続する番号のダブルワードまたはシングルワードのレジスタをコンマで区切り、
中かっこで囲んだリストとして指定します。

### 動作

この命令は、スタックから複数の連続する拡張レジスタをロードします。

### 制限事項

リストは 1 ~ 16 個のレジスタを含む必要があります。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.26 VPUSH

浮動小数点拡張レジスタのプッシュ。

#### 構文

```
VPUSH{cond}{.size} list
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*size* これは任意のデータ·サイズ指定子です。サイズが存在する場合、リストのレジスタのビットのサイズ（32 または 64）に等しい必要があります。

*list* これはストアする拡張レジスタのリストです。連続する番号のダブルワードまたはシングルワードのレジスタをコンマで区切り、中かっこで囲んだリストとして指定します。

#### 動作

この命令は、複数の連続する拡張レジスタをスタックにストアします。

#### 制限事項

*リスト*は 1 ~ 16 個のレジスタを含む必要があります。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.27 VSQRT

浮動小数点平方根。

#### 構文

```
VSQRT{cond}.F<32|64> <Sd|Dd>, <Sm|Dm>
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>* これはデスティネーションの浮動小数点値です。

*<Sm|Dm>* これはオペランドの浮動小数点値です。

#### 動作

この命令の場合：

- 浮動小数点レジスタの値の平方根を計算します。
- 結果を別の浮動小数点レジスタに書き込みます。

#### 制限事項

制限事項はありません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.28 VSTM

浮動小数点多重ストア。

### 構文

```
VSTM{mode}{cond}{.size} Rn{!}, list
```

ここで、

モード　これは次のいずれかのアドレッシング·モードです。
- IA：ストア後にインクリメントします。連続するアドレスは Rn で指定されているアドレスから開始されます。これはデフォルトであり、省略できます。
- DB：ストア前にデクリメントします。連続するアドレスは Rn で指定されているアドレスの直前で終了します。

*cond*　これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*size*　これは任意のデータ·サイズ指定子です。サイズが存在する場合、リストのレジスタのビットのサイズ（32 または 64）に等しい必要があります。

*Rn*　これはベース·レジスタです。SP を使用できます。

*!*　これは変更した値を *Rn* にライトバックする命令を実行するためのコマンドです。mode == DB の場合は必須です。

*list*　これはストアする拡張レジスタのリストです。連続する番号のダブルワードまたはシングルワードのレジスタをコンマで区切り、中かっこで囲んだリストとして指定します。

#### 動作

この命令は、複数の拡張レジスタを、Arm コア·レジスタのベース·アドレスを使用して、連続するメモリ位置にストアします。

### 制限事項

次の制限事項があります。

- リストは少なくとも 1 つのレジスタを含む必要があります。リストがダブルワード·レジスタを含む場合、含まれるレジスタ数は 16 以下である必要があります。
- *Rn* として PC を使用することは非推奨となっています。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.29 VSTR

浮動小数点ストア。

### 構文

```
VSTR{cond}{.32} Sd, [Rn{, #imm}]
VSTR{cond}{.64} Dd, [Rn{, #imm}]
```

ここで、

*cond*　これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*32*、*64*　これらは任意のデータ·サイズ指定子です。

*Sd*　これはシングルワード·ストアのソース·レジスタです。

*Dd*　これはダブルワード·ストアのソース·レジスタです。

*Rn*　これはベース·レジスタです。SP を使用できます。

*imm* これはアドレス形成に使用する + または - のイミディエート・オフセットです。値は、0 ~ 1020 の範囲の 4 の倍数です。*imm* は省略可能で、+0 のオフセットを意味します。

### 動作

この命令は、Arm コア・レジスタからのアドレスおよび *imm* で定義される任意のオフセットを使用して、単一の拡張レジスタをメモリにストアします。

### 制限事項

*Rn* として PC を使用することは非推奨となっています。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.30 VSUB

浮動小数点減算。

### 構文

VSUB{*cond*}.F<32|64> {*<Sd|Dd>*,} *<Sn|Dn>*, *<Sm|Dm>*

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>* これはデスティネーションの浮動小数点値です。

*<Sn|Dn>*、*<Sm|Dm>*
これらはオペランドの浮動小数点値です。

### 動作

この命令の場合：

1. 浮動小数点値を別の浮動小数点値から減算します。
2. この結果をデスティネーション浮動小数点レジスタに書き込みます。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.31 VSEL

条件付き VMOV 命令のペアの代わりを提供します。

#### エンコード

```
VSEL{cond}.F<32|64> <Sd|Dd>, <Sn|Dn>, <Sm|Dm>
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。VSEL に条件コードのサブセットがあります。VSEL の条件コードは、LT、LE、NE および VC が、ソース·オペランドを交換することによって達成できるように、GE、GT、EQ および VS に限られます。

*<Sd|Dd>* これはデスティネーションの単精度または倍精度の浮動小数点値です。

*<Sn|Dn>*、*<Sm|Dm>*
これらは、単精度または倍精度の浮動小数点値のオペランドです。

#### 動作

条件コードの結果に応じて、この命令は次のいずれかの転送を実行します。

- *<Sn|Dn>* ソース・レジスタからデスティネーション・レジスタに。
- *<Sm|Dm>* ソース・レジスタからデスティネーション・レジスタに。

#### 制限事項

VSEL 命令は IT ブロック内に発生してはなりません。

#### 条件フラグ

これらの命令によるフラグの変更はありません。

### 3.11.32 VMAXNM、VMINNM

IEEE754-2008 で規定されている NaN 処理を行う 2 つの浮動小数点数の最小または最大を返します。

#### エンコード

```
VMAXNM.F<32|64> <Sd|Dd>, <Sn|Dn>, <Sm|Dm>

VMINNM.F<32|64> <Sd|Dd>, <Sn|Dn>, <Sm|Dm>
```

ここで、

*<Sd|Dd>* これはデスティネーションの単精度または倍精度の浮動小数点値です。

*<Sn|Dn>*、*<Sm|Dm>*
これらは、単精度または倍精度の浮動小数点値のオペランドです。

#### 動作

VMAXNM 命令は、2 つのソース・レジスタを比較し、最大値をデスティネーション・レジスタに転送します。

VMINNM 命令は、2 つのソース・レジスタを比較し、最小値をデスティネーション・レジスタに転送します。

#### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.33 VCVTA、VCVTN、VCVTP、VCVTM

方向丸めを使用した浮動小数点から整数への変換。

### 構文

```
VCVT<rmode>.S32.F<32|64> <Sd>, <Sm|Dm>
VCVT<rmode>.U32.F<32|64> <Sd>, <Sm|Dm>
```

ここで、

*<Sd|Dd>* これはデスティネーションの単精度または倍精度の浮動小数点値です。

*<Sn|Dn>*、*<Sm|Dm>*
これらは、単精度または倍精度の浮動小数点値のオペランドです。

*<rmode>* これは次のいずれかです。

- A 最近傍点への丸め。
- M 一番近い偶数への丸め。
- N プラスの無限大に向かっての丸め。
- P 負の無限大に向かっての丸め。

### 動作

これらの命令の場合：

1. ソース・レジスタを読み出します。
2. 方向丸めを使用して整数に変換します。
3. デスティネーション・レジスタに書き込みます。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.34 VRINTR、VRINTX

浮動小数点値を浮動小数点フォーマットの整数に丸めます。

### エンコード

```
VRINT{R,X}{cond}.F<32|64> <Sd|Dd>, <Sm|Dm>
```

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*<Sd|Dd>* これはデスティネーションの浮動小数点値です。

*<Sm|Dm>* これらはオペランドの浮動小数点値です。

### 動作

これらの命令の場合：

1. ソース・レジスタを読み出します。
2. FPSCR で指定された丸めモードを使って、最近接の浮動小数点フォーマットの整数値へ丸めます。
3. 結果をデスティネーション・レジスタに書き込みます。
4. VRINTZX 命令のみ。結果が正確でない場合に浮動小数点例外を生成します。

### 制限事項

制限事項はありません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.11.35 VRINTA、VRINTN、VRINTP、VRINTM、VRINTZ

方向丸めを使用して、浮動小数点値を浮動小数点フォーマットの整数に丸めます。

### エンコード

`VRINT<rmode>.F<32|64> <Sd|Dd>, <Sm|Dm>`

ここで、

*<Sd|Dd>* これはデスティネーションの単精度または倍精度の浮動小数点値です。

*<Sn|Dn>*、*<Sm|Dm>*
これらは、単精度または倍精度の浮動小数点値のオペランドです。

*<rmode>* これは次のいずれかです。

- `A` 最近傍点への丸め。
- `M` 一番近い偶数への丸め。
- `N` プラスの無限大に向かっての丸め。
- `P` 負の無限大に向かっての丸め。
- `Z` ゼロに向かっての丸め。

### 動作

これらの命令の場合：

1. ソース・レジスタを読み出します。
2. 命令で指定された方向丸めモードで、最も近い整数値に丸めます。
3. 結果をデスティネーション・レジスタに書き込みます。

### 制限事項

これらの命令を条件付きにはできません。これらの命令は、結果が正確ではない場合でも、不正確例外を生成することはできません。

### 条件フラグ

これらの命令によるフラグの変更はありません。

## 3.12 その他の命令

表 38 に、Cortex®-M7 のその他の命令を示します。

**表 38. その他の命令**

| ニーモニック | 概要 | 参照先 |
|---|---|---|
| BKPT | ブレークポイント | 172 ページのBKPT |
| CPSID | プロセッサ状態の変更、割込みを無効にする | 173 ページのCPS |
| CPSIE | プロセッサ状態の変更、割込みを有効にする | 173 ページのCPS |
| DMB | データ·メモリ·バリア | 174 ページのDMB |
| DSB | データ同期バリア | 174 ページのDSB |
| ISB | 命令同期バリア | 175 ページのISB |
| MRS | 特殊レジスタからレジスタへの転送 | 175 ページのMRS |
| MSR | レジスタから特殊レジスタへの転送 | 176 ページのMSR |
| NOP | 何もしない | 177 ページのNOP |
| SEV | イベント送信 | 177 ページのSEV |
| SVC | スーパバイザ·コール | 178 ページのSVC |
| WFE | イベント待機 | 178 ページのWFE |
| WFI | 割込み待機 | 179 ページのWFI |

### 3.12.1 BKPT

ブレークポイント。

#### 構文

`BKPT #imm`

ここで、

*imm* これは 0 ~ 255 （8 ビット値）の範囲の整数に評価される式です。

#### 動作

BKPT 命令は、プロセッサをデバッグ状態に移行します。デバッグ·ツールはこの命令を使用して、特定のアドレスの命令に到達したときのシステム状態を調査できます。

*imm* はプロセッサによって無視されます。必要に応じて、デバッガはこの値を使用して、ブレークポイントに関する追加情報をストアできます。

BKPT 命令は IT ブロック内に配置できますが、IT 命令によって指定される条件の影響を受けずに、無条件に実行されます。

#### 条件フラグ

この命令によるフラグの変更はありません。

### 例

```
BKPT #0x3    ; 0x3 にイミディエート値をセットするブレークポイント
             ; （デバッガは PC を使用してローケートすることで、
             ; イミディエート値を抽出できます）
```

Arm 社は、セミホスティング以外の目的での、イミディエート値を 0xAB に設定した BKPT 命令の使用を推奨しません。

## 3.12.2 CPS

プロセッサ状態を変更します。

### 構文

`CPS`*`effect iflags`*

ここで、

*effect* これは次のいずれかです。
- `IE` 特殊な目的のレジスタをクリアします。
- `ID:` 特殊な目的のレジスタをセットします。

*iflags* これは次の 1 つまたは複数のフラグの並びです。
- `i` PRIMASKをセットまたはクリアします。
- `f` FAULTMASKをセットまたはクリアします。

### 動作

CPS は PRIMASK と FAULTMASK の特殊レジスタの値を変更します。これらのレジスタの詳細については、25 ページの例外マスク・レジスタを参照してください。

### 制限事項

次の制限事項があります。

- 特権ソフトウェアからのみ CPS を使用。非特権ソフトウェアで使用しても効果はありません。
- CPS は条件付きにできないので、IT ブロック内で使用しないでください。

### 条件フラグ

この命令による条件フラグの変更はありません。

### 例

```
CPSID i  ; 割込みと設定可能なフォールト・ハンドラを無効化（PRIMASK をセット）
CPSID f  ; 割込みとすべてのフォールト・ハンドラを無効化（FAULTMASK をセット）
CPSIE i  ; 割込みと設定可能なフォールト・ハンドラを有効化（PRIMASK をクリア）
CPSIE f  ; 割込みとフォールト・ハンドラを有効化（FAULTMASK をクリア）
```

### 3.12.3 DMB

データ・メモリ・バリア

#### 構文

DMB{*cond*}

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

#### 動作

DMB はデータ・メモリ・バリアとして機能します。これにより、プログラム順で DMB 命令より前に出現するすべての明示的なメモリ・アクセスは、プログラム順で DMB 命令より後に出現するすべての明示的なメモリ・アクセスより前に完了することが確証されます。DMB は、メモリにアクセスしない命令の順序または実行には影響しません。

#### 条件フラグ

この命令によるフラグの変更はありません。

#### 例

```
DMB  ; データ・メモリ・バリア
```

### 3.12.4 DSB

データ同期バリア

#### 構文

DSB{*cond*}

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

#### 動作

DSB は特殊なデータ同期メモリ・バリアとして機能します。プログラム順で DSB より後に出現する命令は、DSB 命令が完了するまで実行されません。DSB 命令は、それより前のすべての明示的なメモリ・アクセスが完了すると、完了します。

#### 条件フラグ

この命令によるフラグの変更はありません。

#### 例

```
DSB ; データ同期バリア
```

### 3.12.5 ISB

命令同期バリア

#### 構文

`ISB{cond}`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

#### 動作

ISB は命令同期バリアとして機能します。ISB はプロセッサのパイプラインを一掃して、ISB 命令が完了した後に、ISB の後に続くすべての命令がキャッシュまたはメモリから再度フェッチされるようにします。

#### 条件フラグ

この命令によるフラグの変更はありません。

#### 例

```
ISB  ; 命令同期バリア
```

### 3.12.6 MRS

特殊レジスタの内容を汎用レジスタに転送します。

#### 構文

`MRS{cond} Rd, spec_reg`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rd* これはデスティネーション·レジスタです。

*spec_reg* これは次のいずれかになります。APSR、IPSR、EPSR、IEPSR、IAPSR、EAPSR、PSR、MSP、PSP、PRIMASK、BASEPRI、BASEPRI_MAX、FAULTMASK、または CONTROL.

すべての EPSR および IPSR フィールドは MRS 命令によって読み出されたときにはゼロです。

#### 動作

MRS は MSR と組み合わせて、PSR を更新（例：Q フラグのクリア）する読出し、修正、書込みのシーケンスの一部として使用します。

プロセス·スワップ·コードでは、関連する PSR の内容を含め、スワップ·アウトされているプロセスのプログラマのモデルの状態を保存する必要があります。同様に、スワップ·インされているプロセスの状態も復元する必要があります。これらの操作には、状態保存命令シーケンスの MRS および状態復元命令シーケンスの MSR を使用します。

BASEPRI_MAX は、MRS 命令と一緒に使用するときは、BASEPRI のエイリアスです。

176 ページのMSR を参照してください。

### 制限事項

*Rd* に SP または PC は使用できません。

### 条件フラグ

この命令によるフラグの変更はありません。

### 例

```
MRS  R0, PRIMASK ; PRIMASK 値を読み出し、R0 に書き込む
```

## 3.12.7 MSR

汎用レジスタの内容を指定した特殊レジスタに転送します。

### 構文

`MSR{cond} spec_reg, Rn`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*Rn* これはソース・レジスタです。

*spec_reg* これは次のいずれかになります。APSR_nzcvq、APSR_g、APSR_nzcvqg、MSP、PSP、PRIMASK、BASEPRI、BASEPRI_MAX、FAULTMASK、または CONTROL.

APSR_nzcvq を参照するために APSR を使用できます。.

### 動作

MSR のレジスタ・アクセス動作は、特権レベルに依存します。非特権ソフトウェアは APSR にのみアクセスできます（23 ページの表 4 を参照）。特権ソフトウェアはすべての特殊レジスタにアクセスできます。

非特権ソフトウェアでは、PSR の未割当てビットまたは実行状態ビットへの書込みは無視されます。

BASEPRI_MAX に書き込む場合、次のどちらかの場合のみ BASEPRI に書き込みます。

- *Rn* が 0 ではなく、現在の BASEPRI 値が 0 の場合。
- *Rn* が 0 ではなく、現在の BASEPRI 値より小さい場合。

175 ページのMRS を参照してください。

### 制限事項

*Rn* に SP または PC は使用できません。

### 条件フラグ

この命令は、*Rn* の値に基づいて明示的にフラグを更新します。

### 例

```
MSR  CONTROL, R1 ; R1 の値を読み出して CONTROL レジスタに書き込む
```

### 3.12.8 NOP

何もしない。

#### 構文

`NOP{cond}`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

#### 動作

NOP は何も行いません。NOP は必ずしも時間を消費する NOP ではありません。プロセッサにより、この命令は、実行ステージに到達する前にパイプラインから削除される場合があります。

たとえば、NOP をパディングに使用することで、次に続く命令を 64 ビット境界に配置できます。

#### 条件フラグ

この命令によるフラグの変更はありません。

#### 例

```
NOP  ; 何もしない
```

### 3.12.9 SEV

イベントを送信します。

#### 構文

`SEV{cond}`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

#### 動作

SEV は、マルチプロセッサ・システム内のすべてのプロセッサに対してイベントを発生させるヒント命令です。また、ローカル・イベント・レジスタを 1 にセットします（49 ページの電源管理を参照）。

#### 条件フラグ

この命令によるフラグの変更はありません。

#### 例

```
SEV ; イベント送信
```

### 3.12.10 SVC

スーパバイザ·コール。

#### 構文

`SVC{cond} #imm`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

*imm* これは 0 ~ 255 （8 ビット値）の範囲の整数に評価される式です。

#### 動作

SVC 命令は、SVC 例外を発生させます。

*imm* はプロセッサによって無視されます。必要な場合、例外ハンドラでこの値を取得して、要求されているサービスを特定できます。

#### 条件フラグ

この命令によるフラグの変更はありません。

#### 例

```
SVC  #0x32 ;スーパーバイザ・コール（SVCall ハンドラはスタックされた
           ; PC によりローケートすることで、イミディエート値を抽出可能）
```

### 3.12.11 WFE

イベント待機。

#### 構文

`WFE{cond}`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

#### 動作

WFE はヒント命令です。

イベント·レジスタが 0 の場合、WFE は次のいずれかのイベントが発生するまで実行を中断します。

- 例外（例外マスク·レジスタでマスクされている場合または現在の優先度レベルの場合を除く）。
- 例外が保留状態に入る（システム制御レジスタで SEVONPEND がセットされている場合）。
- デバッグ·エントリ要求（デバッグが有効な場合）。
- ペリフェラルまたはマルチプロセッサ·システムの別のプロセッサが SEV 命令によって発生させたイベント

イベント·レジスタが 1 の場合、WFE はそれを 0 にクリアして、すぐに戻ります。

詳細については、49 ページの電源管理を参照してください。

#### 条件フラグ

この命令によるフラグの変更はありません。

### 例

```
WFE  ; イベント待機
```

## 3.12.12 WFI

割込み待機。

### 構文

`WFI{cond}`

ここで、

*cond* これはオプションの条件コードです。68 ページの条件付き実行を参照してください。

### 動作

WFI は、次のいずれかのイベントが発生するまで実行を中断するヒント命令です。

- マスクされていない割込みが発生して実行される。
- PRIMASK によってマスクされた割込みが保留状態。
- デバッグ・エントリ・リクエスト。

### 条件フラグ

この命令によるフラグの変更はありません。

### 例

```
WFI ; 割込み待機
```

# 4 Cortex-M7 ペリフェラル

## 4.1 Cortex-M7 ペリフェラルについて

プライベート･ペリフェラル･バス（PPB）のアドレス･マップを次に示します。

**表 39. コア･ペリフェラルのレジスタ領域**

| アドレス | コア･ペリフェラル | 説明 |
|---|---|---|
| 0xE000E008 ~ 0xE000E00F | システム制御ブロック | 189 ページの表 50 |
| 0xE000E010 ~ 0xE000E01F | システム･タイマ | 210 ページの表 71 |
| 0xE000E100 ~ 0xE000E4EF | ネスト化されたベクタ割込みコントローラ | 181 ページの表 40 |
| 0xE000ED00 ~ 0xE000ED3F | システム制御ブロック | 189 ページの表 50 |
| 0xE000ED78 ~ 0xE000ED84 | プロセッサの機能 | 214 ページの表 77 |
| 0xE000ED90 ~ 0xE000EDB8 | メモリ保護ユニット | 219 ページの表 84 |
| 0xE000EF00 ~ 0xE000EF03 | ネスト化されたベクタ割込みコントローラ | 181 ページの表 40 |
| 0xE000EF30 ~ 0xE000EF44 | 浮動小数点ユニット | 229 ページの表 94 |
| 0xE000EF50 ~ 0xE000EF78 | キャッシュのメンテナンス操作 | 236 ページの表 100 |
| 0xE000EF90 ~ 0xE000EFA8 | アクセス制御 | 241 ページの表 104 |

レジスタの説明内：

- レジスタ･*タイプ*は次のように記述されます。
  **RW** 読出しおよび書込み
  **RO** 読出し専用。
  **WO** 書込み専用。
- 必要な特権には、レジスタにアクセスするために必要な特権レベルが次のように示されます。
  **特権**
  特権ソフトウェアだけがレジスタにアクセスできます。
  **非特権**
  非特権ソフトウェアと特権ソフトウェアの両方がレジスタにアクセスできます。

非特権ソフトウェアから特権レジスタにアクセスしようとすると、バス・フォールトが発生します。

## 4.2 ネスト化されたベクタ割込みコントローラ

このセクションでは、NVIC、およびそれが使用するレジスタについて説明します。NVIC は、次をサポートします。

- 1 ~ 240 割込み
- 割込みごとにプログラム可能な優先度レベル（0 ~ 255）。レベルが大きいほど優先度が低いので、最も割込み優先度が高いのはレベル 0 です。
- レベルとパルスによる割込み信号の検出。
- 割込みの動的な再優先度付け。
- 優先度値をグループ優先度フィールドとサブ優先度フィールドにグループ化。
- 割込みのテールチェイン。
- 外部ノンマスカブル割込み（NMI）。

プロセッサは、例外エントリ時に自動的にその状態をスタックに格納し、例外からの復帰時にその状態をスタックから取り出します。命令のオーバーヘッドは発生しません。これにより、少ない遅延で例外を処理できます。NVIC レジスタのハードウェア実装を次に示します。

**表 40. NVIC レジスタの概要**

| アドレス | 名前 | タイプ | 必要特権 | リセット値 | 説明 |
|---|---|---|---|---|---|
| 0xE000E100 ~ 0xE000E11C | NVIC_ISER0 - NVIC_ISER7 | RW | 特権 | 0x00000000 | 182 ページの割込みセット・イネーブル・レジスタ |
| 0XE000E180 ~ 0xE000E19C | NVIC_ICER0 - NVIC_ICER7 | RW | 特権 | 0x00000000 | 183 ページの割込みクリア・イネーブル・レジスタ |
| 0XE000E200 ~ 0xE000E21C | NVIC_ISPR0 - NVIC_ISPR7 | RW | 特権 | 0x00000000 | 183 ページの割込みセット・ペンディング・レジスタ |
| 0XE000E280 ~ 0xE000E29C | NVIC_ICPR0 - NVIC_ICPR7 | RW | 特権 | 0x00000000 | 184 ページの割込みクリア・ペンディング・レジスタ |
| 0xE000E300 ~ 0xE000E31C | NVIC_IABR0 - NVIC_IABR7 | RW | 特権 | 0x00000000 | 185 ページの割込みアクティブ・ビット・レジスタ |
| 0xE000E400 ~ 0xE000E4EF | NVIC_IPR0 - NVIC_IPR59 | RW | 特権 | 0x00000000 | 185 ページの割込み優先度レジスタ |
| 0xE000EF00 | STIR | WO | 設定可能(1) | 0x00000000 | 186 ページのソフトウェア・トリガ割込みレジスタ |

1. 詳細については、レジスタの説明を参照してください。

### 4.2.1 CMSIS を使用した Cortex®-M7 NVIC レジスタへのアクセス

CMSIS 関数により、さまざまな Cortex®-M プロファイル・プロセッサ間でソフトウェアを移植できます。CMSIS の使用時に NVIC レジスタにアクセスするには、次の関数を使用します。

**表 41. CMSIS の NVIC アクセス関数**

| CMSIS 関数 | 説明 |
|---|---|
| `void NVIC_EnableIRQ(IRQn_Type IRQn)` (1) | 割込みまたは例外を有効にします。 |
| `void NVIC_DisableIRQ(IRQn_Type IRQn)` (1) | 割込みまたは例外を無効にします。 |
| `void NVIC_SetPendingIRQ(IRQn_Type IRQn)` (1) | 割込みまたは例外の保留ステータスを 1 にセットします。 |
| `void NVIC_ClearPendingIRQ(IRQn_Type IRQn)` (1) | 割込みまたは例外の保留ステータスを 0 にクリアします。 |
| `uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)` (1) | 割込みまたは例外の保留ステータスを読み出します。この関数は、保留ステータスが 1 にセットされている場合は 0 以外の値を返します。 |
| `void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)` (1) | 設定可能な優先度レベルを持つ割込みまたは例外の優先度を 1 にセットします。 |
| `uint32_t NVIC_GetPriority(IRQn_Type IRQn)` (1) | 設定可能な優先度レベルを持つ割込みまたは例外の優先度を読み出します。この関数は、現在の優先度レベルを返します。 |

1. 入力パラメータ IRQn は IRQ 番号です（詳細は40 ページの表 19 を参照）。

### 4.2.2 割込みセット・イネーブル・レジスタ

NVIC_ISER0 ~ NVIC_ISER7 の各レジスタは割込みを有効にして、どの割込みが有効なのかを示します。レジスタの属性については、181 ページの表 40 のレジスタの概要を参照してください。

ビット割当てを次に示します。

**図 17. ISER のビット割当て**

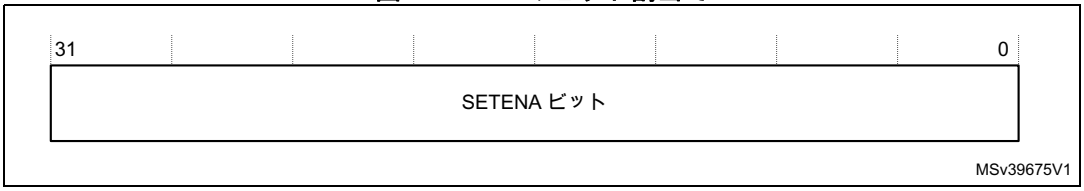



**表 42. ISER ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:0] | SETENA | 割込みセット・イネーブル・ビット。<br>書込み：<br>0：影響はありません。<br>1：割込みの有効化<br>読出し：<br>0：割込みは無効。<br>1：割込み有効。 |

保留中の割込みが有効になると、NVIC はその優先度に基づいて割込みをアクティブにします。割込みが有効ではない場合、その割込み信号をアサートすると、割込み状態が保留中に変化しますが、NVIC はその優先度に関係なく割込みをアクティブにしません。

### 4.2.3 割込みクリア・イネーブル・レジスタ

NVIC_ICER0 ~ NVIC_ICER7 の各レジスタは割込みを無効にして、どの割込みが有効なのかを示します。レジスタの属性については、181 ページの表 40 のレジスタの概要を参照してください。

ビット割当てを次に示します。

**図 18. ICER のビット割当て**

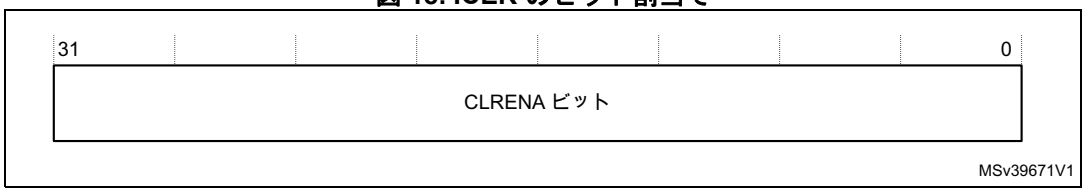



**表 43. ICER ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:0] | CLRENA | 割込みクリア・イネーブル・ビット。<br>書込み：<br>0：影響はありません。<br>1：割込みを無効化。<br>読出し：<br>0：割込みは無効。<br>1：割込み有効。 |

### 4.2.4 割込みセット・ペンディング・レジスタ

NVIC_ISPR0 ~ NVIC_ISPR7 の各レジスタは、割込みを強制的に保留状態にして、どの割込みが保留中なのかを示します。レジスタの属性については、181 ページの表 40 のレジスタの概要を参照してください。

ビット割当てを次に示します。

**図 19. ISPR のビット割当て**

表 44. ISPR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:0] | SETPEND | 割込みセット・ペンディング・ビット<br>書込み：<br>0：影響はありません。<br>1：割込み状態を保留中に変更します。<br>読出し：<br>0：割込みは保留中ではありません。<br>1：割込みは保留中です。 |

次の項目に対応した ISPR ビットへの 1 の書込み：

- 保留中の割込みは無効です。
- 無効な割り込みセットは、その割込みの状態を保留中に設定します。

### 4.2.5 割込みクリア・ペンディング・レジスタ

NVIC_ICPR0 ~ NCVIC_ICPR7 の各レジスタは、割込みの保留状態を解除して、どの割込みが保留中なのかを示します。レジスタの属性については、181 ページの表 40 のレジスタの概要を参照してください。

ビット割当てを次に示します。

図 20. ICPR のビット割当て


表 45. ICPR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:0] | CLRPEND | 割込みクリア・ペンディング・ビット<br>書込み：<br>0：影響はありません。<br>1：割込みの保留状態を解除します。<br>読出し：<br>0：割込みは保留中ではありません。<br>1：割込みは保留中です。 |

ICPR ビットへの 1 の書込みは、対応する割込みのアクティブ状態に影響を及ぼしません。

### 4.2.6 割込みアクティブ・ビット・レジスタ

NVIC_IABR0 ~ NVIC_IABR7 の各レジスタは、どの割込みがアクティブかを示します。レジスタの属性については、181 ページの表 40 のレジスタの概要を参照してください。

ビット割当てを次に示します。

**図 21. IABR のビット割当て**

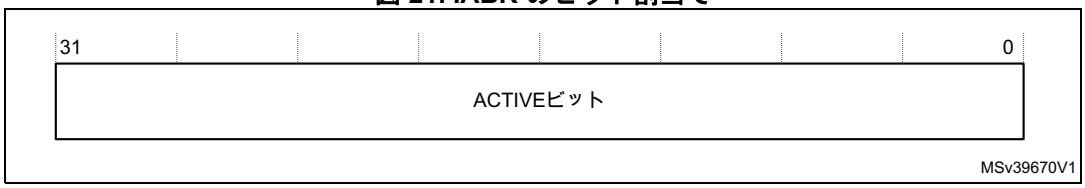



**表 46. IABR のビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:0] | ACTIVE | 割込みアクティブ・フラグ：<br>0：割込みはアクティブではありません。<br>1：割込みはアクティブです。 |

対応する割込みのステータスがアクティブまたはアクティブかつ保留中の場合、読み出すビットは 1 になります。

### 4.2.7 割込み優先度レジスタ

NVIC_IPR0 ~ NVIC_IPR59 の各レジスタは、各割込みに 8 ビットの優先度フィールドを提供します。これらのレジスタは、バイトアクセス可能です。属性については、181 ページの表 40 のレジスタ概要を参照してください。各レジスタは、次に示すように 4 つの優先度フィールドを保持します。

**図 22. IPR のビット割当て**

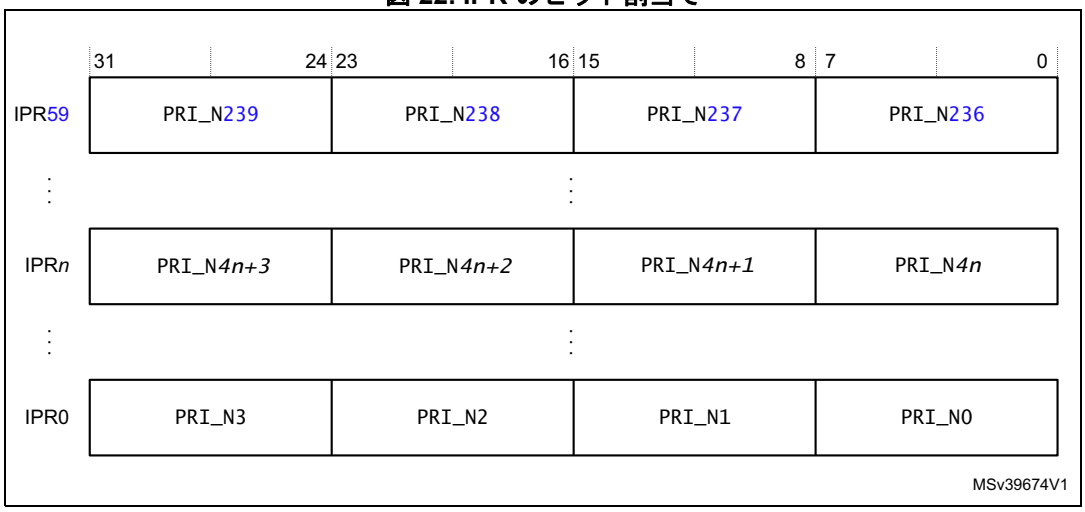


表 47. IPR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:24] | 優先度、バイト・オフセット 3 | 各優先度フィールドが 0 ~ 255 の優先度値を保持します。値が小さいほど対応する割込みの優先度が高くなります。プロセッサは各フィールドのビット [7:n] のみ実装し、ビット [n-1:0] は読み出す値は 0 となり、書込みは無視されます。 |
| [23:16] | 優先度、バイト・オフセット 2 | |
| [15:8] | 優先度、バイト・オフセット 1 | |
| [7:0] | 優先度、バイト・オフセット 0 | |

ソフトウェアから見た割込み優先順位を示す割込み優先順位配列へのアクセスの詳細については、182 ページのCMSIS を使用した Cortex®-M7 NVIC レジスタへのアクセスを参照してください。

割込み *m* の IPR 番号とバイト・オフセットは、次の手順に従って求めます。

- 対応する IPR 番号は 185 ページの表 46 参照。*n* は、*n* = *m* DIV 4 で与えられます。
- このレジスタの必要な優先度フィールドのバイト・オフセットは *m* MOD 4 です。ここで、
  - バイト・オフセット 0 はレジスタ・ビット [7:0] を参照します。
  - バイト・オフセット 1 はレジスタ・ビット [15:8] を参照します。
  - バイト・オフセット 2 はレジスタ・ビット [23:16] を参照します。
  - バイト・オフセット 3 はレジスタ・ビット [31:24] を参照します。

### 4.2.8 ソフトウェア・トリガ割込みレジスタ

ソフトウェアからの割込みを生成するには、STIR に書き込みます。STIR の属性については、181 ページの表 40 のレジスタの概要を参照してください。

SCR の USERSETMPEND ビットが 1 にセットされている場合、非特権ソフトウェアは STIR にアクセスできます（196 ページのシステム制御レジスタを参照）。

特権ソフトウェアのみが、STIR への非特権アクセスを有効にできます。

ビット割当てを次に示します。

図 23. STIR のビット割当て

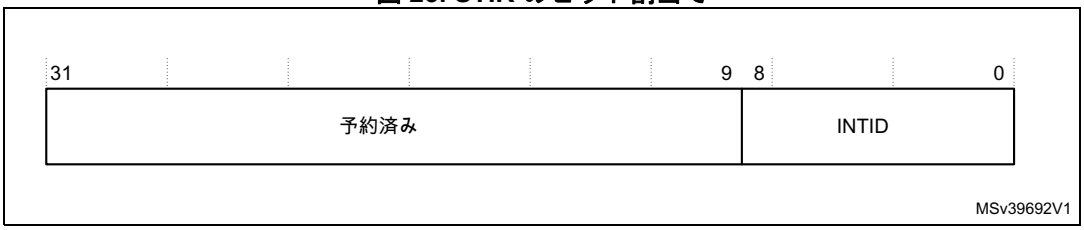



表 48. STIR ビット割当て

| ビット | フィールド | 機能 |
|---|---|---|
| [31:9] | - | 予約済み。 |
| [8:0] | INTID | トリガする割込みの割込み ID で、0 ~ 239 の範囲内です。たとえば、値 0x03 は、割込み IRQ3 を指定します。 |

### 4.2.9 レベル割込みとパルス割込み

プロセッサは、レベルセンシティブ割込みとパルス割込みの両方をサポートします。パルス割込みは、エッジトリガ割込みとも呼ばれます。

レベル検出割込みは、ペリフェラルが割込み信号をクリアするまでアサート状態を保持します。通常これは、ISR がペリフェラルにアクセスして、割込み要求をクリアさせるために発生します。パルス割込みは、プロセッサ·クロックの立ち上がりエッジに同期してサンプリングされた割込み信号です。NVIC が割込みを確実に検出するように、ペリフェラルは少なくとも 1 クロック·サイクルの間は割込み信号のアサートを保持する必要があります。その間に NVIC はパルスを検出して、割込みをラッチします。

プロセッサは、ISR に入ると、自動的に割込みの保留状態を解除します（187 ページのハードウェアとソフトウェアによる割込み制御を参照）。レベル検出割込みの場合、プロセッサが ISR から復帰する前に信号がネゲートされない場合、割込みは再び保留中になり、プロセッサはその ISR を再実行する必要があります。これは、割込み処理が不要になるまで、ペリフェラルは割込み信号のアサート状態を保持できることを意味します。

#### ハードウェアとソフトウェアによる割込み制御

- Cortex®-M7 は、すべての割込みをラッチします。ペリフェラル割込みは、次のいずれかの理由によって保留になります。
- NVIC が、割込み信号が HIGH であるのを検出し、割込みがアクティブではない場合。
- NVIC が割込み信号の立ち上がりエッジを検出した場合。
- ソフトウェアが、割込みセット保留レジスタの対応するビットに書き込む（183 ページの割込みセット・ペンディング・レジスタを参照）か、または STIR に書き込んで割込みを保留中にします（186 ページのソフトウェア·トリガ割込みレジスタを参照）。

保留中の割込みは、次のいずれかが発生するまで保留状態に維持されます。

- プロセッサが割込みの ISR に入る。これにより、割込みの状態が保留中からアクティブに変化します。その後、
  - レベル検出割込みの場合は、プロセッサが ISR から復帰する際に NVIC が割込み信号をサンプリングします。信号がアサートされている場合は、割込み状態が保留中に変化し、これによってプロセッサがただちに ISR を再開する可能性があります。アサートされていない場合は、割込み状態が非アクティブに変化します。
  - パルス割込みの場合、NVIC は割込み信号の監視を続行し、パルスが発生すると、割込み状態が保留中およびアクティブに変化します。この場合、プロセッサが ISR から復帰すると、割込み状態は保留中に変化し、これによってプロセッサがただちに ISR を再開する可能性があります。
  - プロセッサが ISR の処理中に割込み信号にパルスが発生しない場合、プロセッサが ISR から復帰すると、割込みの状態は非アクティブに変化します。
- ソフトウェアが、割込みクリア・ペンディング·レジスタの対応するビットに書き込みます。

  レベル検出割込みでは、割込み信号がまだアサートされている場合、割込み状態は変化しません。アサートされていない場合は、割込み状態が非アクティブに変化します。

  パルス割込みでは、割込み状態は次のように変化します。
  - 状態が保留中の場合、非アクティブ。
  - 状態がアクティブで保留中の場合、アクティブ

### 4.2.10 NVIC 設計のヒントとコツ

ソフトウェアが正しく整列されたレジスタ·アクセスを使用するようにします。プロセッサは、NVIC レジスタへのアンアライインド·アクセスをサポートしません。サポートされているアクセス·サイズについては、個々のレジスタの説明を参照してください。

割込みは、無効であっても保留状態に入ることができます。割込みを無効にすると、プロセッサが割込みを取得しなくなるだけです。

VTOR をプログラミングしてベクタ·テーブルを再配置する前に、新しいベクタ·テーブルのベクタ·テーブル·エントリが、フォールト·ハンドラ、NMIおよび割込みなどすべての有効な例外に対してセット・アップされていることを確認します。詳細については、194 ページのベクタ·テーブル·オフセット·レジスタを参照してください。

#### NVIC のプログラミングのヒント

ソフトウェアは、CPSIE I と CPSID I の各命令を使用して、割込みを有効化および無効化します。これらの命令に対して、CMSIS では次の組込み関数を用意しています。

```
void __disable_irq(void) // Disable Interrupts
void __enable_irq(void) // Enable Interrupts
```

さらに、CMSIS では以下のような多くの NVIC 制御関数を提供しています。

**表 49. CMSIS NVIC 制御関数**

| CMSIS 割込み制御関数 | 説明 |
|---|---|
| `void NVIC_SetPriorityGrouping(uint32_t priority_grouping)` | 優先度のグループ化を設定します。 |
| `void NVIC_EnableIRQ(IRQn_t IRQn)` | IRQn を有効にします。 |
| `void NVIC_DisableIRQ(IRQn_t IRQn)` | IRQn を無効にします。 |
| `uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)` | IRQn が保留中の場合に true （IRQ 番号）を返します。 |
| `void NVIC_SetPendingIRQ (IRQn_t IRQn)` | IRQn を保留中に設定します。 |
| `void NVIC_ClearPendingIRQ (IRQn_t IRQn)` | IRQn の保留ステータスをクリアします。 |
| `uint32_t NVIC_GetActive (IRQn_t IRQn)` | アクティブな割込みの IRQ 番号を返します。 |
| `void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)` | IRQn の優先度を設定します。 |
| `uint32_t NVIC_GetPriority (IRQn_t IRQn)` | IRQn の優先度を読み出します。 |

入力パラメータ IRQn は IRQ 番号です（40 ページの表 19 を参照）。これらの関数の詳細については、CMSIS ドキュメントを参照してください。

## 4.3 システム制御ブロック

*システム制御ブロック*（SCB）は、システム実装情報を提供し、システムを制御します。これには、システム例外の設定、制御、およびレポートが含まれます。システム制御ブロックのレジスタは、次のとおりです。

**表 50. システム制御ブロックのレジスタの概要**

| アドレス | 名前 | タイプ | 必要特権 | リセット値 | 説明 |
|---|---|---|---|---|---|
| 0xE000E008 | ACTLR | RW | 特権 | 0x00000000 | 190 ページの補助制御レジスタ |
| 0xE000ED00 | CPUID | RO | 特権 | 0x410FC270 | 191 ページのCPUID ベース・レジスタ |
| 0xE000ED04 | ICSR | RW(1) | 特権 | 0x00000000 | 191 ページの割込み制御およびステート・レジスタ |
| 0xE000ED08 | VTOR | RW | 特権 | 不明 | 194 ページのベクタ・テーブル・オフセット・レジスタ |
| 0xE000ED0C | AIRCR | RW (1) | 特権 | 0xFA050000 | 194 ページのアプリケーション割込みおよびリセット制御レジスタ |
| 0xE000ED10 | SCR | RW | 特権 | 0x00000000 | 196 ページのシステム制御レジスタ |
| 0xE000ED14 | CCR | RW | 特権 | 0x00000200 | 197 ページの設定および制御レジスタ |
| 0xE000ED18 | SHPR1 | RW | 特権 | 0x00000000 | 199 ページのシステム・ハンドラ優先度レジスタ 1 |
| 0xE000ED1C | SHPR2 | RW | 特権 | 0x00000000 | 200 ページのシステム・ハンドラ優先度レジスタ 2 |
| 0xE000ED20 | SHPR3 | RW | 特権 | 0x00000000 | 200 ページのシステム・ハンドラ優先度レジスタ 3 |
| 0xE000ED24 | SHCRS | RW | 特権 | 0x00000000 | 201 ページのシステム・ハンドラ制御およびステート・レジスタ |
| 0xE000ED28 | CFSR | RW | 特権 | 0x00000000 | 202 ページの設定可能なフォールト・ステータス・レジスタ |
| 0xE000ED28 | MMSR(2) | RW | 特権 | 0x00 | 208 ページのメモリ管理フォールト・アドレス・レジスタ |
| 0xE000ED29 | BFSR (2) | RW | 特権 | 0x00 | 204 ページのバス・フォールト・ステータス・レジスタ |
| 0xE000ED2A | UFSR (2) | RW | 特権 | 0x0000 | 190 ページの補助制御レジスタ |
| 0xE000ED2C | HFSR | RW | 特権 | 0x00000000 | 207 ページのハード・フォールト・ステータス・レジスタ |
| 0xE000ED34 | MMAR | RW | 特権 | 不明 | 208 ページのメモリ管理フォールト・アドレス・レジスタ |
| 0xE000ED38 | BFAR | RW | 特権 | 不明 | 209 ページのバス・フォールト・アドレス・レジスタ |
| 0xE000ED3C | AFSR | RAZ/WI | 特権 | - | 補助フォールト・ステータス・レジスタは実装されていません。 |

1. 詳細については、レジスタの説明を参照してください。
2. CFSR のサブレジスタ

### 4.3.1 補助制御レジスタ

ACTLR には、次のプロセッサ機能のディセーブル·ビットがあります。

- FPU 例外出力
- デュアル発行機能
- ITM および DWT からのトレース出力の一掃
- 動的読出し割当てモード

このレジスタは、デフォルトで Cortex®-M7 プロセッサから最適な性能を引き出すように設定されており、通常は変更する必要がありません。

ACTLR の属性については、189 ページの表 50 のレジスタの概要を参照してください。ビット割当てを次に示します。

**図 24. ACTLR のビット割当て**

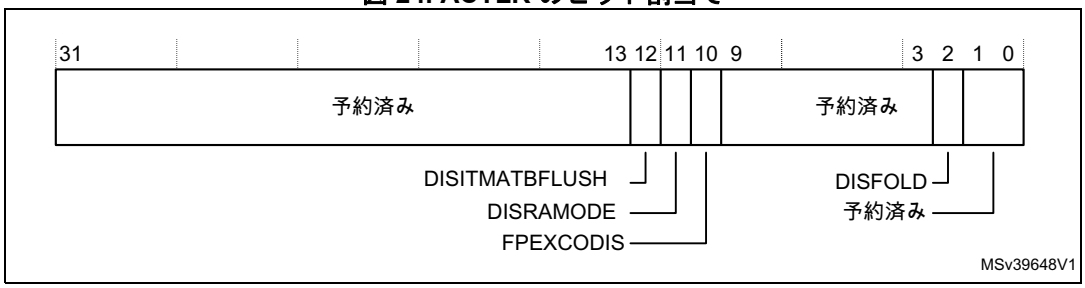



**表 51. ACTLR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:13] | - | 予約済みです。 |
| [12] | DISITMATBFLUSH | ITM および DWT ATB の一掃を無効にします。<br>0：通常動作。<br>1：ITM および DWT ATB の一掃は無効です。 |
| [11] | DISRAMODE | ライトバック、書込み割当てメモリ領域の動的読出し割当てモードを無効にします。<br>0：通常動作。<br>1：動的読出し割当てモードは無効です。 |
| [10] | FPEXCODIS | FPU 例外出力を無効にします。<br>0：通常動作。<br>1：FPU 例外出力は無効です。 |
| [9:3] | - | 予約済み。 |
| [2] | DISFOLD | 二重発行機能を無効にします。<br>0：通常動作。<br>1：二重発行機能は無効です。このビットをセットすると性能が低下します。 |
| [1:0] | - | 予約済み。 |

### 4.3.2 CPUID ベース・レジスタ

CPUID レジスタには、プロセッサの部品番号、バージョン、および実装情報が格納されます。属性については、189 ページの表 50 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 25. CPUID のビット割当て**

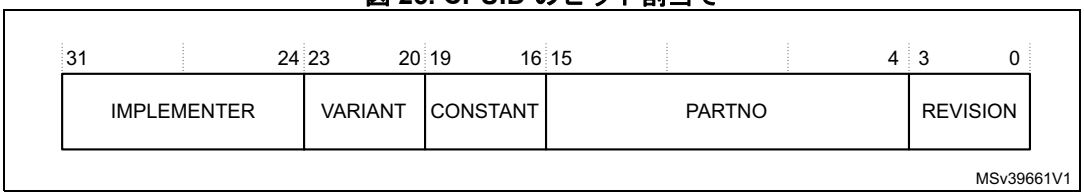



**表 52. CPUID ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:24] | IMPLEMENTER | 実装者コード<br>0x41 Arm |
| [23:20] | VARIANT | バリアント番号、製品リビジョン識別子 r*n*p*n* の r の値:<br>0x0 リビジョン 0 |
| [19:16] | CONSTANT | 0xF として読み出されます。 |
| [15:4] | PARTNO | プロセッサの部品番号 :<br>0xC27 : Cortex®-M7 |
| [3:0] | RIVISION | リビジョン番号、製品リビジョン識別子 r*n*p*n* p の値:<br>0x0 : パッチ 0 |

### 4.3.3 割込み制御およびステート・レジスタ

**ICSRの内容は以下のとおりです。**

- 提供するもの :
  - ノンマスカブル割込み（NMI）例外のセット・ペンディング・ビット。
  - PendSV 例外と SysTick 例外のセット・ペンディング・ビットとクリア・ペンディング・ビット。
- 内容 :
  - 処理中の例外の例外番号。
  - 横取りされたアクティブな例外の有無。
  - 保留中の例外の中で優先度が最も高い例外の番号
  - 保留中の割込みの有無

ICSR 属性については、189 ページの表 50 のレジスタの概要、および 192 ページの表 53 の型の説明を参照してください。ビット割当てを次に示します。

**図 26. ICSR のビット割当て**

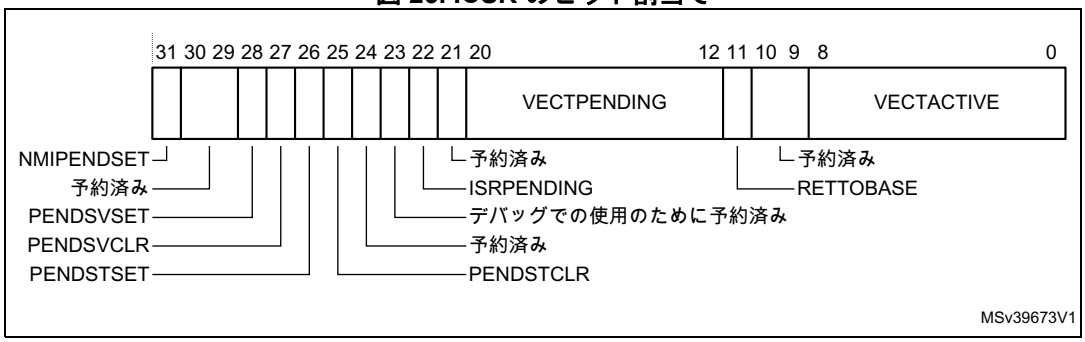



**表 53. ICSR のビット割当て**

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [31] | NMIPENDSET | RW | NMI セット・ペンディング・ビット。<br>書込み：<br>0：影響はありません。<br>1：NMI 例外の状態を保留中に変更します。<br>読出し：<br>0：NMI 例外は保留中ではありません。<br>1：NMI 例外は保留中です。<br>NMI は優先度が最も高い例外なので、通常はこのビットに 1 が書き込まれるとすぐにプロセッサが NMI 例外ハンドラを開始して、ハンドラが開始されるとこのビットが 0 にクリアされます。プロセッサがこのハンドラを実行中に NMI が再リリースされた場合、NMI 実行ハンドラで 1 が読めます。 |
| [30:29] | - | - | 予約済み。 |
| [28] | PENDSVSET | RW | PendSV セット・ペンディング・ビット。<br>書込み：<br>0：影響はありません。<br>1：PendSV 例外の状態を保留中に変更します。<br>読出し：<br>0：PendSV 例外は保留中ではありません。<br>1：PendSV 例外は保留中です。<br>PendSV 例外の状態を保留中にセットする唯一の方法が、このビットに 1 を書き込むことです。 |
| [27] | PENDSVCLR | WO | PendSV クリア・ペンディング・ビット。<br>書込み：<br>0：影響はありません。<br>1：PendSV 例外の保留状態を解除します。 |
| [26] | PENDSTSET | RW | SysTick 例外セット・ペンディング・ビット。<br>書込み：<br>0：影響はありません。<br>1：SysTick 例外の状態を保留中に変更します。<br>読出し：<br>0：SysTick 例外は保留中ではありません。<br>1：SysTick 例外は保留中です。 |

表 53. ICSR のビット割当て （続き）

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [25] | PENDSTCLR | WO | SysTick 例外クリア・ペンディング･ビット。<br>書込み：<br>0：影響はありません。<br>1：SysTick 例外の保留状態を解除します。<br>このビットは WO です。レジスタを読み出したときの値は不定です。 |
| [24] | - | - | 予約済み。 |
| [23] | デバッグでの使用のために予約済み | RO | このビットはデバッグ用に予約済みであり、プロセッサがデバッグ中ではない場合の読出し値は 0 です。 |
| [22] | ISRPENDING | RO | 割込み保留中フラグ。NMI とフォールトは除外します。<br>0：割込みは保留中ではありません。<br>1：割込みが保留中です。 |
| [21] | - | - | 予約済み。 |
| [20:12] | VECTPENDING | RO | 保留中の有効な例外の中で優先度が最も高い例外の番号を示します。<br>0：保留中の例外なし。<br>非ゼロ：保留中の有効な例外の中で優先度が最も高い例外の番号。<br>このフィールドによって示される値に、BASEPRI レジスタと FAULTMASK レジスタの影響は含まれますが、PRIMASK レジスタの影響は含まれません。 |
| [11] | RETTOBASE | RO | 横取りされたアクティブな例外の有無を示します。<br>0：実行すべき横取りされたアクティブな例外があります。<br>1：アクティブな例外がないか、または現在実行中の例外が唯一のアクティブな例外です。 |
| [10:9] | - | - | 予約済み。 |
| [8:0] | VECTACTIVE (1) | RO | アクティブな例外の番号が格納されます。<br>0：スレッド･モード<br>1：現在アクティブな例外の例外番号(1)。<br>この値から 16 を減算すると、CMSIS IRQ 番号が得られます。これは、割込みのクリア・イネーブル、セット・イネーブル、クリア・ペンディング、セット・ペンディング、または優先度の各レジスタへのインデックスになります（23 ページの表 5 を参照）。 |

1. これは IPSR ビット [8:0] と同じ値です（23 ページの割込みプログラム･ステータス･レジスタを参照）。

ICSR に書き込む際、次の操作をすると、効果は予測不能になります。

- PENDSVSET ビットに 1 を書き込み、PENDSVCLR ビットに 1 を書き込む。
- PENDSTSET ビットに 1 を書き込み、PENDSTCLR ビットに 1 を書き込む

### 4.3.4 ベクタ·テーブル·オフセット·レジスタ

VTOR は、メモリ・アドレス 0x00000000 からのベクタ・テーブル・ベース・アドレスのオフセットを示します。属性については、189 ページの表 50 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 27. VTOR のビット割当て**

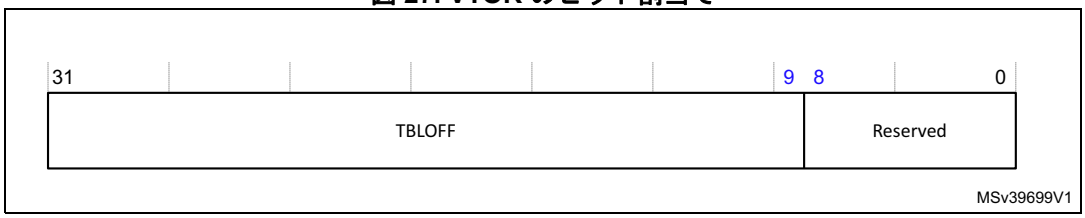



**表 54. VTOR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:9] | TBLOFF | ベクタ·テーブル·ベース·オフセット·フィールド。メモリ・マップのボトムからテーブル·ベースのオフセットのビット [29:7] が格納されています。 |
| [8:0] | - | 予約済み。 |

TBLOFF をセットする場合、オフセットはベクタ·テーブル内の例外エントリの数に基づいて整列される必要があります。

テーブル·アライメントの要件は、テーブル·オフセットのビット [8:0] が常に 0 であることを意味します。

### 4.3.5 アプリケーション割込みおよびリセット制御レジスタ

AIRCR は、例外モデルの優先度グループ化制御、データ·アクセスのエンディアン·ステータス、およびシステムのリセット制御に使用します。属性については、189 ページの表 50 および 195 ページの表 55 のレジスタ概要を参照してください。

このレジスタに書き込むには、VECTKEY フィールドに 0x5FA を書き込む必要があります。そうしないと、プロセッサによって書込みが無視されます。

ビット割当てを次に示します。

**図 28. AIRCR のビット割当て**

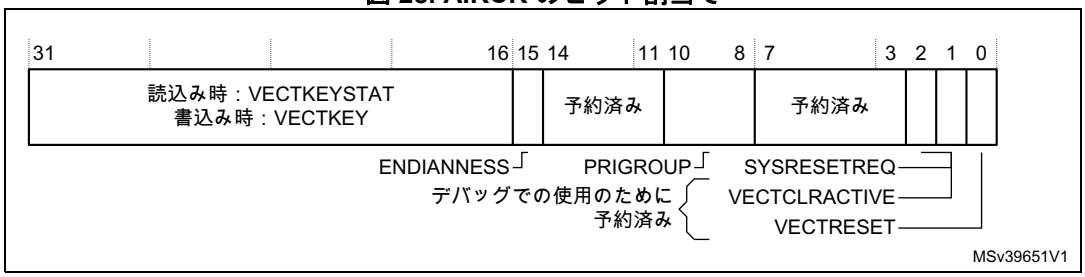


表 55. AIRCR のビット割当て

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [31:16] | 読出し：<br>VECTKEYSTAT<br>書込み：VECTKEY | RW | レジスタ・キー：<br>0xFA05 として読み出されます。<br>書込み時は、VECTKEY に 0x5FA を書き込みます。そうしないと、書込みは無視されます。 |
| [15] | ENDIANNESS | RO | データのエンディアン形式ビット：<br>0：リトルエンディアン |
| [14:11] | - | - | 予約済みです。 |
| [10:8] | PRIGROUP | RW | 割込み優先度グループ化フィールド。このフィールドは、グループ優先度とサブ優先度の分割を指定します（2 進小数点を参照）。 |
| [7:3] | - | - | 予約済み。 |
| [2] | SYSRESETREQ | WO | システム・リセット要求：<br>0：システム・リセット要求なし。<br>1：リセットを要求する外部システムに信号をアサートします。<br>デバッグを除くすべての主要コンポーネントに対する大規模なシステム・リセットを強制的に行うことを目的としています。<br>このビットは 0 として読み出されます。 |
| [1] | VECTCLRACTIVE | WO | デバッグでの使用のために予約済み。このビットは 0 として読み出されます。このレジスタに書き込むには、このビットに 0 を書き込む必要があります。そうしないと、動作は予測不能になります。 |
| [0] | VECTRESET | WO | デバッグでの使用のために予約済み。このビットは 0 として読み出されます。このレジスタに書き込むには、このビットに 0 を書き込む必要があります。そうしないと、動作は予測不能になります。 |

## 2 進小数点

PRIGROUP フィールドは、割込み優先度レジスタの PRI_n フィールドをグループ優先度フィールドとサブ優先度フィールドに分割する 2 進小数点の位置を示します。表 56 は、PRIGROUP 値による分割制御を示します。

実装する優先度ビットが 8 ビットより少ない場合についてさらに説明すると、テーブルから無効な行を削除して、グループ優先度の数とサブ優先度の *数の*列のエントリを変更することができます。

表 56. 優先度のグループ化

| | 割込み優先度レベル値、PRI_*N*[7:0] | | | 数 | |
|---|---|---|---|---|---|
| PRIGROUP | 2 進小数点(1) | グループ優先度ビット | サブ優先度ビット | グループ優先度 | サブ優先度 |
| 0b000 | bxxxxxxx.y | [7:1] | [0] | 128 | 2 |
| 0b001 | bxxxxxx.yy | [7:2] | [1:0] | 64 | 4 |
| 0b010 | bxxxxx.yyy | [7:3] | [2:0] | 32 | 8 |
| 0b011 | bxxxx.yyyy | [7:4] | [3:0] | 16 | 16 |
| 0b100 | bxxx.yyyyy | [7:5] | [4:0] | 8 | 32 |
| 0b101 | bxx.yyyyyy | [7:6] | [5:0] | 4 | 64 |

表 56. 優先度のグループ化 （続き）

| | 割込み優先度レベル値、PRI_N[7:0] | | | 数 | |
|---|---|---|---|---|---|
| PRIGROUP | 2 進小数点(1) | グループ優先度ビット | サブ優先度ビット | グループ優先度 | サブ優先度 |
| 0b110 | bx.yyyyyyy | [7] | [6:0] | 2 | 128 |
| 0b111 | b.yyyyyyyy | なし | [7:0] | 1 | 256 |

1. PRI_n[7:0] フィールドは 2 進小数点を示しています。x はグループ優先度フィールドのビットを、y はサブ優先度フィールドのビットを、それぞれ表します。

注： **例外の横取りを判断する場合はグループ優先度フィールドのみを使用します（43 ページの割込み優先度のグループ化を参照）。**

### 4.3.6 システム制御レジスタ

SCR は、低電力状態の開始と終了の機能を制御します。属性については、189 ページの表 50 のレジスタ概要を参照してください。ビット割当てを次に示します。

図 29. SCR ビット割当て：

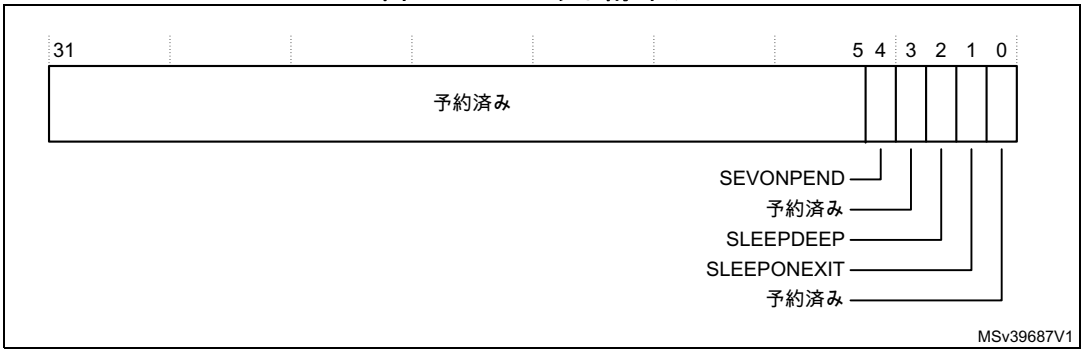



表 57. SCR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:5] | - | 予約済み。 |
| [4] | SEVONPEND | 保留ビット送信イベント<br>0：プロセッサをウェイクアップできるのは有効な割込みまたはイベントのみであり、無効な割込みは除外されます。<br>1：有効なイベントおよび無効な割込みを含むすべての割込みがプロセッサをウェイクアップできます。<br>イベントまたは割込みが保留状態になると、イベント信号がプロセッサを WFE からウェイクアップします。プロセッサがイベントを待機していない場合は、イベントが登録され、次の WFE に影響を及ぼします。<br>プロセッサは、SEV 命令の実行または外部イベントによってもウェイクアップします。 |
| [3] | - | 予約済み。 |

表 57. SCR ビット割当て （続き）

| ビット | 名前 | 機能 |
|---|---|---|
| [2] | SLEEPDEEP | プロセッサが低電力モードとしてスリープまたはディープ･スリープのどちらを使用するかを制御します。<br>0：スリープ<br>1：ディープ･スリープ |
| [1] | SLEEPONEXIT | ハンドラ･モードからスレッド･モードに復帰するときの sleep-on-exit を示します：<br>0：スレッド･モードに復帰するときにスリープに移行しません。<br>1：ISRから復帰するときにスリープまたはディープ･スリープに移行します。<br>このビットを 1 にセットすると、割込みで駆動するアプリケーションが空のメイン･アプリケーションに復帰することを回避できます。 |
| [0] | - | 予約済み。 |

## 4.3.7 設定および制御レジスタ

CCR は、スレッド･モードへの移行を制御し、次の動作を有効にします。

- NMI、ハード･フォールト、および FAULTMASK で昇格されたフォールトのハンドラがバス･フォールトを無視すること
- 0 による除算およびアンアライン ド･アクセスのトラップ
- 非特権ソフトウェアによる STIR へのアクセス（186 ページのソフトウェア･トリガ割込みレジスタ を参照）
- 命令およびデータ・キャッシュの有効化制御

CCR の属性については、189 ページの表 50 のレジスタの概要を参照してください。

ビット割当てを次に示します。

図 30. CCR のビット割当て

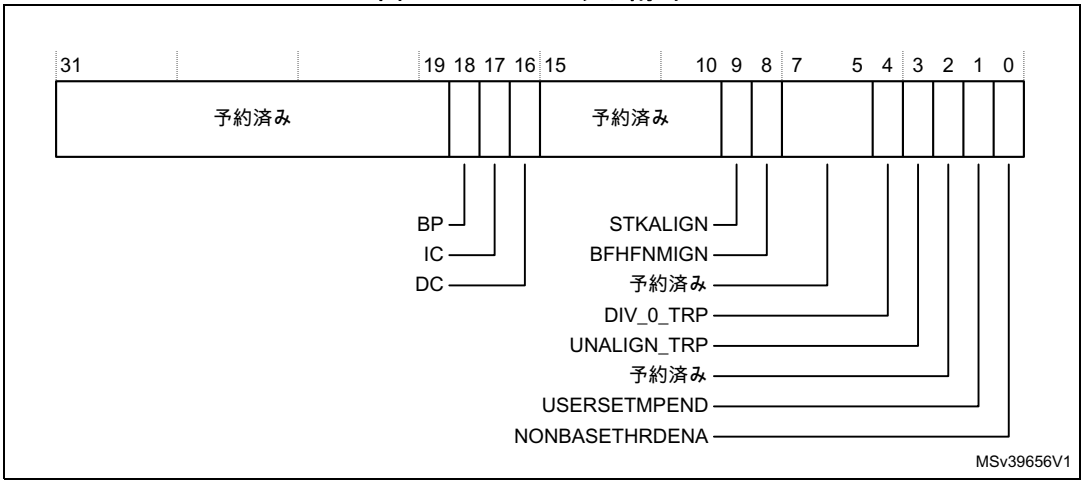



表 58. CCR ビット割当て

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [31:19] | - | - | 予約済み。 |
| [18] | BP | RO | 常に 1 つとして読み出します。分岐予測が有効であることを示します。 |

表 58. CCR ビット割当て （続き）

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [17] | IC | RW | L1 命令キャッシュを有効にします。<br>0：L1 命令キャッシュは無効です。<br>1：L1命令キャッシュは有効です。 |
| [16] | DC | RW | L1data キャッシュを有効にします。<br>0：L1 データ・キャッシュは無効です。<br>1：L1データ・キャッシュは有効です。 |
| [15:10] | - | - | 予約済み。 |
| [9] | STKALIGN | RO | 常に 1 つとして読み出します。例外開始時のスタック整列が 8 バイト整列であることを示します。<br>例外の開始時に、プロセッサはスタックされている PSR のビット [9] を使用してスタック･アライメントを示します。例外から復帰する時に、このスタック･ビットを使用して、正しいスタック･アライメントを復元します。 |
| [8] | BFHFNMIGN | RW | 優先度が -1 または -2 のハンドラは、ロードおよびストア命令によって発生するデータ･バス・フォールトを無視できるようにします。これは、ハード･フォールト、NMI、および FAULTMASK によって昇格されたハンドラに適用されます。<br>0：ロードおよびストア命令によって発生したデータ･バス･フォールトによりロックアップが生じます。<br>1：優先度が -1 または -2 で実行されているハンドラは、ロードおよびストア命令によって発生するデータ･バス･フォールトを無視します。<br>ハンドラとそのデータが絶対的に安全なメモリにある場合のみ、このビットを 1 にセットします。通常は、制御パスの問題の検出とその修正のためにシステム･デバイスやブリッジを調査するために、このビットを使用します。 |
| [7:5] | - | - | 予約済み。 |
| [4] | DIV_0_TRP | RW | プロセッサが 0 で除算する SDIV または UDIV 命令を実行したときに、フォールトを発生または停止できるようにします。<br>0：0 による除算をトラップしません。<br>1：0 による除算をトラップします。<br>このビットが 0 にセットされている場合、0 による除算は商 0 を返します。 |
| [3] | UNALIGN_TRP | RW | アンアライン ド･アクセスのトラップを有効にします。<br>0：ハーフワードおよびワードのアンアライン ド･アクセスをトラップしません。<br>1：ハーフワードおよびワードのアンアライン ド･アクセスをトラップします。<br>このビットが 1 にセットされている場合、アンアライン ド･アクセスによって用法フォールトが発生します。<br>整列されていない LDM、STM、LDRD、および STRD の各命令は、UNALIGN_TRP が 1 にセットされているかどうかに関係なく、常にフォールトを発生させます。 |
| [2] | - | - | 予約済み。 |

表 58. CCR ビット割当て （続き）

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [1] | USERSETMPEND | RW | 非特権ソフトウェアによる STIR へのアクセスを有効にします（186 ページのソフトウェア･トリガ割込みレジスタを参照）。<br>0：無効にします。<br>1：有効にします。 |
| [0] | NONBASETHRDENA | RW | プロセッサがスレッド･モードに移行する方法を表示します。<br>0：プロセッサは、アクティブな例外が存在しない場合のみ、スレッド･モードに移行できます。<br>1：プロセッサは、EXC_RETURN 値の制御の下で、任意のレベルからスレッド･モードに移行できます（46 ページの例外からの復帰を参照）。 |

## 4.3.8 システム･ハンドラ優先度レジスタ

SHPR1 ~ SHPR3 の各レジスタは、優先度を設定可能な例外ハンドラの優先度レベルを 0 ~ 255 の範囲で設定します。

SHPR1 ~ SHPR3 は、バイト単位でアクセスできます。属性については、189 ページの表 50 のレジスタ概要を参照してください。

システム･フォールト･ハンドラおよび各ハンドラの優先度フィールドとレジスタを次に示します。

表 59. システム･フォールト･ハンドラ優先度フィールド

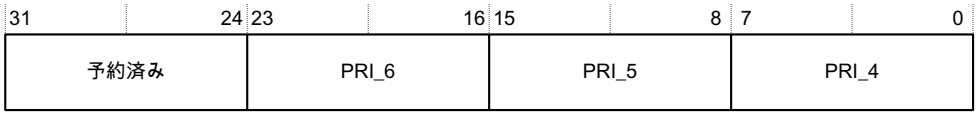
| ハンドラ | フィールド | レジスタの説明 |
|---|---|---|
| MemManage | PRI_4 | システム･ハンドラ優先度レジスタ 1 |
| BusFault | PRI_5 | |
| UsageFault | PRI_6 | |
| SVCall | PRI_11 | システム･ハンドラ優先度レジスタ 2 |
| PendSV | PRI_14 | システム･ハンドラ優先度レジスタ 3 |
| SysTick | PRI_15 | |

各 PRI_*n* フィールドは 8 ビット幅ですが、プロセッサは各フィールドのビット [7:M] のみ実装し、ビット [M-1:0] は 0 として読み出され、ここへの書込みは無視されます。

### システム･ハンドラ優先度レジスタ 1

ビット割当てを次に示します。

図 31. SHPR1 ビット割当て




MSv39689V1

表 60. SHPR1 レジスタのビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:24] | PRI_7 | 予約済みです。 |
| [23:16] | PRI_6 | システム･ハンドラ 6、用法フォールトの優先度 |
| [15:8] | PRI_5 | システム･ハンドラ 5、バス・フォールトの優先度 |
| [7:0] | PRI_4 | システム･ハンドラ 4、メモリ管理の優先度 |

### システム･ハンドラ優先度レジスタ 2

ビット割当てを次に示します。

図 32. SHPR2 ビット割当て

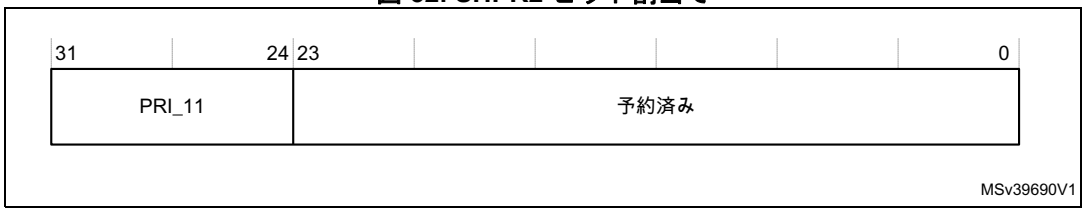



表 61. SHPR2 レジスタのビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:24] | PRI_11 | システム･ハンドラ 11、SVCall の優先度 |
| [23:0] | - | 予約済みです。 |

### システム･ハンドラ優先度レジスタ 3

ビット割当てを次に示します。

図 33. SHPR3 ビット割当て

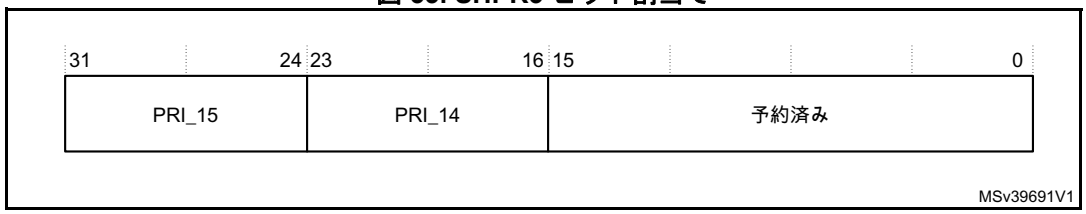



表 62. SHPR3 レジスタのビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:24] | PRI_15 | システム･ハンドラ 15、SysTick 例外の優先度 |
| [23:16] | PRI_14 | システム･ハンドラ 14、PendSV の優先度 |
| [15:0] | - | 予約済みです。 |

### 4.3.9 システム･ハンドラ制御およびステート・レジスタ

SHCSR は、システム･ハンドラを有効にし、以下を示します。

- BusFault、MemManage フォールト、および SVC 例外のペンディング・ステータス。
- システム･ハンドラのアクティブ･ステータス

SHCSR の属性については、189 ページの表 50 のレジスタの概要を参照してください。ビット割当てを次に示します。

**図 34. SHCSR のビット割当て**

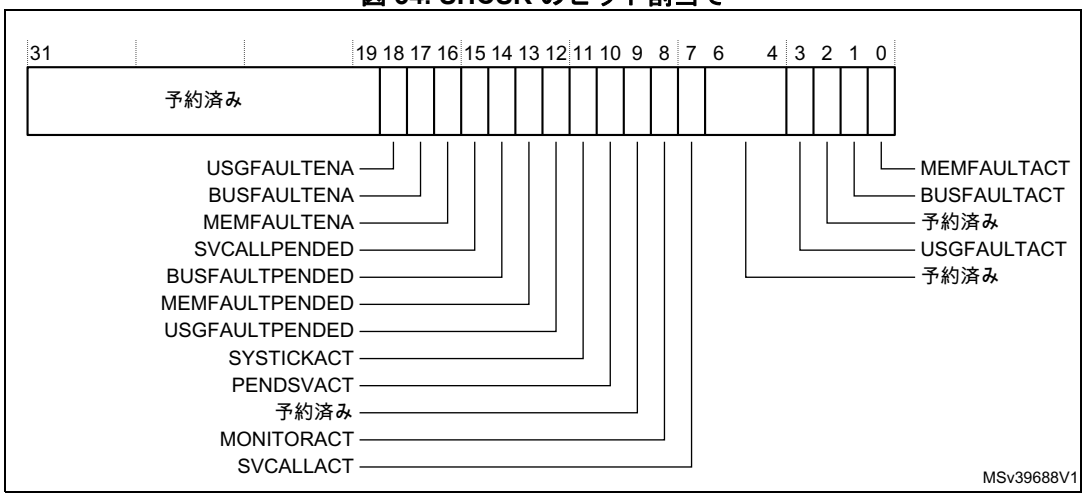



**表 63. SHCSR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:19] | - | 予約済みです。 |
| [18] | USGFAULTENA | 用法フォールト・イネーブル･ビット、有効にするには 1 をセット(1) |
| [17] | BUSFAULTENA | バス・フォールト・イネーブル･ビット、有効にするには 1 をセット (1) |
| [16] | MEMFAULTENA | メモリ管理イネーブル･ビット、有効にするには 1 をセット (1) |
| [15] | SVCALLPENDED | SVCall 保留ビット、例外が保留中の場合に 1 として読出し(2) |
| [14] | BUSFAULTPENDED | バス･フォールト例外保留ビット､例外が保留中の場合に 1 として読出し(2) |
| [13] | MEMFAULTPENDED | メモリ管理例外保留ビット、例外が保留中の場合に 1 として読出し (2) |
| [12] | USGFAULTPENDED | 用法フォールト例外保留ビット､例外が保留中の場合に 1 として読出し (2) |
| [11] | SYSTICKACT | SysTick 例外アクティブ･ビット、例外がアクティブの場合に 1 として読出し(3) |
| [10] | PENDSVACT | PendSV 例外アクティブ･ビット、例外がアクティブの場合に 1 として読出し |
| [9] | - | 予約済みです。 |
| [8] | MONITORACT | デバッグモニタ･アクティブ･ビット､デバッグモニタがアクティブの場合に 1 として読出し |
| [7] | SVCALLACT | SVCall アクティブ･ビット、SVC 呼出しがアクティブの場合に 1 として読出し |
| [6:4] | - | 予約済みです。 |

表 63. SHCSR ビット割当て （続き）

| ビット | 名前 | 機能 |
|---|---|---|
| [3] | USGFAULTACT | 用法フォールト例外アクティブ·ビット、例外がアクティブの場合に 1 として読出し |
| [2] | - | 予約済みです。 |
| [1] | BUSFAULTACT | バス・フォールト例外アクティブ·ビット、例外がアクティブの場合に 1 として読出し |
| [0] | MEMFAULTACT | メモリ管理例外アクティブ·ビット、例外がアクティブの場合に 1 として読出し |

1. イネーブル·ビットは、例外を有効にするには 1 をセット、無効にするには 0 をセットします。
2. 保留ビットは、例外が保留中の場合は 1 として読み出され、保留中ではない場合は 0 として読み出されます。これらのビットに書き込むことによって、例外の保留ステータスを変更できます。
3. アクティブ·ビットは、例外がアクティブの場合は 1 として読み出され、アクティブではない場合は 0 として読み出されます。これらのビットに書き込むことによって、例外のアクティブ·ステータスを変更できます。ただし、このセクションの注意を参照してください。

システム·ハンドラが無効なときに対応するフォールトが発生した場合、プロセッサはそのフォールトをハード·フォールトとして処理します。

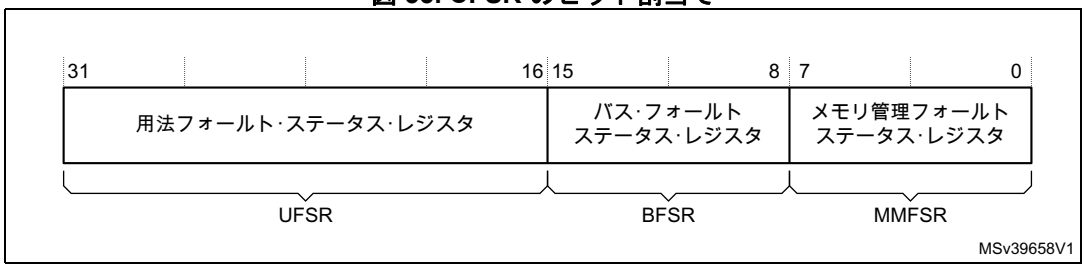
このレジスタに書き込むことによって、システム例外の保留ステータスまたはアクティブ·ステータスを変更できます。OS カーネルは、アクティブ·ビットを書き込むことによって、現在の例外タイプを変更するコンテキスト·スイッチを実行できます。

- ソフトウェアが、スタックされている内容を正しく調整しないでこのレジスタのアクティブ·ビットの値を変更すると、プロセッサでフォールト例外が発生する可能性があります。このレジスタに書き込むソフトウェアが、現在のアクティブ·ステータスを保持し、後で復元するようにしてください。
- システム·ハンドラを有効にした後で、このレジスタのビットの値を変更する必要がある場合、必要なビットだけが変更されるようにするには、読出し／変更／書込みを使用する必要があります。

### 4.3.10 設定可能なフォールト・ステータス・レジスタ

CFSR は、MemManage fault、BusFault、または UsageFault の原因を示します。属性については、189 ページの表 50 のレジスタ概要を参照してください。ビット割当てを次に示します。

図 35. CFSR のビット割当て




以下のサブセクションでは、CFSRを構成するサブレジスタについて説明します。

- 203 ページのメモリ管理フォールト·ステータス·レジスタを参照してください。
- 204 ページのバス·フォールト·ステータス·レジスタを参照してください。
- 206 ページの用法フォールト·ステータス·レジスタを参照してください。

CFSR は、バイト単位でアクセスできます。CFSR またはそのサブレジスタには、次のようにアクセスできます。

- CFSR 全体には 0xE000ED28 にワード･アクセスします。
- MMFSR には 0xE000ED28 にバイト･アクセスします。
- MMFSR と BFSR には 0xE000ED28 にハーフワード･アクセスします。
- BFSR には 0xE000ED29 にバイト･アクセスします。
- UFSR には 0xE000ED2A にハーフワード･アクセスします。

### メモリ管理フォールト･ステータス･レジスタ

MMFSR のフラグは、メモリアクセス異常の原因を示します。ビット割当てを次に示します。

**図 36. MMFSR のビット割当て**

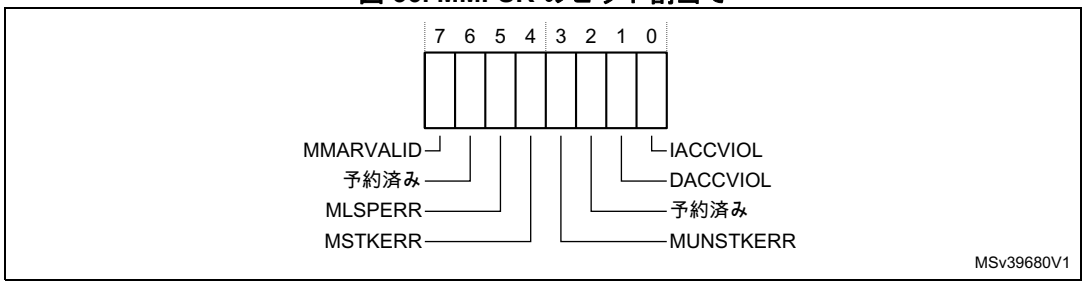



**表 64. MMFSR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [7] | MMARVALID | *MemManage Fault Address register*（メモリ管理フォールト･アドレス･レジスタ）（MMFAR）有効フラグ：<br>0：MMAR の値は有効なフォールト･アドレスではありません。<br>1：MMAR に有効なフォールト･アドレスが保持されています。<br>メモリ管理フォールトが発生して、優先度によってハード･フォールトに昇格された場合は、ハード･フォールト･ハンドラがこのビットを 0 にセットする必要があります。これにより、MMAR 値が上書きされているスタックされたアクティブなメモリ管理フォールト･ハンドラに戻った際の問題が回避されます。 |
| [6] | - | 予約済み。 |
| [5] | MLSPERR | 0：浮動小数点のレイジーな状態保持の際にメモリ管理フォールトは発生していません。<br>1：浮動小数点のレイジーな状態保持の際にメモリ管理フォールトが発生しました。 |
| [4] | MSTKERR | 例外開始時のスタッキングでのメモリ管理フォールト：<br>0：スタッキング･フォールトはありません。<br>1：例外開始時のスタッキングで 1 つまたは複数のアクセス違反が発生しました。<br>このビットが 1 の場合、SP は調整されたままですが、スタック内のコンテキスト領域の値は正しくない可能性があります。プロセッサはフォールト･アドレスを MMAR に書き込みません。 |
| [3] | MUNSTKERR | 例外から復帰する際のアンスタッキングでのメモリ管理フォールト。<br>0：アンスタッキング･フォールトはありません。<br>1：例外復帰時のアンスタッキングで 1 つまたは複数のアクセス違反が発生しました。<br>このフォールトは、ハンドラに関連付けられます。このことは、このビットが 1 のときに、元の復帰スタックがまだ存在することを意味します。プロセッサは、失敗した復帰に基づく SP の調整および新しい保存を行いません。プロセッサはフォールト･アドレスを MMAR に書き込みません。 |

表 64. MMFSR ビット割当て （続き）

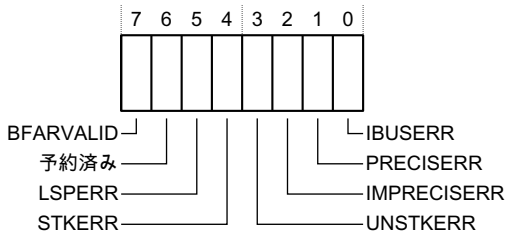
| ビット | 名前 | 機能 |
|---|---|---|
| [2] | - | 予約済みです。 |
| [1] | DACCVIOL | データ･アクセス違反フラグ：<br>0：データ･アクセス違反フォールトはありません。<br>1：プロセッサは、ロードまたはストアが許可されていない位置でそれらの操作を試みました。<br>このビットが 1 のとき、スタックされる例外復帰 PC の値は、フォール命令を指しています。プロセッサは、アクセスしようとしたアドレスを MMAR にロードしました。 |
| [0] | IACCVIOL | 命令アクセス違反フラグ：<br>0：命令アクセス違反フォールトはありません。<br>1：プロセッサは、命令フェッチが許可されていない位置でその操作を試みました。<br>このフォールトは、MPU が無効な場合または存在しない場合であっても、XN 領域へのあらゆるアクセスで発生します。<br>このビットが 1 のとき、スタックされる例外復帰 PC の値は、フォール命令を指しています。プロセッサはフォールト･アドレスを MMAR に書き込みません。 |

MMFSR ビットはスティッキーです。つまり、フォールトが発生すると、関連するビットが 1 にセットされます。1 にセットされたビットは、そのビットに 1 を書き込むことによって、またはリセットによってのみ 0 にクリアされます。

### バス･フォールト･ステータス･レジスタ

BFSR のフラグは、バス・アクセス・フォールトの原因を示します。ビット割当てを次に示します。

図 37. BFSR のビット割当て




MSv39653V1

表 65. BFSR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [7] | BFARVALID | *BusFault Address register*（バス・フォールト・アドレス･レジスタ）（BFAR）有効フラグ：<br>0：BFAR の値は有効なフォールト･アドレスではありません。<br>1：BFAR に有効なフォールト･アドレスが保持されています。<br>アドレスが既知であるバス・フォールトが発生すると、プロセッサはこのビットを 1 にセットします。後で発生するメモリ管理フォールトなどの他のフォールトがこのビットを 0 にセットする可能性があります。<br>バス・フォールトが発生して、優先度によってハード･フォールトに昇格された場合は、ハード･フォールト･ハンドラがこのビットを 0 にセットする必要があります。これにより、BFAR 値が上書きされているスタックされたアクティブなバス・フォールト･ハンドラに戻った場合、問題が回避されます。 |
| [6] | - | 予約済み。 |

表 65. BFSR ビット割当て （続き）

| ビット | 名前 | 機能 |
|---|---|---|
| [5] | LSPERR | 0：浮動小数点のレイジーな状態保持の際にバス·フォールトは発生していません。<br>1：浮動小数点のレイジーな状態保持の際にバス·フォールトが発生しました。 |
| [4] | STKERR | 例外開始時のスタッキングでのバス・フォールト：<br>0：スタッキング·フォールトはありません。<br>1：例外開始時のスタッキングで 1 つまたは複数のバス・フォールトが発生しました。<br>プロセッサがこのビットを 1 にセットすると、SP は調整されたままですが、スタック内のコンテキスト領域の値は正しくない可能性があります。プロセッサはフォールト·アドレスを BFAR に書き込みません。 |
| [3] | UNSTKERR | 例外から復帰する際のアンスタッキングでのバス・フォールト。<br>0：アンスタッキング·フォールトはありません。<br>1：例外復帰時のアンスタッキングで 1 つまたは複数のバス·フォールトが発生しました。<br>このフォールトは、ハンドラに関連付けられます。このことは、プロセッサがこのビットを 1 にセットする際に、元の復帰スタックがまだ存在することを意味します。プロセッサは、失敗した復帰に基づく SP の調整、新しい保存、および BFAR へのフォールト·アドレスの書込みを行いません。 |
| [2] | IMPRECISERR | 不正確なデータ·バス·エラー：<br>0：不正確なデータ·バス·エラーはありません。<br>1：データ·バス·エラーが発生しましたが、スタック·フレームの復帰アドレスは、エラーの原因となった命令とは関係ありません。<br>プロセッサは、このビットを 1 にセットする際にフォールト·アドレスを BFAR に書き込みません。<br>これは非同期フォールトです。従って、現在のプロセスの優先度がバス·フォールトの優先度より高いときにこのバス·フォールトが検出された場合、このバス・フォールトは保留中になり、アクティブになるのはプロセッサがより高い優先度のすべてのプロセスから復帰してからとなります。プロセッサが不正確なバス・フォールトのハンドラを開始する前に正確なフォールトが発生した場合、ハンドラは、IMPRECISERR が 1 にセットされていることと、いずれかの正確なフォールトのステータス·ビットが 1 にセットされていることの両方を検出することになります。 |
| [1] | PRECISERR | 正確なデータ·バス·エラー：<br>0：正確なデータ·バス·エラーはありません。<br>1：データ·バス·エラーが発生し、スタックされる例外復帰 PC の値は、フォールトの原因となった命令を指しています。<br>プロセッサは、このビットを 1 にセットする際にフォールト·アドレスを BFAR に書き込みます。 |
| [0] | IBUSERR | 命令バス·エラー：<br>0：命令バス·エラーはありません。<br>1：命令バス·エラー。<br>プロセッサは、命令をプリフェッチする際に命令バス·エラーを検出しますが、そのフォールト命令を発行しようとする場合のみ、IBUSERR フラグを 1 にセットします。<br>プロセッサは、このビットを 1 にセットする際にフォールト·アドレスを BFAR に書き込みません。 |

BFSR ビットはスティッキーです。つまり、フォールトが発生すると、関連するビットが 1 にセットされます。1 にセットされたビットは、そのビットに 1 を書き込むことによって、またはリセットによってのみ 0 にクリアされます。

### 用法フォールト･ステータス･レジスタ

UFSR は、用法フォールトの原因を示します。ビット割当てを次に示します。

**図 38. UFSR のビット割当て**

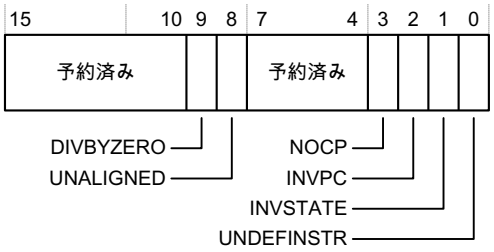



**表 66. UFSR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [15:10] | - | 予約済み。 |
| [9] | DIVBYZERO | 0 による除算の用法フォールト：<br>0：0 による除算のフォールトがないか、または0 による除算のトラップが無効です。<br>1：プロセッサが除数 0 で SDIV または UDIV 命令を実行しました。<br>プロセッサがこのビットを 1 にセットしている場合、スタックされる例外復帰 PC の値は、0 による除算を実行した命令を指しています。<br>CCR の DIV_0_TRP ビットを 1 にセットして、0 による除算のトラップを有効にします（197 ページの設定および制御レジスタを参照）。 |
| [8] | UNALIGNED | アンアラインド･アクセスの用法フォールト：<br>0：アンアラインド･アクセスによるフォールトがないか、またはアンアラインド･アクセスのトラップが有効ではありません。<br>1：プロセッサがアンアラインド･メモリ･アクセスを実行しました。<br>CCR の UNALIGN_TRP ビットを 1 にセットして、アンアラインド･アクセスのトラップを有効にします（197 ページの設定および制御レジスタを参照）。<br>整列されていない LDM、STM、LDRD、および STRD の各命令は、UNALIGN_TRP の設定に関係なく、常にフォールトを発生させます。 |
| [7:4] | - | 予約済み。 |
| [3] | NOCP | コプロセッサ用法フォールトはありません：<br>0：コプロセッサにアクセスしようとしたことによる用法フォールトは発生していません。<br>1：プロセッサがコプロセッサにアクセスしようとしました。<br>プロセッサは、コプロセッサ命令をサポートしません。 |
| [2] | INVPC | 無効な PC のロードによる用法フォールト。EXC_RETURN による無効な PC のロードにより発生します。<br>0：無効な PC のロードによる用法フォールトはありません。<br>1：無効なコンテキストまたは無効な EXC_RETURN 値の結果として、プロセッサが PC への EXC_RETURN の不正ロードを試みました。<br>このビットが 1 にセットされている場合、スタックされる例外復帰 PC の値は、PC の不正なロードを実行しようとした命令を指しています。 |

表 66. UFSR ビット割当て （続き）

| ビット | 名前 | 機能 |
|---|---|---|
| [1] | INVSTATE | 無効な状態による用法フォールト：<br>0：無効な状態による用法フォールトはありません。<br>1：プロセッサが EPSR を不正に使用する命令を実行しようとしました。<br>このビットが 1 にセットされている場合、スタックされる例外復帰 PC の値は、EPSR を不正に使用しようとした命令を指しています。<br>未定義命令が EPSR を使用する場合、このビットは 1 にセットされません。 |
| [0] | UNDEFINSTR | 未定義命令による用法フォールト：<br>0：未定義命令による用法フォールトはありません。<br>1：プロセッサが未定義命令を実行しようとしました。<br>このビットが 1 にセットされている場合、スタックされる例外復帰 PC の値は、未定義命令を指しています。<br>未定義命令は、プロセッサがデコードできない命令です。 |

UFSR ビットはスティッキーです。つまり、フォールトが発生すると、関連するビットが 1 にセットされます。1 にセットされたビットは、そのビットに 1 を書き込むことによって、またはリセットによってのみ 0 にクリアされます。

### 4.3.11 ハード・フォールト・ステータス・レジスタ

HFSR は、ハード・フォールト・ハンドラをアクティブにしたイベントに関する情報を提供します。属性については、189 ページの表 50 のレジスタ概要を参照してください。

このレジスタへのアクセスは、読出しと書込みクリアです。これは、このレジスタのビットは普通に読み出されるが、ビットに 1 を書き込むとそのビットが 0 にクリアされることを意味します。ビット割当てを次に示します。

図 39. HFSR のビット割当て

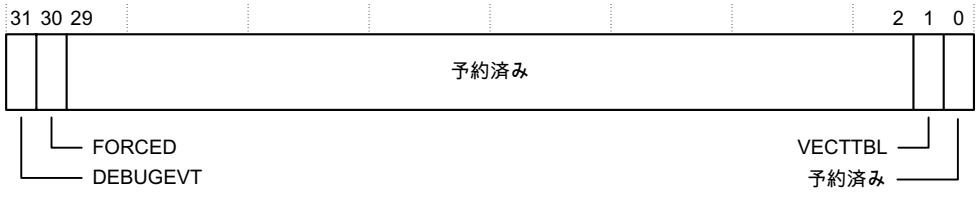



表 67. HFSR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31] | DEBUGEVT | デバッグでの使用のために予約済み。このレジスタに書き込むには、このビットに 1 を書き込む必要があります。そうしないと、動作は予測不能になります。 |
| [30] | FORCED | 優先度を設定可能なフォールトが、優先度により、またはそれが無効であることが理由で処理できないために昇格されたことによって生成された強制ハード・フォールトを示します。<br>0：強制ハード・フォルトはありません。<br>1：強制ハード・フォールトがあります。<br>このビットが 1 にセットされている場合、ハード・フォールト・ハンドラは、他のフォールト・ステータス・レジスタを読み出して、フォールトの原因を見つける必要があります。 |

表 67. HFSR ビット割当て （続き）

| ビット | 名前 | 機能 |
|---|---|---|
| [29:2] | - | 予約済み。 |
| [1] | VECTTBL | 例外処理中にベクタ·テーブルの読出しでバス・フォールトが発生したことを示します：<br>0：ベクタ·テーブル読出し時のバス・フォールトはありません。<br>1：ベクタテーブル読出し時のバス・フォールトがあります。<br>このエラーは常に、ハード·フォールト·ハンドラによって処理されます。<br>このビットが 1 にセットされている場合、スタックされる例外復帰 PC の値は、例外によって横取りされた命令を指しています。 |
| [0] | - | 予約済み。 |

HFSR ビットはスティッキーです。つまり、フォールトが発生すると、関連するビットが 1 にセットされます。1 にセットされたビットは、そのビットに 1 を書き込むことによって、またはリセットによってのみ 0 にクリアされます。

### 4.3.12 メモリ管理フォールト・アドレス・レジスタ

MMFAR には、メモリ管理フォールトを生成した位置のアドレスが含まれています。属性については、189 ページの表 50 のレジスタ概要を参照してください。ビット割当てを次に示します。

表 68. MMFAR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:0] | ADDRESS | MMFSR の MMARVALID ビットが 1 にセットされている場合、このフィールドはメモリ管理フォールトを生成した位置のアドレスを保持します。 |

アンアライン ド·アクセスによるフォールトの場合、このアドレスはフォールトが発生した実際のアドレスです。1 つの読出し命令または書込み命令が複数のアライン ド·アクセスに分割される可能性があるので、フォールト·アドレスは、要求されたアクセス·サイズの範囲内の任意のアドレスとなります。

MMFSR のフラグは、フォールトの原因および MMFAR の値が有効かどうかを示します。197 ページの設定および制御レジスタを参照してください。

### 4.3.13 バス･フォールト･アドレス･レジスタ

BFAR には、バス･フォールトを生成した位置のアドレスが含まれています。属性については、189 ページの表 50 のレジスタ概要を参照してください。ビット割当てを次に示します。

**表 69. BFAR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:0] | ADDRESS | BFSR の BFARVALID ビットが 1 にセットされている場合、このフィールドはバス･フォールトを生成した位置のアドレスを保持します。 |

アンアライインド･アクセスによるフォールトが発生した場合、BFAR のアドレスは、命令によって要求されたアドレスになります（たとえこのアドレスがフォールト･アドレスではなくても）。

BFSR のフラグは、フォールトの原因および BFAR の値が有効かどうかを示します。189 ページの表 50 を参照してください。

### 4.3.14 システム制御ブロック設計のヒントとコツ

システム制御ブロックのレジスタにアクセスに適したサイズのアライインド･アクセスを使用するようにします。

- CFSR および SHPR1 ~ SHPR3 以外は、ワードでのアライインド･アクセスを使用する必要があります。
- CFSR および SHPR1 ~ SHPR3 の場合、バイト･アクセスを使用するか、ハーフワードまたはワードでのアライインド･アクセスを使用できます。

プロセッサは、システム制御ブロックのレジスタへのアンアライインド･アクセスをサポートしません。

フォールト･ハンドラで実際のフォールト･アドレスを調べるには、次の手順を実行します。

1. MMFAR または BFAR の値を読み出して保存します。
2. MMFSR の MMARVALID ビットまたは BFSR の BFARVALID ビットを読み出します。MMFAR または BFAR のアドレスは、このビットが 1 の場合のみ有効です。

それより優先度の高い例外によって MMFAR または BFAR の値が変更される可能性があるため、ソフトウェアはこの手順に従う必要があります。たとえば、優先度のより高いハンドラが現在のフォールト･ハンドラから横取りした場合、このフォールトによって MMFAR または BFAR の値が変更される可能性があります。

さらに、CMSIS では以下のような多くのシステム制御関数を提供しています。

**表 70. システム制御のための CMSIS 関数**

| CMSIS システム制御関数 | 説明 |
|---|---|
| `void NVIC_SystemReset (void)` | システムをリセットします。 |

## 4.4 システム・タイマ、SysTick

プロセッサには 24 ビットのシステム･タイマ SysTick があります。このタイマは、再ロード値から 0 にカウント･ダウンし、次のクロック･エッジで SYST_RVR レジスタの値を再ロード、（つまりラップ）した後、後続のクロックでカウント･ダウンします。

プロセッサがデバッグで停止している間は、カウンタはデクリメントしません。

システム・タイマのレジスタは、次のとおりです。

**表 71. システム･タイマのレジスタの概要**

| アドレス | 名前 | タイプ | 必要特権 | リセット値 | 説明 |
|---|---|---|---|---|---|
| 0xE000E010 | SYST_CSR | RW | 特権 | 0x00000004 | SysTick 制御およびステータス･レジスタ |
| 0xE000E014 | SYST_RVR | RW | 特権 | 不定 | SysTick RELOAD 値レジスタ |
| 0xE000E018 | SYST_CVR | RW | 特権 | 不定 | SysTick 現在値レジスタ |
| 0xE000E01C | SYST_CALIB | RO | 特権 | 0xC0000000 | SysTick 較正値レジスタ |

### 4.4.1 SysTick 制御およびステータス･レジスタ

SysTick SYST_CSR レジスタは、SysTick の機能を有効にします。属性については、表 71 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 40. SysTick SYST_CSR ビット割当て**

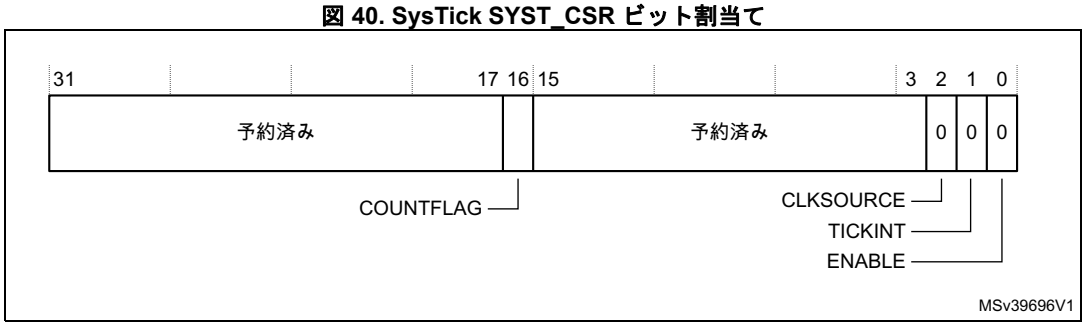



**表 72. SysTick SYST_CSR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:17] | - | 予約済み。 |
| [16] | COUNTFLAG | 最後の読出しの後にタイマが 0 になった場合、1 を返します。 |
| [15:3] | - | 予約済み。 |
| [2] | CLKSOURCE | クロックソースを示します：<br>– 0：外部クロック<br>– 1：プロセッサクロック |

表 72. SysTick SYST_CSR ビット割当て （続き）

| ビット | 名前 | 機能 |
|---|---|---|
| [1] | TICKINT | SysTick 例外リクエストを有効にします。<br>0：0 までカウント･ダウンしても SysTick 例外要求をアサートしません。<br>1：0 までカウント･ダウンしたら SysTick 例外要求をアサートします。<br>ソフトウェアは、COUNTFLAG を使用して、SysTick が 0 までカウント･ダウンされたかどうかを判断できます。 |
| [0] | ENABLE | カウンタを有効にします：<br>0：カウンタは無効です。<br>1：カウンタは有効です。 |

ENABLE が 1 にセットされている場合、カウンタは SYST_RVR レジスタから RELOAD 値をロードしてからカウント･ダウンします。0 に到達すると、COUNTFLAG を 1 にセットして、任意で TICKINT の値に応じて SysTick をアサートします。次に、RELOAD 値を再ロードしてからカウントを始めます。

## 4.4.2 SysTick RELOAD 値レジスタ

SYST_RVR レジスタは、SYST_CVR レジスタにロードする開始値を指定します。属性については、210 ページの表 71 のレジスタ概要を参照してください。ビット割当てを次に示します。

図 41. SYST_RVR ビット割当て

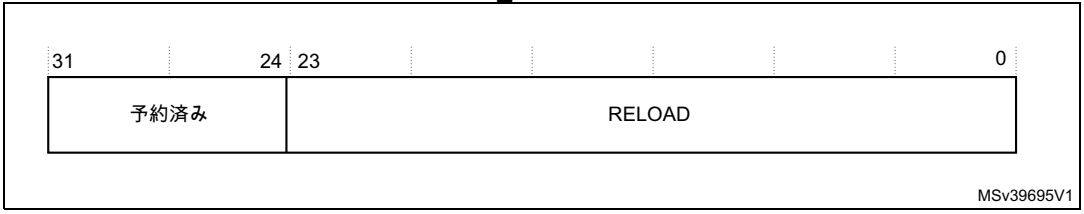



表 73. SYST_RVR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:24] | - | 予約済み。 |
| [23:0] | RELOAD | カウンタが有効で 0 に達するときに SYST_CVR レジスタにロードする値。RELOAD 値の計算を参照。 |

### RELOAD 値の計算

RELOAD 値には、0x00000001 ~ 0x00FFFFFF の範囲の任意の値を指定できます。開始値を 0 にすることは可能ですが、SysTick 例外要求と COUNTFLAG は、1 から 0 にカウントするときにアクティブになるため、効果はありません。

RELOAD 値は、その使い方に従って計算されます。例えば、周期 N のプロセッサ･クロック･サイクルのマルチショット･タイマを生成するには、RELOAD 値として N-1 を使用します。100 クロック･パルスごとに SysTick 割込みが必要な場合、RELOAD を 99 にセットします。

### 4.4.3 SysTick 現在値レジスタ

SYST_CVR レジスタには、SysTick カウンタの現在値が格納されます。属性については、210 ページの表 71 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 42. SYST_CVR ビット割当て：**

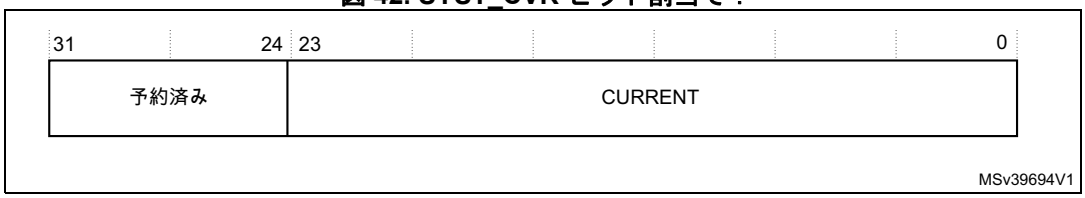



**表 74. SYST_CVR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:24] | - | 予約済み。 |
| [23:0] | CURRENT | 読み出すと、SysTick カウンタの現在値が返されます。<br>任意の値の書込みにより、このフィールドは 0 にクリアされ、さらに SYST_CSR COUNTFLAG ビットが 0 にクリアされます。 |

### 4.4.4 SysTick 較正値レジスタ

SYST_CALIB レジスタは、SysTick 較正プロパティを示します。属性については、210 ページの表 71 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 43. SYST_CALIB ビット割当て**

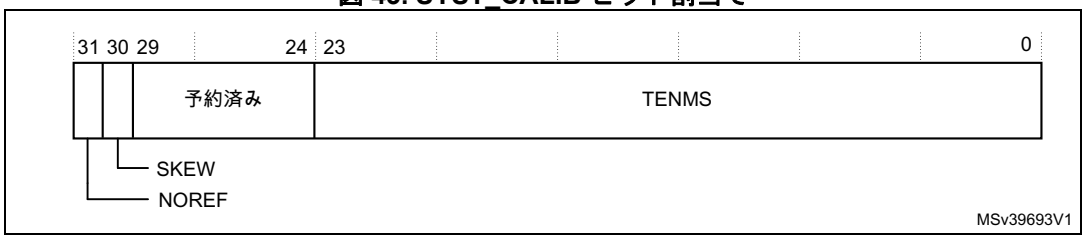



**表 75. SYST_CALIB ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31] | NOREF | デバイスがプロセッサにリファレンスクロックを供給するかどうかを示します。<br>0：リファレンスクロックが供給されています。<br>1：リファレンスクロックは供給されていません。<br>デバイスがリファレンスクロックを供給しない場合、SYST_CSR.CLKSOURCE ビットは 1 として読み出され、書込みは無視されます。 |
| [30] | SKEW | TENMS 値が正確かどうかを示します：<br>0：TENMS 値は正確です。<br>1：TENMS 値が不正確であるか、または与えられていません。<br>不正確な TENMS 値が、ソフトウェア・リアルタイム·クロックとしての SysTick の適合性に影響を及ぼす可能性があります。 |
| [29:24] | - | 予約済み。 |
| [23:0] | TENMS | 10ms（100Hz）のタイミングでの再ロード値です。システム・クロック・スキュー誤差の影響を受けます。値がゼロを示している場合、較正値は不明です。 |

較正情報が不明な場合は、プロセッサ·クロックまたは外部クロックの周波数から、必要な較正値を計算します。

### 4.4.5 SysTick 設計のヒントとコツ

SysTick カウンタは、プロセッサ·クロックを使用して動作します。低電力モードでこのクロック信号が停止した場合、SysTick カウンタは停止します。

ソフトウェアが、ワードでのアラインド·アクセスを使用して、SysTick のレジスタにアクセスするようにします。

SysTick カウンタの再ロード値と現在値はリセット時には未定義です。SysTick カウンタの正しい初期化シーケンスを次に示します。

1. 再ロード値をプログラムします。
2. 現在値をクリアします。
3. 制御およびステータス·レジスタをプログラムします。

さらに、CMSIS では以下のような多くの SysTick 制御関数を提供しています。

**表 76. SysTick 制御のための CMSIS 関数**

| CMSIS SysTick制御関数 | 説明 |
|---|---|
| `uint32_t SysTick_Config(uint32_t ticks)` | SysTick タイマを使用し、ticks パラメータで定義されたインターバルで周期的な SysTick 割り込みを生成します。 |

## 4.5 プロセッサの機能

プロセッサ機能レジスタは、ソフトウェアにキャッシュ設定情報を提供します。識別スペース・レジスタは、次のとおりです。

**表 77. 識別スペースの概要**

| アドレス | 名前 | タイプ | 必要特権 | リセット値 | 説明 |
|---|---|---|---|---|---|
| 0xE000ED78 | CLIDR | RO | 特権 | 0x09000003 | キャッシュ・レベル ID レジスタ |
| 0xE000ED7C | CTR | RO | 特権 | 0x8303C003 | 215 ページのキャッシュ・タイプ・レジスタ |
| 0xE000ED80 | CCSIDR | RO | 特権 | 不明 | 216 ページのキャッシュ・サイズ ID レジスタ |
| 0xE000ED84 | CSSELR | RW | 特権 | 不明 | 217 ページのキャッシュ・サイズ選択レジスタ |

すべてのレジスタには、特権ロードおよび特権ストアからのみアクセスできます。これらのレジスタに非特権アクセスを行うと、バス・フォールトが発生します。

### 4.5.1 キャッシュ・レベル ID レジスタ

CLIDR は、各レベルに実装されているキャッシュのタイプと、コヒーレンシと統合のレベルを識別します。属性については、214 ページの表 77 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 44. CLIDR のビット割当て**

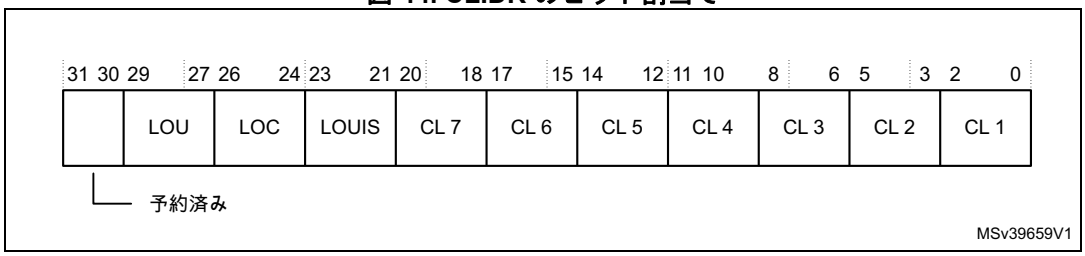



**表 78. CLIDR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:30] | - | SBZ. |
| [29:27] | LOU | 統合のレベル<br>0b001：レベル 2（いずれかのキャッシュが実装されている場合）。<br>0b000：レベル 1（命令キャッシュもデータ・キャッシュも実装されていない場合）。 |
| [26:24] | LOC | コヒーレンシのレベル<br>0b001：レベル 2（いずれかのキャッシュが実装されている場合）。<br>0b000：レベル 1（命令キャッシュもデータ・キャッシュも実装されていない場合）。 |
| [23:21] | LOUIS | RAZ. |
| [20:18] | CL 7 | 0b000：CL 7 はキャッシュなし。 |
| [17:15] | CL 6 | 0b000：CL 6 はキャッシュなし。 |
| [14:12] | CL 5 | 0b000：CL 5 はキャッシュなし。 |
| [11:9] | CL 4 | 0b000：CL 4 はキャッシュなし。 |

表 78. CLIDR ビット割当て （続き）

| ビット | 名前 | 機能 |
|---|---|---|
| [8:6] | CL 3 | 0b000：CL 3 はキャッシュなし。 |
| [5:3] | CL 2 | 0b000：CL 2 はキャッシュなし。 |
| [2] | CL 1 | RAZ.CL1 にユニファイド・キャッシュがないことを示します。 |
| [1] | CL 1 | 1：データ・キャッシュが実装されています。<br>0：データ・キャッシュは実装されていません。 |
| [0] | CL 1 | 1：命令キャッシュが実装されています。<br>0：命令キャッシュは実装されていません。 |

### 4.5.2 キャッシュ・タイプ・レジスタ

CTR は、キャッシュ・アーキテクチャに関する情報を提供します。属性については、214 ページの表 77 のレジスタ概要を参照してください。ビット割当てを次に示します。

図 45. CTR のビット割当て

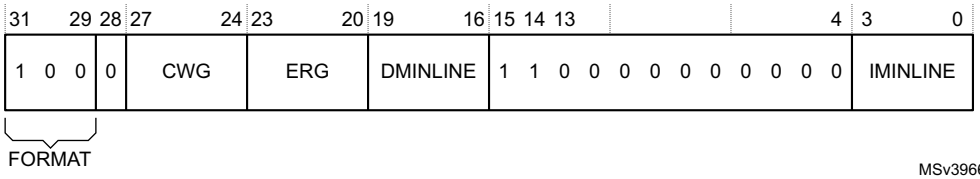



表 79. CTR ビット割当て

| ビット | 名前 | 説明 |
|---|---|---|
| [31:29] | FORMAT | レジスタ・フォーマット<br>0b100：Armv7 レジスタ・フォーマット。 |
| [28] | - | 予約済み、RAZ |
| [27:24] | CWG | Cache Writeback Granule（キャッシュ・ライト・バックの単位）<br>0b0011：Cortex®-M7 プロセッサは 8 ワード単位。 |
| [23:20] | ERG | Exclusives Reservation Granule（排他的予約の単位）<br>0b0000：プロセッサ内のローカル・モニタは、物理アドレスを保持しません。あらゆる STREX 命令アクセスを、前の LDREX 命令のアドレスと一致するとして処理します。これは、実装される排他的予約の単位がメモリアドレス範囲全体であることを意味します。 |
| [19:16] | DMINLINE | コアが制御するすべてのデータおよびユニファイド・キャッシュの中で最小のキャッシュライン。<br>0b0011：Cortex®-M7 プロセッサ の 8 ワード。 |
| [15:14] | - | すべてのビットが RAO |
| [13:4] | - | 予約済み、RAZ |
| [3:0] | IMINLINE | プロセッサが制御するすべての命令キャッシュの最小キャッシュライン。<br>0b0011：Cortex®-M7 プロセッサ の 8 ワード。 |

### 4.5.3 キャッシュ・サイズ ID レジスタ

CCSIDR は、CSSELR によって現在選択されているキャッシュの設定を示します。命令キャッシュまたはデータ・キャッシュが設定されない場合、対応する CCSIDR は RAZ です。属性については、214 ページの表 77 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 46. CCSIDR のビット割当て**

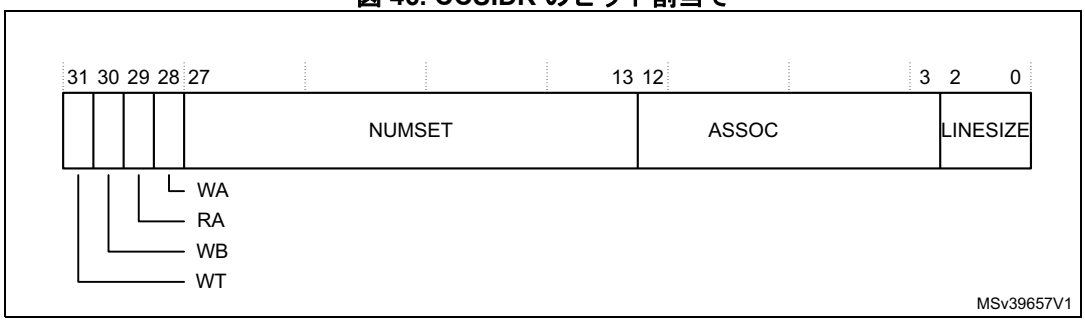



**表 80. CCSIDR ビット割当て**

| ビット | 名前 | 機能[1] |
|---|---|---|
| [31] | WT | ライトスルーに使用可能なサポートを示します。<br>1：ライトスルーのサポートが利用可能です。 |
| [30] | WB | ライトバックに使用可能なサポートを示します。<br>1：ライトバックのサポートが使用可能です。 |
| [29] | RA | 読出し割当てに使用可能なサポートを示します。<br>1：読出し割当てのサポートが使用可能です。 |
| [28] | WA | 書込み割当てに使用可能なサポートを示します。<br>1：書込み割当てのサポートが使用可能です。 |
| [27:13] | NUMSET | セット数を次のように示します。<br>（セット数）- 1。 |
| [12:3] | ASSOC | ウェイ数を次のように示します。<br>（ウェイ数）- 1。 |
| [2:0] | LINESIZE | 各キャッシュラインのワード数を示します。 |

1. 有効なビットフィールドのエンコードは 217 ページの表 81 を参照してください。

LineSize フィールドは、キャッシュライン内のワード数の log(2) より 2 少ないものとしてエンコードされます。たとえば、値 0x0 はキャッシュラインに 4 ワードがあることを示します。これはキャッシュの最小サイズです。値 0x1 は、キャッシュラインに 8 ワードがあることを示します。

表 81 に、CCSIDR の個別のビット・フィールドとレジスタの完全なエンコードを示します。これを使って、*Cache Size Selection Register*（キャッシュサイズ選択レジスタ）（CSSELR）によって選択された L1 データまたは命令キャッシュのキャッシュサイズを決定します。キャッシュ・サイズ選択レジスタを参照してください。

表 81. CCSIDR エンコード

| CSSELR | キャッシュ | サイズ | レジスタの完全なエンコード | レジスタ・ビット・フィールドのエンコード | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | WT | WB | RA | WA | NumSets | アソシアティブ | LineSize |
| 0x0 | データ・キャッシュ | 4 KB | 0xF003E019 | 1 | 1 | 1 | 1 | 0x001F | 0x3 | 0x1 |
| | | 8 KB | 0xF007E019 | | | | | 0x003F | | |
| | | 16 KB | 0xF00FE019 | | | | | 0x007F | | |
| | | 32 KB | 0xF01FE019 | | | | | 0x00FF | | |
| | | 64 KB | 0xF03FE019 | | | | | 0x01FF | | |
| 0x1 | 命令キャッシュ | 4 KB | 0xF007E009 | 1 | 1 | 1 | 1 | 0x003F | 0x1 | 0x1 |
| | | 8 KB | 0xF00FE009 | | | | | 0x007F | | |
| | | 16 KB | 0xF01FE009 | | | | | 0x00FF | | |
| | | 32 KB | 0xF03FE009 | | | | | 0x01FF | | |
| | | 64 KB | 0xF07FE009 | | | | | 0x03FF | | |

### 4.5.4 キャッシュ・サイズ選択レジスタ

CSSELR は、現在 CCSIDR に設定が表示されているキャッシュを選択します。属性は 214 ページの表 77 のレジストリの概要を参照してください。ビット割当ては次のとおりです。

図 47. CSSELR のビット割当て

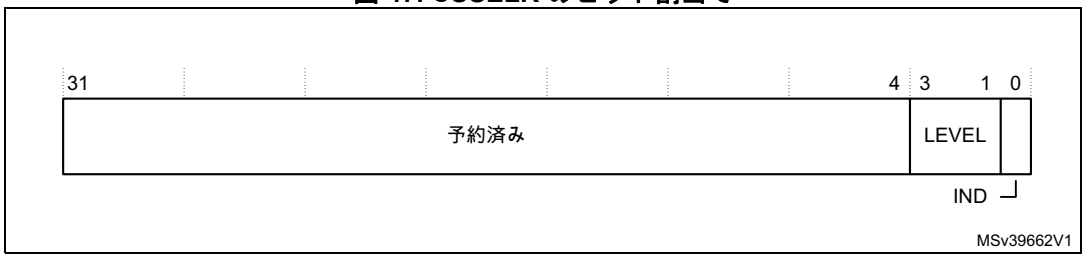



表 82. CSSELR ビット割当て

| ビット | 名前 | 説明 |
|---|---|---|
| [31:4] | - | 予約済み |
| [3:1] | LEVEL | 選択されたキャッシュ・レベルを示します。<br>0b000：レベル 1キャッシュ<br>このフィールドは読出し専用で、書込みは無視されます。 |
| [0] | IND | 命令キャッシュまたはデータ・キャッシュの選択を有効にします。<br>0：データ・キャッシュ<br>1：命令キャッシュ |

## 4.6 メモリ保護ユニット

*Memory Protection Unit*（メモリ保護ユニット）（MPU）は、メモリ・マップを複数の領域に分割して、各領域の位置、サイズ、アクセス許可、およびメモリ属性を定義します。以下をサポートしています。

- 領域ごとに独立した属性設定。
- 領域のオーバーラップ。
- システムへのメモリ属性のエクスポート

メモリ属性は、領域へのメモリ･アクセスの動作に影響を及ぼします。Cortex®-M7 MPU は次を定義します。

- 8 または 16 の個別のメモリ領域（0 ~ 7 または 0 ~ 15）
- バックグラウンド領域

メモリ領域がオーバーラップする場合、番号が最も大きい領域の属性がメモリ･アクセスに影響を及ぼします。たとえば、領域 7 の属性は、領域 7 とオーバーラップするどの領域の属性よりも優先されます。

バックグラウンド領域は、デフォルトのメモリ･マップと同じメモリ･アクセス属性を持ちますが、特権ソフトウェアからのみアクセス可能です。

Cortex®-M7 MPU のメモリ･マップは統合されています。これは、命令アクセスとデータ･アクセスが同じ領域設定を持っていることを意味します。

プログラムが MPU によって禁止されているメモリ領域にアクセスすると、プロセッサはメモリ管理フォールトを生成します。これによりフォールト例外が発生し、OS 環境内でプロセスが終了する可能性があります。OS 環境内では、カーネルが、実行するプロセスに応じて動的に MPU 領域設定を更新できます。通常、組込み OS は MPU を使用してメモリを保護します。

MPU 領域は、メモリ･タイプに基づいて設定されます（33 ページのメモリの領域、タイプ、および属性を参照）。

表 83 に、使用可能な MPU 領域属性を示します。これらには、一般にプロセッサにキャッシュが設定されている場合にのみ関係する共有可能性とキャッシュ動作の属性が含まれます。

**表 83. メモリ属性の概要**

| メモリタイプ | 共有可能性 | その他の属性 | 説明 |
|---|---|---|---|
| Strongly-ordered | - | - | Strongly-ordered メモリに対するアクセスは、すべてプログラム順に発生します。すべての Strongly-ordered 領域は、共有されることを前提としています。 |
| デバイス | 共有 | - | 複数のプロセッサで共有されるメモリマッピングされたペリフェラル。 |
| - | 非共有 | - | 単一プロセッサのみが使用するメモリマッピングされたペリフェラル。 |
| ノーマル | 共有 | キャッシュ不可能なライトスルー、キャッシュ可能なライトバックキャッシュ可能 | 複数のプロセッサで共有されるノーマル･メモリ。 |
| - | 非共有 | キャッシュ不可能なライトスルー、キャッシュ可能なライトバックキャッシュ可能 | 単一プロセッサのみが使用するノーマル･メモリ。 |

MPU レジスタを使用して、MPU 領域とその属性を定義します。MPU レジスタは、次のとおりです。

**表 84. MPU レジスタの概要**

| アドレス | 名前 | タイプ | 必要特権 | リセット値 | 説明 |
|---|---|---|---|---|---|
| 0xE000ED90 | MPU_TYPER | RO | 特権 | 0x00000800 | 220 ページのMPU タイプ・レジスタ |
| 0xE000ED94 | MPU_CTRL | RW | 特権 | 0x00000000 | 220 ページのMPU 制御レジスタ |
| 0xE000ED98 | MPU_RNR | RW | 特権 | 不明 | 222 ページのMPU 領域番号レジスタ |
| 0xE000ED9C | MPU_RBAR | RW | 特権 | 不明 | 222 ページのMPU 領域ベース･アドレス･レジスタ |
| 0xE000EDA0 | MPU_RASR | RW | 特権 | -(1) | 223 ページのMPU 領域属性およびサイズ･レジスタ |
| 0xE000EDA4 | MPU_RBAR_A1 | RW | 特権 | 不明 | RBAR のエイリアス。222 ページのMPU 領域ベース･アドレス･レジスタ を参照してください。 |
| 0xE000EDA8 | MPU_RASR_A1 | RW | 特権 | -(1) | RASR のエイリアス。223 ページのMPU 領域属性およびサイズ･レジスタ を参照してください。 |
| 0xE000EDAC | MPU_RBAR_A2 | RW | 特権 | 不明 | RBAR のエイリアス。222 ページのMPU 領域ベース･アドレス･レジスタ を参照してください。 |
| 0xE000EDB0 | MPU_RASR_A2 | RW | 特権 | -(1) | RASR のエイリアス。223 ページのMPU 領域属性およびサイズ･レジスタ を参照してください。 |
| 0xE000EDB4 | MPU_RBAR_A3 | RW | 特権 | 不明 | RBAR のエイリアス。222 ページのMPU 領域ベース･アドレス･レジスタ を参照してください。 |
| 0xE000EDB8 | MPU_RASR_A3 | RW | 特権 | -(1) | RASR のエイリアス。223 ページのMPU 領域属性およびサイズ･レジスタ を参照してください。 |

1. 0 にリセットされる ENABLE フィールド以外は不明です。

### 4.6.1 MPU タイプ・レジスタ

MPU_TYPE レジスタは、MPU が存在しているかどうか、存在している場合はサポートする領域の数を示します。MPU が存在しない場合、MPU_TYPE レジスタは RAZ です。属性については、表 84 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 48. TYPE のビット割当て**

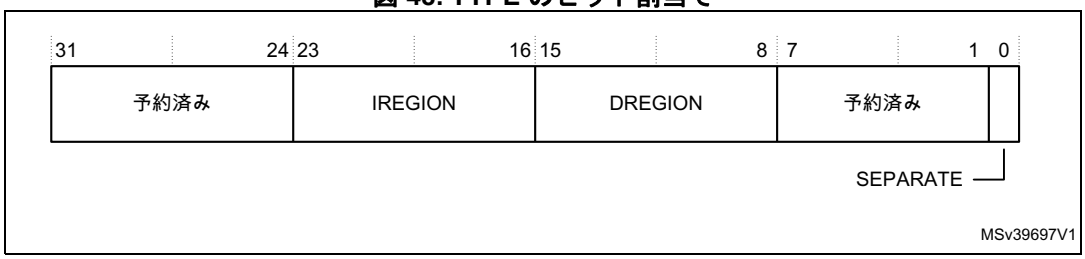



**表 85. TYPE のビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:24] | - | 予約済み。 |
| [23:16] | IREGION | サポートされている MPU 命令領域の数を示します。<br>常に 0x00 を含みます。MPU のメモリ･マップは統合されており、DREGION フィールドに記述されます。 |
| [15:8] | DREGION | サポートされている MPU データ領域の数を示します:<br>0x08：8 MPU 領域<br>0x10：16 MPU 領域 |
| [7:1] | - | 予約済み。 |
| [0] | SEPARATE | 命令とデータのメモリ･マップとして統合マップまたは個別マップのどちらをサポートしているかを示します。<br>0：統合 |

### 4.6.2 MPU 制御レジスタ

MPU_CTRL レジスタは、次の目的に使用します。

- MPU を有効にします。
- デフォルトのメモリ･マップのバックグラウンド領域を有効にします。
- ハード･フォールト、*Non Maskable Interrupt*（ノンマスカブル割込み）（NMI）、および FAULTMASK で昇格されたハンドラ内での MPU の使用を有効にします。

MPU_CTRL の属性については、219 ページの表 84 のレジスタの概要を参照してください。ビット割当てを次に示します。

**図 49. MPU_CTRL ビット割当て**

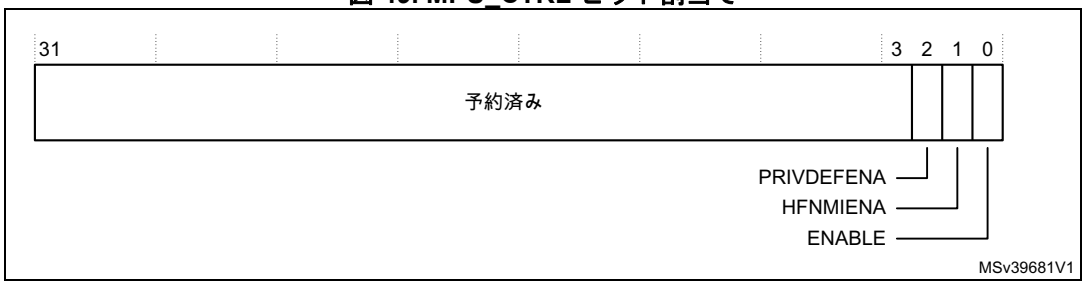



**表 86. MPU_CTRL ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:3] | - | 予約済み。 |
| [2] | PRIVDEFENA | デフォルトのメモリ･マップへの特権ソフトウェアによるアクセスを有効にします：<br>0：MPU が有効な場合、デフォルトのメモリ･マップの使用を無効にします。有効な領域に含まれない位置へのメモリ･アクセスではフォールトが発生します。<br>1：MPU が有効な場合、特権ソフトウェアによるアクセスで、バックグラウンド領域としてデフォルトのメモリ･マップを使用できるようになります。<br>バックグラウンド領域が有効な場合、領域番号 -1 であるかのように動作します。定義されている有効な領域は、このデフォルトのマップより優先されます。<br>MPU が無効な場合、プロセッサはこのビットを無視します。 |
| [1] | HFNMIENA | ハード･フォールト、NMI、および FAULTMASK の各ハンドラの実行中の MPU の動作を有効にします。<br>MPU が有効な場合：<br>0：ENABLE ビットの値に関係なく、ハード･フォールト、NMI、および FAULTMASK の各ハンドラの実行中、MPU は無効です。<br>1：ハード･フォールト、NMI、および FAULTMASK の各ハンドラの実行中、MPU は有効です。<br>MPU が無効で、このビットが 1 にセットされている場合、動作は予測不能です。 |
| [0] | ENABLE | MPU を有効にします：<br>0：MPU は無効です。<br>1：MPU は有効です。 |

ENABLE と PRIVDEFENA の両方が 1 にセットされている場合：

- 特権アクセスの場合、デフォルトのメモリ･マップは 30 ページのCortex®-M7 の構成で説明されているとおりです。特権ソフトウェアによるアクセスで有効なメモリ領域がアドレス指定されていない場合、デフォルトのメモリ･マップによって定義されている動作が行われます。
- 非特権ソフトウェアによるアクセスで有効なメモリ領域がアドレス指定されていない場合、メモリ管理フォールトが発生します。

システム制御空間には、ENABLE ビットの値に関係なく、常に XN と Strongly-ordered ルールが適用されます。

ENABLE ビットが 1 にセットされている場合、PRIVDEFENA ビットが 1 にセットされている場合を除いて、システムが機能するには、メモリ･マップの少なくとも 1 つの領域が有効である必要があります。PRIVDEFENA ビットが 1 にセットされていて、有効な領域が存在しない場合は、特権ソフトウェアのみ動作できます。

ENABLE ビットが 0 にセットされている場合、システムはデフォルトのメモリ･マップを使用します。この場合、MPU が実装されていない場合と同じメモリ属性になります（214 ページの表 77 を参照）。

デフォルトのメモリ･マップは、特権ソフトウェアと非特権ソフトウェアの両方のアクセスに適用されます。

MPU が有効な場合、システム制御空間とベクタ･テーブルへのアクセスは常に許可されます。その他の領域は、その領域と、PRIVDEFENA が 1 にセットされているかどうかに応じてアクセスが可能になります。

HFNMIENA が 1 にセットされている場合を除いて、プロセッサが優先度 -1 または -2 で例外ハンドラを実行中の場合、MPU は有効ではありません。これらの優先度が可能なのは、ハード･フォールトまたは NMI 例外を処理中の場合または FAULTMASK が有効な場合のみです。それらの 2 つの優先度で動作中の場合、HFNMIENA ビットを 1 にセットすると、MPU が有効になります。

### 4.6.3 MPU 領域番号レジスタ

MPU_RNR は、MPU_RBAR レジスタと MPU_RASR レジスタが参照するメモリ領域を選択します。属性については、214 ページの表 77 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 50. MPU_RNR ビット割当て**

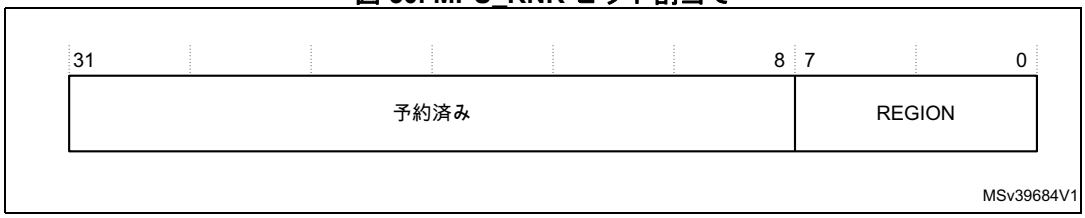



**表 87. MPU_RNR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:8] | - | 予約済み。 |
| [7:0] | REGION | MPU_RBAR レジスタと MPU_RASR レジスタが参照する MPU 領域を示します。<br>MPU は 8 個または 16 個のメモリ領域をサポートするので、このフィールドには 0 ~ 7 または 0 ~ 15 の値を指定できます。 |

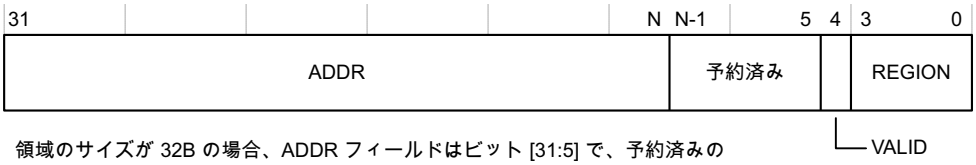
通常は、MPU_RBAR または MPU_RASR にアクセスする前に、このレジスタに必要な領域番号を書き込みます。ただし、VALID ビットを 1 にセットして MPU RBARに書き込むことで領域番号を変更できます（MPU 領域ベース･アドレス･レジスタを参照）。この書込みにより、REGION フィールドの値が更新されます。

### 4.6.4 MPU 領域ベース･アドレス･レジスタ

MPU_RBAR は、MPU_RNR によって選択されている MPU 領域のベース･アドレスを定義し、MPU_RNR の値を更新できます。属性については、219 ページの表 84 のレジスタ概要を参照してください。

VALID ビットを 1 にセットして MPU_RBAR に書き込むことによって、現在の領域番号を変更し、MPU_RNR を更新します。属性については、219 ページの表 84 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 51. MPU_RBAR ビット割当て：**




領域のサイズが 32B の場合、ADDR フィールドはビット [31:5] で、予約済みのフィールドはありません。

MSv39683V1

**表 88. MPU_RBAR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:N] | ADDR | 領域ベース·アドレス·フィールド。N の値は、領域サイズによって異なります。詳細については、ADDR フィールドを参照してください。 |
| [(N-1):5] | - | 予約済み。 |
| [4] | VALID | MPU 領域番号有効ビット：<br>書込み：<br>0：MPU_RNR は変更されません。プロセッサは次のように動作します。<br>MPU_RNR で指定されている領域のベース·アドレスを更新します。<br>REGION フィールドの値を無視します。<br>1：プロセッサは、以下のとおりです。<br>MPU_RNR の値を REGION フィールドの値に更新します。<br>REGION フィールドで指定されている領域のベース·アドレスを更新します。<br>常に 0 が読み出されます。 |
| [3:0] | REGION | MPU 領域フィールド：<br>書込み時の動作については、VALID フィールドの説明を参照してください。<br>読出し時は、RNR で指定されている現在の領域番号を返します。 |

### ADDR フィールド

ADDR フィールドは、MPU_RBAR のビット [31:N] です。MPU_RASR の SIZE フィールドで指定される領域サイズによって、N の値は次のように定義されます。

$N = \log_2$（領域サイズ（バイト単位））

MPU_RASR で領域サイズが 4 GB に設定されている場合、有効な ADDR フィールドはありません。この場合、この領域がメモリ·マップ全体を占有し、ベース·アドレスは 0x00000000 です。

ベース·アドレスは、領域のサイズに整列されます。たとえば、64KB の領域は、64KB の倍数（0x00010000、0x00020000 など）で整列されている必要があります。

## 4.6.5 MPU 領域属性およびサイズ·レジスタ

MPU_RASR は、MPU_RNR で指定されている MPU 領域の領域サイズとメモリ属性を定義し、その領域とサブ領域を有効にします。属性については、219 ページの表 84 のレジスタ概要を参照してください。

MPU_RASR は、ワードアクセスが可能です。

- 上位ハーフワードは、領域属性を保持します。
- 下位ハーフワードは、領域サイズおよび領域とサブ領域のイネーブル·ビットを保持します。

ビット割当てを次に示します。

図 52. MPU_RASR ビット割当て

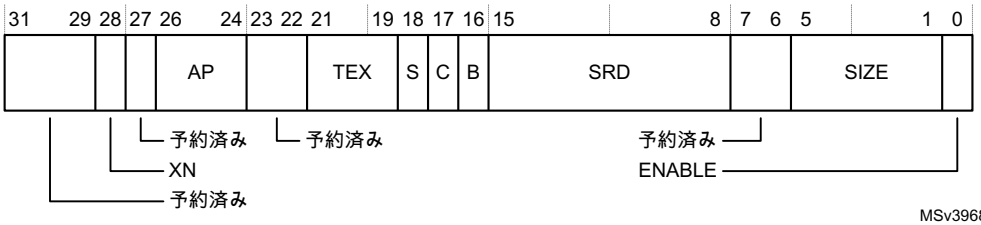



表 89. MPU_RASR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:29] | - | 予約済み。 |
| [28] | XN | 命令アクセス·ディセーブル·ビット<br>0：命令フェッチは有効。<br>1：命令フェッチは無効 |
| [27] | - | 予約済み。 |
| [26:24] | AP | アクセス許可フィールド、226 ページの表 93 を参照してください。 |
| [23:22] | - | 予約済み。 |
| [21:19, 17, 16] | TEX、C、B | メモリアクセス属性。225 ページの表 91 を参照してください。 |
| [18] | S | 共有可能ビット。225 ページの表 91 を参照してください。 |
| [15:8] | SRD | サブ領域ディセーブル·ビット。このフィールドの各ビットは次を示します。<br>0：対応するサブ領域が有効。<br>1：対応するサブ領域が無効。<br>詳細については、228 ページのサブ領域を参照してください。<br>サイズが 128 バイト以下の領域はサブ領域をサポートしません。そのような領域の属性を書き込む場合は、SRD フィールドに 0x00 を書き込みます。 |
| [7:6] | - | 予約済み。 |
| [5:1] | SIZE | MPU 保護領域のサイズを指定。最小許容値は 4 （0b00100）です。詳細については、SIZE フィールドの値を参照してください。 |
| [0] | ENABLE | 領域イネーブル·ビット。 |

アクセス許可については、225 ページのMPU アクセス許可属性を参照してください。

## SIZE フィールドの値

SIZE フィールドは、RNR で指定されている MPU メモリ領域のサイズを次のように定義します。

（領域サイズ（バイト単位））= $2^{(SIZE + 1)}$

最小許容領域サイズは 32 バイト（SIZE 値 4 に相当）です。表 90 に、SIZE 値の例、および対応する領域サイズと MPU_RBAR の N 値を示します。

表 90. SIZE フィールドの値の例

| SIZE 値 | 領域サイズ | N 値(1) | 注 |
|---|---|---|---|
| 0b00100 (4) | 32 バイト | 5 | 最小許容サイズ |
| 0b01001 (9) | 1 KB | 10 | - |

表 90. SIZE フィールドの値の例 （続き）

| SIZE 値 | 領域サイズ | N 値(1) | 注 |
|---|---|---|---|
| 0b10011 (19) | 1 MB | 20 | - |
| 0b11101 (29) | 1 GB | 30 | - |
| 0b11111 (31) | 4 GB | 32 | 最大許容サイズ |

1. MPU_RBAR 内（222 ページのMPU 領域ベース･アドレス･レジスタを参照）。

### 4.6.6 MPU アクセス許可属性

このセクションでは、MPU アクセス許可属性について説明します。RASRの TEX、C、B、S、AP、および XN の各アクセス許可ビットは、対応するメモリ領域へのアクセスを制御します。必要な許可を持たないメモリ領域へのアクセスが行われた場合､MPU は許可フォールトを生成します｡表 91 に、TEX、C、B、および S の各アクセス許可ビットのエンコードを示します。

表 91. TEX、C、B、S のエンコード

| TEX | C | B | S | メモリタイプ | 共有可能性 | その他の属性 |
|---|---|---|---|---|---|---|
| 0b000 | 0 | 0 | x(1) | Strongly-ordered | 共有可能 | - |
| | | 1 | x (1) | デバイス | 共有可能 | - |
| | 1 | 0 | 0 | ノーマル | 共有不可 | 外部および内部ライトスルー書込み割当てなし |
| | | | 1 | | 共有可能 | |
| | | 1 | 0 | ノーマル | 共有不可 | 外部および内部ライトバック書込み割当てなし |
| | | | 1 | | 共有可能 | |
| 0b001 | 0 | 0 | 0 | ノーマル | 共有不可 | 外部および内部キャッシュ不可 |
| | | | 1 | | 共有可能 | |
| | | 1 | x (1) | 予約済みエンコード | | - |
| | 1 | 0 | x (1) | 実装定義属性 | | - |
| | | 1 | 0 | ノーマル | 共有不可 | 外部および内部ライトバック書込みおよび読出し割当て |
| | | | 1 | | 共有可能 | |
| 0b010 | 0 | 0 | x (1) | デバイス | 共有不可 | 共有不可デバイス |
| | | 1 | x (1) | 予約済みエンコード | | - |
| | 1 | x (1) | x (1) | 予約済みエンコード | | - |
| 0b1BB | A | A | 0 | ノーマル | 共有不可 | キャッシュ可能メモリ､BB = 外部ポリシー､AA = 内部ポリシー｡AA ビットと BB ビットのエンコードについては、226 ページの表 92 を参照してください。 |
| | | | 1 | | 共有可能 | |

1. MPU はこのビットの値を無視します。

表 92 に、TEX 値が 4 ~ 7 の範囲のメモリ属性エンコードのキャッシュ･ポリシーを示します。

表 92. メモリ属性エンコードのキャッシュ·ポリシー

| エンコード、AA または BB | 対応するキャッシュ·ポリシー |
|---|---|
| 00 | キャッシュ格納不可 |
| 01 | ライトバック、書込みおよび読出し割当て |
| 10 | ライトスルー、書込み割当てなし |
| 11 | ライトバック、書込み割当てなし |

表 93 に、特権ソフトウェアと非特権ソフトウェアのアクセス許可を定義する AP エンコードを示します。

表 93. AP エンコード

| AP[2:0] | 特権<br>アクセス許可 | 非特権<br>アクセス許可 | 説明 |
|---|---|---|---|
| 000 | アクセスなし | アクセスなし | アクセスはすべて許可フォールトを生成 |
| 001 | RW | アクセスなし | 特権ソフトウェアからのアクセスのみ |
| 010 | RW | RO | 非特権ソフトウェアからの書込みは許可フォールトを生成 |
| 011 | RW | RW | フル·アクセス |
| 100 | 予測不能 | 予測不能 | 予約済みです。 |
| 101 | RO | アクセスなし | 特権ソフトウェアによる読出しのみ |
| 110 | RO | RO | 特権ソフトウェアまたは非特権ソフトウェアによる読出し専用 |
| 111 | RO | RO | 特権ソフトウェアまたは非特権ソフトウェアによる読出し専用 |

### 4.6.7 MPU の不一致

アクセスが MPU 許可に違反する場合、プロセッサは、メモリ管理フォールトを生成します（28 ページの例外と割込みを参照）。MMFSR は、フォールトの原因を示します。詳細については、203 ページのメモリ管理フォールト·ステータス·レジスタを参照してください。

### 4.6.8 MPU 領域の更新

MPU 領域の属性を更新するには、MPU_RNR、MPU_RBAR、および MPU_RASR の各レジスタを更新します。各レジスタを個別にプログラミングするか、またはマルチワード書込みを使用してこれらのレジスタをすべてプログラムすることができます。MPU_RBAR および MPU_RASR エイリアスを使用すると、STM 命令で最大 4 つの領域を同時にプログラムすることが可能です。

#### 個別ワードによる MPU 領域の更新

1 つの領域を設定する簡単なコード：

```
; R1 = 領域番号
; R2 = サイズ／有効化
; R3 = 属性
; R4 = アドレス
LDR R0,=MPU_RNR          ; 0xE000ED98, MPU 領域番号レジスタ
STR R1, [R0, #0x0]       ; 領域番号
STR R4, [R0, #0x4]       ; 領域ベース·アドレス
STRH R2, [R0, #0x8]      ; 領域のサイズと有効化
```

```
STRH R3, [R0, #0xA]      ; 領域属性
```

変更している領域が既に有効になっている場合、新しい領域設定を MPU に書き込む前に領域を無効にします。例：

```
; R1 = 領域番号
; R2 = サイズ／有効化
; R3 = 属性
; R4 = アドレス
LDR R0,=MPU_RNR           ; 0xE000ED98、MPU 領域番号レジスタ
STR R1, [R0, #0x0]        ; 領域番号
BIC R2, R2, #1            ; 無効化
STRH R2, [R0, #0x8]       ; 領域のサイズと有効化
STR R4, [R0, #0x4]        ; 領域ベース･アドレス
STRH R3, [R0, #0xA]       ; 領域属性
ORR R2, #1                ; 有効化
STRH R2, [R0, #0x8]       ; 領域のサイズと有効化
```

ソフトウェアは、次のようにメモリ･バリア命令を使用する必要があります。

- バッファ書込みなど、MPU 設定の変更によって影響を受ける可能性がある未処理のメモリ転送が存在する場合は、MPU セットアップの前。
- 新しい MPU 設定を使用する必要があるメモリ転送を含む場合は、MPU セットアップの後

MPU セットアップの際は、ソフトウェアは、Strongly-ordered メモリ領域である PPB 経由で MPU にアクセスするため、メモリ･バリア命令を実行する必要はありません。

たとえば、プログラミング･シーケンスの直後にすべてのメモリ･アクセス動作を実行することが要求されている場合、DSB 命令と ISB 命令を使用します。DSB は、コンテキスト切替えの終了時など、MPU 設定の変更後に必要です。ISB は、分岐または呼出しによって 1 つまたは複数の MPU 領域をプログラムするコードに入る場合に必要です。例外を受け取ることによってプログラミングシーケンスに入り、例外からの復帰を使用してプログラミングシーケンスから抜ける場合、ISB 命令は不要です。

### マルチワード書込みによる MPU 領域の更新

情報の分割方法によっては、マルチワード書込みを使用して直接プログラムできます。次に示す再プログラミングについて考えてみましょう。

```
; R1 = 領域番号
; R2 = アドレス
; R3 = サイズ、属性を 1 つに
LDR R0, =MPU_RNR     ; 0xE000ED98、MPU 領域番号レジスタ
STR R1, [R0, #0x0]   ; 領域番号
STR R2, [R0, #0x4]   ; 領域ベース･アドレス
STR R3, [R0, #0x8]   ; 領域属性、サイズ、および有効化
```

これを STM 命令を使用して最適化します。

```
; R1 = 領域番号
; R2 = アドレス
; R3 = サイズ、属性を 1 つに
LDR R0, =MPU_RNR     ; 0xE000ED98、MPU 領域番号レジスタ
STM R0, {R1-R3}      ; 領域番号、アドレス、属性、サイズ、および有効化
```

事前にパックされた情報の場合、これを 2 ワードで実行できます。これは、MPU_RBAR に必要な領域番号が含まれ、その VALID ビットが 1 にセットされていることを意味します（222 ページのMPU 領域ベース･アドレス･レジスタを参照）。ブート･ローダなど、静的にパックされたデータには以下を使用します。

```
; R1 = アドレスと領域番号を 1 つに
; R2 = サイズと属性を 1 つに
LDR R0, =MPU_RBAR   ; 0xE000ED9C、MPU 領域ベースレジスタ。
STR R1, [R0, #0x0]  ; 領域ベース・アドレスと
                    ; VALID （ビット 4）を結合した領域番号を 1 にセットします。
STR R2, [R0, #0x4]  ; 領域属性、サイズ、および有効化。
```

### サブ領域

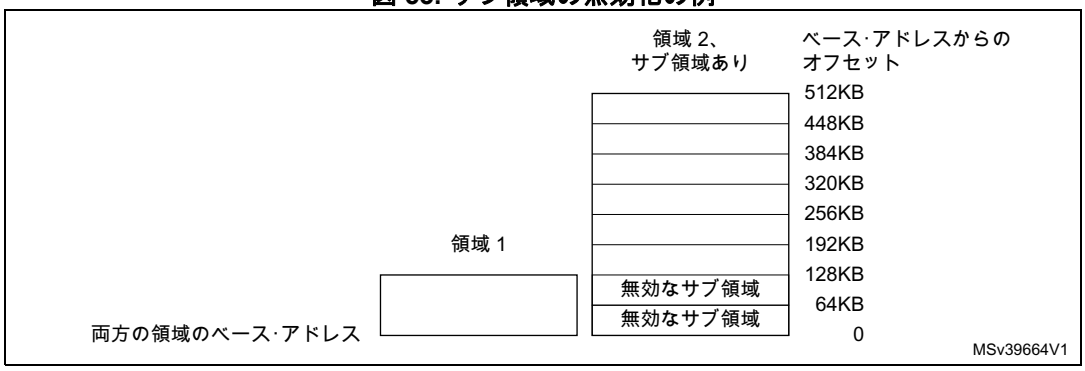
256 バイト以上の領域は、8 つの等サイズのサブ領域に分割されます。MPU_RASR の SRD フィールドで対応するビットをセットして、サブ領域を無効にします（223 ページのMPU 領域属性およびサイズ・レジスタを参照）。SRD の最下位ビットが最初のサブ領域を制御し、最上位ビットが最後のサブ領域を制御します。あるサブ領域を無効にすると、その無効な範囲とオーバーラップするもう 1 つの領域が代わりに一致をとります。無効なサブ領域にオーバーラップする他の有効領域がなく、アクセスが非特権であるか、バックグラウンド領域が無効になっている場合、MPU はフォールトを発行します。

32、64、または 128 バイトの領域はサブ領域をサポートしません。これらのサイズの領域を使用する場合、SRD フィールドを 0x00 にセットする必要があります。そうしないと、MPU の動作は予測不能になります。

### SRD の使用例

同じベース・アドレスを持つ 2 つの領域はオーバーラップします。領域 1 は 128KB、領域 2 は 512KB です。領域 1 の属性が先頭の 128KB の領域に確実に適用されるようにするため、領域 2 の SRD フィールドを 0b00000011 にセットして、次の図に示すように先頭の 2 つのサブ領域を無効にします。

**図 53. サブ領域の無効化の例**




## 4.6.9 MPU 設計のヒントとコツ

予測不能な動作を避けるため、割込みハンドラがアクセスする可能性がある領域の属性を更新する前に、割込みを無効にします。

プロセッサは、MPU レジスタへのアンアラインド・アクセスをサポートしません。

MPU レジスタは、ワードでのアラインド・アクセスのみをサポートします。バイトおよびハーフワード・アクセスは予測できません。

MPU をセットアップする際、MPU がすでにプログラムされている場合は、未使用領域を無効にして、以前の領域設定が新しい MPU セットアップによって影響を及ぼさないようにします。

## 4.7 浮動小数点ユニット

Cortex®-M7 *Floating-Point Unit*（浮動小数点ユニット）（FPU）は FPv5 浮動小数点拡張機能を実装します。

FPU は、単精度および倍精度の加算、減算、乗算、除算、積和演算、平方根演算を完全にサポートします。また、固定小数点データ形式と浮動小数点データ形式の間の変換、および浮動小数点の定数命令を提供します。

FPU は、IEEE 754 規格と呼ばれる *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic* に準拠した浮動小数点計算機能を提供します。

実装が単精度 FPU のみをサポートする場合、シリコンベンダーは次のテキストも含める必要があります。FPU には 32 個の単精度拡張レジスタが含まれますが、これには、ロード、ストア、および転送の各演算で 16 個のダブルワード･レジスタとしてアクセスすることもできます。

表 94 に、FPU つきの Cortex®-M7 プロセッサの浮動小数点システム･レジスタを示します。

**表 94. Cortex®-M7 浮動小数点システム･レジスタ**

| アドレス | 名前 | タイプ | 必要特権 | リセット | 説明 |
|---|---|---|---|---|---|
| 0xE000ED88 | CPACR | RW | 特権 | 0x00000000 | 230 ページのコプロセッサ･アクセス制御レジスタ |
| 0xE000EF34 | FPCCR | RW | 特権 | 0xC0000000 | 230 ページの浮動小数点コンテキスト制御レジスタ |
| 0xE000EF38 | FPCAR | RW | 特権 | - | 232 ページの浮動小数点コンテキスト・アドレス・レジスタ |
| - | FPSCR(1) | RW | 非特権 | - | 232 ページの浮動小数点ステータス制御レジスタ |
| 0xE000EF3C | FPDSCR | RW | 特権 | 0x00000000 | 234 ページの浮動小数点デフォルト･ステータス制御レジスタ |

1. FPSCR レジスタはメモリマップされません。VMSR と VMRS 命令を使ってアクセスが可能です。162 ページのVMRS と 163 ページのVMSR を参照してください。ソフトウェアは、FPU が有効な場合 FPSCR のみにアクセスできます。234 ページのFPU の有効化を参照してください。

以降のセクションでは、このプロセッサに固有の実装である浮動小数点システム･レジスタについて説明します。

### 4.7.1 コプロセッサ･アクセス制御レジスタ

CPACR レジスタは、コプロセッサのアクセス特権を指定します。属性については、229 ページの表 94 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 54. CPACR のビット割当て**

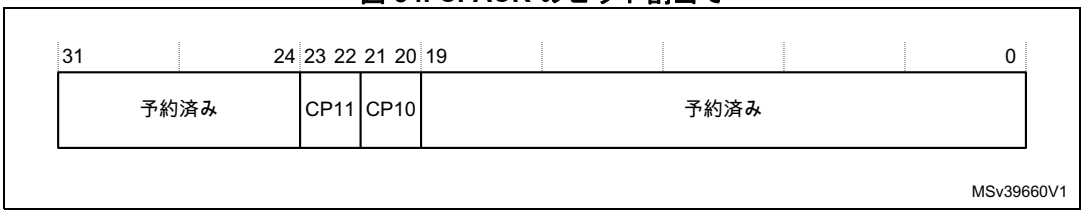



**表 95. CPACR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:24] | - | 予約済み。0 として読み出され、書込みは無視します。 |
| 値が 10 および 11 の *n* に対し、[2*n*+1:2*n*] | CP*n* | コプロセッサ *n* のアクセス特権を示します。各フィールドの取りうる値を次に示します。<br>0b00：アクセス拒否。アクセスを試みると、NOCP 用法フォールトが生成されます。<br>0b01：特権アクセスのみ。非特権アクセスを試みると、NOCP フォールトが生成されます。<br>0b10：予約済み。アクセスの結果は予測不能です。<br>0b11：フル･アクセス。 |
| [19:0] | - | 予約済み。0 として読み出され、書込みは無視します。 |

### 4.7.2 浮動小数点コンテキスト制御レジスタ

FPCCR レジスタは、FPU 制御データをセットまたは返します。属性については、229 ページの表 94 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 55. FPCCR のビット割当て**

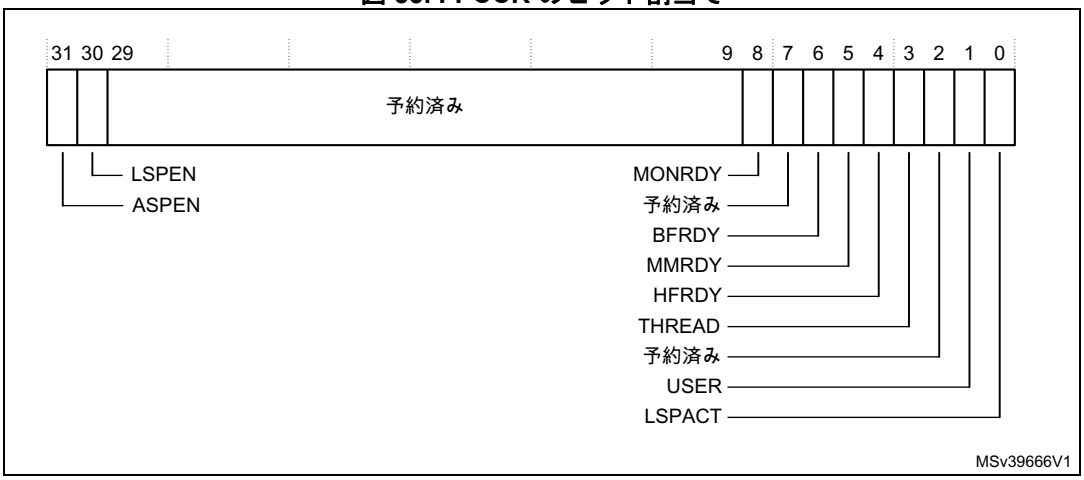


表 96. FPCCR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31] | ASPEN | 浮動小数点命令の実行において CONTROL.FPCA 設定を有効にします。これにより、例外開始時と例外終了時に、浮動小数点コンテキストについて、ハードウェア状態の保持と復元が自動的に行われます。<br>0：浮動小数点命令の実行において CONTROL.FPCA 設定を無効にします。<br>1：浮動小数点命令の実行において CONTROL.FPCA 設定を有効にします。 |
| [30] | LSPEN | 0：浮動小数点コンテキストのレイジーな自動状態保持を無効にします。<br>1：浮動小数点コンテキストのレイジーな自動状態保持を有効にします。 |
| [29:9] | - | 予約済み。 |
| [8] | MONRDY | 0：デバッグモニタが無効であるか、または浮動小数点スタック･フレームの割当て時に優先度により MON_PEND の設定が許可されませんでした。<br>1：デバッグモニタが有効であり、浮動小数点スタック･フレームの割当て時に優先度により MON_PEND の設定が許可されています。 |
| [7] | - | 予約済み。 |
| [6] | BFRDY | 0：バス･フォールトが無効であるか、または浮動小数点スタック･フレームの割当て時に優先度によりバス･フォールト･ハンドラを保留状態に設定することが許可されませんでした。<br>1：バス･フォールトが有効であり、浮動小数点スタック･フレームの割当て時に優先度によりバス･フォールト･ハンドラを保留状態に設定することが許可されました。 |
| [5] | MMRDY | 0：メモリ管理が無効であるか、または浮動小数点スタック･フレームの割当て時に優先度によりメモリ管理ハンドラを保留状態に設定することが許可されませんでした。<br>1：メモリ管理が有効であり、浮動小数点スタック･フレームの割当て時に優先度によりメモリ管理ハンドラを保留状態に設定することが許可されました。 |
| [4] | HFRDY | 0：浮動小数点スタック･フレームの割当て時に優先度によりハード･フォールト･ハンドラを保留状態に設定することが許可されませんでした。<br>1：浮動小数点スタック･フレームの割当て時に優先度によりハード･フォールト･ハンドラを保留状態に設定することが許可されました。 |
| [3] | THREAD | 0：浮動小数点スタック･フレームの割当て時にモードがスレッド･モードではありませんでした。<br>1：浮動小数点スタック･フレームの割当て時にモードがスレッド･モードでした。 |
| [2] | - | 予約済み。 |
| [1] | USER | 0：浮動小数点スタック･フレームの割当て時に特権レベルがユーザではありませんでした。<br>1：浮動小数点スタック･フレームの割当て時に特権レベルがユーザでした。 |
| [0] | LSPACT | 0：レイジーな状態保持はアクティブではありません。<br>1：レイジーな状態保持はアクティブです。浮動小数点スタック・フレームは割り当てられましたが、状態の保存が延期されました。 |

### 4.7.3 浮動小数点コンテキスト・アドレス・レジスタ

FPCAR レジスタは、例外スタック·フレームに割り当てられている未入力の浮動小数点レジスタ空間の位置を保持しています。属性については、229 ページの表 94 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 56. FPCAR のビット割当て**

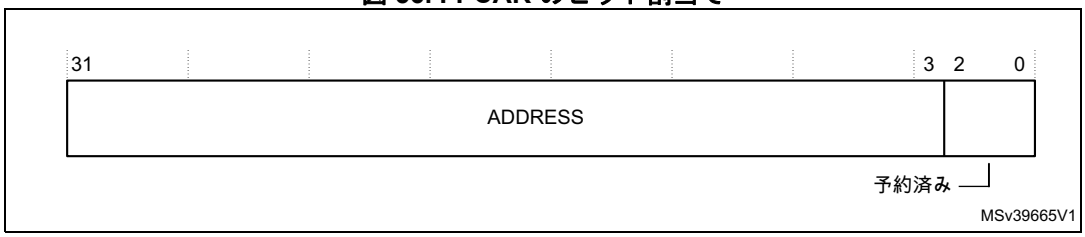



**表 97. FPCAR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:3] | ADDRESS | 例外スタック·フレームに割り当てられている未入力の浮動小数点レジスタ空間の位置。 |
| [2:0] | - | 予約済み。0 として読み出され、書込みは無視します。 |

### 4.7.4 浮動小数点ステータス制御レジスタ

FPSCR レジスタは、浮動小数点システムの必要なすべてのユーザ·レベル制御を提供します。ビット割当てを次に示します。

**図 57. FPSCR のビット割当て**

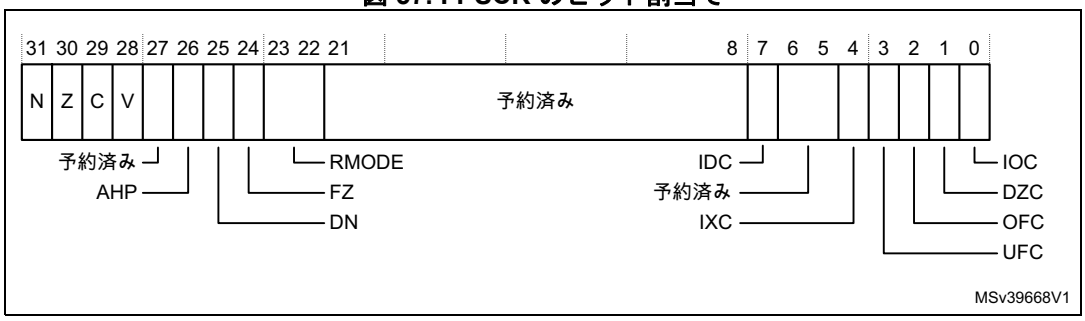



**表 98. FPSCR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31] | N | 条件コードに関するフラグ。このフラグは、浮動小数点比較演算によって更新されます。<br>N：ネガティブ条件コード·フラグ。<br>Z：ゼロ条件コード·フラグ。<br>C：キャリー条件コード·フラグ。<br>V：オーバーフロー条件コード·フラグ。 |
| [30] | Z | |
| [29] | C | |
| [28] | V | |
| [27] | - | 予約済み。 |
| [26] | AHP | オルタネィティブ半精度制御ビット。<br>0：IEEE 半精度形式が選択されています。<br>1：オルタネィティブ半精度形式が選択されています。 |

表 98. FPSCR ビット割当て （続き）

| ビット | 名前 | 機能 |
|---|---|---|
| [25] | DN | デフォルトの NaN モード制御ビット。<br>0：NaN オペランドは浮動小数点演算の出力まで伝播します。<br>1：1 つまたは複数の NaN を含む演算はすべて、デフォルト NaN を返します。 |
| [24] | FZ | Flush-to-zero モード制御ビット。<br>0：Flush-to-zero モードは無効です。浮動小数点システムの動作は、IEEE 754 規格に完全に準拠しています。<br>1：Flush-to-zero モードは有効です。 |
| [23:22] | RMODE | 丸めモード制御フィールド。このフィールドのエンコードは次のとおりです。<br>0b00：近似値への丸め（RN）モード<br>0b01：プラス無限大への丸め（RP）モード<br>0b10：マイナス無限大への丸め（RM）モード<br>0b11：0 への丸め（RZ）モード<br>指定された丸めモードはほとんどすべての浮動小数点命令で使用されます。 |
| [21:8] | - | 予約済み。 |
| [7] | IDC | 入力非正規化累積例外ビット。ビット [4:0] を参照してください。 |
| [6:5] | - | 予約済み。 |
| [4] | IXC | 浮動小数点例外の累積例外ビット。ビット [7] も参照してください。これらの各ビットは 1 にセットされ、最後に 0 が書き込まれてから対応する例外が発生したことを示します。<br>IDC、ビット [7]：入力非正規化累積例外ビット。<br>IXC：不正確累積例外ビット。<br>UFC：アンダーフロー累積例外ビット。<br>OFC：オーバーフロー累積例外ビット。<br>DZC：0 による除算累積例外ビット。<br>IOC：無効演算累積例外ビット。 |
| [3] | UFC | |
| [2] | OFC | |
| [1] | DZC | |
| [0] | IOC | |

### 4.7.5 浮動小数点デフォルト・ステータス制御レジスタ

FPDSCR レジスタは、浮動小数点ステータス制御データのデフォルト値を保持します。属性については、229 ページの表 94 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 58. FPDSCR のビット割当て**

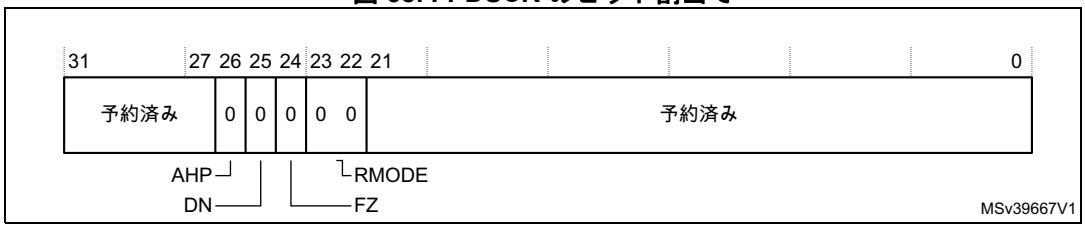



**表 99. FPDSCR ビット割当て**

| ビット | 名前 | 機能 |
|---|---|---|
| [31:27] | - | 予約済みです。 |
| [26] | AHP | FPSCR.AHP のデフォルト値 |
| [25] | DN | FPSCR.DN のデフォルト値 |
| [24] | FZ | FPSCR.FZ のデフォルト値 |
| [23:22] | RMODE | FPSCR.RMode のデフォルト値 |
| [21:0] | - | 予約済みです。 |

### 4.7.6 FPU の有効化

FPU は、リセットにより無効になります。浮動小数点命令を使用する前に有効にする必要があります。例 4-1：FPU の有効化に、特権モードで FPU を有効にするためのコード・シーケンスの例を示します。プロセッサが CPACR に対して読み書きするには、特権モードである必要があります。

**例 4-1：FPU の有効化**

```
CPACR   EQU  0xE000ED88
LDR     R0,  =CPACR          ; CPACR の読出し
LDR     r1, [R0]             ; ビット 20 ~ 23 を設定して
                             ; CP10 と CP11 コプロセッサを有効にします。
ORR     R1, R1, #(0xF << 20)
STR     R1, [R0]             ; CPACR に修正値をライト・バックします。
DSB
ISB                          ; FPU が有効になっているので、
                             ; パイプラインをリセットします。
```

### 4.7.7 FPU 例外割込みの有効化とクリア

FPU 例外フラグは、割込みコントローラ経由で割込みを生成します。FPU 割込みは、割込みコントローラ経由でグローバルに制御されます。

個別のマスクは存在しません。FPU 割込みの有効化／無効化は割込みコントローラ･レベルで実行されます。IXC 例外フラグは発生頻度が非常に高いため、割込みコントローラには接続されておらず、割込みを生成できません。必要な場合は、ポーリングによって管理する必要があります。

FPU 例外フラグのクリアは、FPU コンテキストの保存／復元設定に依存します。

- 浮動小数点レジスタを保存しない：浮動小数点コンテキスト制御レジスタ（FPCCR）のビット 30 の LSPEN=0、およびビット 31 の ASPEN=0 の場合
  浮動小数点ステータス制御レジスタ（FPSCR）で割込みソースをクリアする必要があります。
  例：

```
register uint32_t fpscr_val = 0;
fpscr_val  = __get_FPSCR();
{ check exception flags }
fpscr_val &= (uint32_t)~0x8F; // Clear all exception flags
__set_FPSCR(fpscr_val);
```

- レイジーな保存／復元：浮動小数点コンテキスト制御レジスタ（FPCCR）のビット 30 の LSPEN=1、およびビット 31 の ASPEN=X の場合。
  浮動小数点コンテキストのレイジーな保存/復元では、浮動小数点ステータス制御レジスタ（FPSCR）をダミー読出ししてステータスを固定した後に読み出し、そして FPSCR をクリアします。
  その後で、FPSCR をスタック処理します。
  例：

```
register uint32_t fpscr_val = 0;
register uint32_t reg_val = 0;
reg_val =  __get_FPSCR();        // dummy access
fpscr_val=*(__IO uint32_t*)(FPU->FPCAR +0x40);
{ check exception flags }
fpscr_val &= (uint32_t)~0x8F ;   // Clear all exception flags
*(__IO uint32_t*)(FPU->FPCAR +0x40)=fpscr_val;
__DMB() ;
```

- 浮動小数点レジスタを自動的に保存／復元：浮動小数点コンテキスト制御レジスタ（FPCCR）のビット 30 の LSPEN=0、およびビット 31 の ASPEN=1 の場合。
  浮動小数点コンテキストの自動的な保存/復元では、浮動小数点ステータス制御レジスタ（FPSCR）を読み出して、レジスタをクリアします。
  その後で、FPSCR をスタック処理します。
  例：

```
// FPU Exception handler
void FPU_ExceptionHandler(uint32_t lr, uint32_t sp)
{
register uint32_t fpscr_val;
   if(lr == 0xFFFFFFE9)
   {
     sp = sp + 0x60;
   }
   else if(lr == 0xFFFFFFED)
   {
```

```
        sp = __get_PSP() + 0x60 ;
      }
    fpscr_val = *(uint32_t*)sp;
    { check exception flags }
    fpscr_val &= (uint32_t)~0x8F ;   // Clear all exception flags
    *(uint32_t*)sp = fpscr_val;
    __DMB() ;
    }
    // FPU IRQ Handler
    void __asm FPU_IRQHandler(void)
    {
    IMPORT  FPU_ExceptionHandler
    MOV   R0, LR             // move LR to R0
    MOV   R1, SP             // Save SP to R1 to avoid any modification to
the stack pointer from FPU_ExceptionHandler
                             //
    VMRS R2, FPSCR           // dummy read access, to force clear
    B     FPU_ExceptionHandler
    BX    LR
    }
```

## 4.8 キャッシュのメンテナンス操作

キャッシュのメンテナンス操作には、特権ロードおよび特権ストアによってのみアクセスできます。これらのレジスタに非特権アクセスを行うと、常にバス・フォールトが生成されます。

**表 100. キャッシュ・メンテナンス空間レジスタの概要**

| アドレス | 名前 | タイプ | 必要特権 | リセット値 | 説明 |
|---|---|---|---|---|---|
| 0xE000EF50 | ICIALLU | WO | 特権 | 不明 | *Point of Unification*（統合のポイント）（PoU）の命令キャッシュをすべて無効化します。(1) |
| 0xE000EF54 | - | - | - | - | 予約済みです。 |
| 0xE000EF58 | ICIMVAU | WO | 特権 | 不明 | 命令キャッシュは PoU へのアドレスにより無効化 (1) |
| 0xE000EF5C | DCIMVAC | WO | 特権 | 不明 | データ・キャッシュは *Point of Coherency*（コヒーレンシ点）（PoC）へのアドレスにより無効化(2) |
| 0xE000EF60 | DCISW | WO | 特権 | 不明 | データ・キャッシュはセット／ウェイによって無効化されます。 |
| 0xE000EF64 | DCCMVAU | WO | 特権 | 不明 | PoU へのアドレスによるデータ・キャッシュ (1) |
| 0xE000EF68 | DCCMVAC | WO | 特権 | 不明 | PoC へのアドレスによるデータ・キャッシュの消去 (2) |
| 0xE000EF6C | DCCSW | WO | 特権 | 不明 | セット／ウェイによるデータ・キャッシュの消去 |
| 0xE000EF70 | DCCIMVAC | WO | 特権 | 不明 | PoC へのアドレスによるデータ・キャッシュの消去と無効化 (2) |
| 0xE000EF74 | DCCISW | WO | 特権 | 不明 | セット／ウェイによるデータ・キャッシュの消去と無効化 |
| 0xE000EF78 | BPIALL | RAZ/WI | 特権 | - | BPIALL レジスタは実装されません。 |

1. PoU によるキャッシュ・メンテナンス操作は、Cortex®-M7 データと命令キャッシュの間でデータを同期させるために使用できます（例えばソフトウェアが自己修正コードを使用する場合）。
2. PoC によるキャッシュ・メンテナンス操作は、Cortex®-M7 データ・キャッシュとシステム DMA などの外部エージェント間でのデータ同期に使用できます。

### 4.8.1 フル命令キャッシュ操作

ICIALLU は WO であり、書込みデータは無視され、読出しは 0 を返します。このレジスタに書き込むと、要求されたキャッシュのメンテナンス操作が実行されます。分岐予測器のメンテナンスが必要ないため、BPIALL レジスタは Cortex-M7 プロセッサに実装されません。レジスタは RAZ/WI です。

### 4.8.2 アドレスによる命令およびデータ・キャッシュ操作

キャッシュのメンテナンス操作レジスタは、ICIMVAU、DCIMVAC、DCCMVAU、DCCMVAC、および DCCIMVAC です。これらのレジスタは WO で、読出しは 0 を返します。属性については、表 100 のレジスタ概要を参照してください。ビット割当てを次に示します。

**表 101. キャッシュ操作レジスタのビット割当て**

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [31:0] | MVA | WO | リクエストされた操作の MVA |

### 4.8.3 セットウェイによるデータ・キャッシュ操作

DCISW、DCCSW、および DCCISW レジスタは WO です。このビットを読み出すと 0 が返されます。属性については、236 ページの表 100 のレジスタ概要を参照してください。ビット割当てを次に示します。

**図 59. キャッシュ操作のビット割当て**

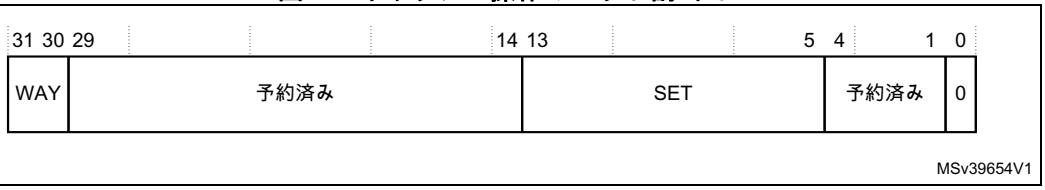



**表 102. セットウェイ・ビット割当てによるキャッシュ操作**

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [31:30] | WAY | WO | 操作が適用されるウェイ。<br>データ・キャッシュの場合、値 0、1、2、および 3 がサポートされます。 |
| [29:14] | - | - | 予約済みです。 |
| [13:5] | SET | WO | 操作が適用されるセット／インデックス。キャッシュ内のインデックス数は、設定されたキャッシュ・サイズによって異なります。これが最大値より小さい場合は、このフィールドの LSB を使用します。キャッシュ内のセット数は、216 ページのキャッシュ・サイズ ID レジスタを読みだすことで決定することができます。 |
| [4:1] | - | - | 予約済みです。 |
| [0] | - | - | 常に 0 が読み出されます。 |

### 4.8.4 CMSIS を使用した Cortex®-M7 キャッシュのメンテナンス操作

CMSIS 関数により、さまざまな Cortex®-M プロファイル・プロセッサ間でソフトウェアを移植できます。CMSIS の使用時にキャッシュのメンテナンス操作にアクセスするには、次の関数を使用します。

**表 103. CMSIS アクセス・キャッシュのメンテナンス操作**

| CMSIS 関数 | 説明 |
|---|---|
| `void SCB_EnableICache(void)` | 命令キャッシュを無効化してから有効にします。 |
| `void SCB_DisableICache(void)` | 命令キャッシュを無効にし、その内容を無効にします。 |
| `void SCB_InvalidateICache(void)` | 命令キャッシュの無効化 |
| `void SCB_EnableDCache(void)` | データ・キャッシュを無効化してから有効化します。 |
| `void SCB_DisableDCache(void)` | データ・キャッシュを無効にし、その内容を消去して無効にします。 |
| `void SCB_InvalidateDCache(void)` | データ・キャッシュの無効化 |
| `void SCB_CleanDCache(void)` | データ・キャッシュの消去 |
| `void SCB_CleanInvlaidateDCache(void)` | データ・キャッシュの消去と無効化 |

### 4.8.5 L1 キャッシュの初期化と有効化

ユーザは、キャッシュのメンテナンス操作を使用して、次のことができます。

- キャッシュ起動タイプの操作。
- 他のバス・マスタから共有データが見えるようにキャッシュを操作すること。
- 外部 DMA エージェントによって変更されたデータを有効化し、Cortex®-M7 プロセッサから認識できるようにします。

命令キャッシュを有効または無効にした後、パイプラインを一掃するため ISB 命令を発行する必要があります。これにより、後続のすべての命令フェッチで、命令キャッシュの有効化または無効化の効果を確認することができます。

リセット後、各キャッシュを有効にする前に無効にする必要があります。

データ・キャッシュを無効化する場合、キャッシュ全体をクリアして、ダーティ・データが外部メモリに確実に一掃されるようにする必要があります。

キャッシュが無効化された後に外部メモリが変更された可能性がある場合、データ・キャッシュを有効にする前に、データ・キャッシュ全体を無効化する必要があります。

キャッシュが無効化された後に外部メモリが変更された可能性がある場合、命令キャッシュを有効にする前に、命令キャッシュ全体を無効化する必要があります。

L1 データおよび命令キャッシュは、ソフトウェアで有効化する前に無効化する必要があります。そうしないと、予測できない動作が発生する可能性があります。

#### データ・キャッシュ全体を無効化します。

ソフトウェアは、次のコード例を使用して、データ・キャッシュ全体を無効化できます（プロセッサに含まれている場合）。この操作は、キャッシュの各ラインにわたって繰り返し、*Private Peripheral Bus*（プライベート・ペリフェラル・バス）（PPB）メモリ領域の DCISW レジスタを使用してラインを無効化します。キャッシュのウェイとセットの数は、CCSIDR レジスタを読み出すことによって決定されます。

```
CCSIDR  EQU 0xE000ED80
CSSELR  EQU 0xE000ED84
DCISW   EQU 0xE000EF60
        MOV r0, #0x0
        LDR r11, =CSSELR
        STR r0, [r11]               ; データ・キャッシュサイズの選択。
        DSB
        LDR r11, =CCSIDR
        LDR r2, [r11]               ; キャッシュサイズの識別
        AND r1, r2, #0x7            ; キャッシュラインのワード数
        ADD r7, r1, #0x4
        MOV r1, #0x3ff
        ANDS r4, r1, r2, LSR #3
        MOV r1, #0x7fff
        ANDS r2, r1, r2, LSR #13
        CLZ r6, r4
        LDR r11, =DCISW
inv_loop1
        MOV r1, r4
inv_loop2
        LSL r3, r1, r6
        LSL r8, r2, r7
        ORRr 3, r3, r8
        STR r3, [r11]               ; D キャッシュラインの無効化
        SUBS r1, r1, #0x1
        BGE inv_loop2
        SUBS r2, r2, #0x1
        BGE inv_loop1
        DSB
        ISB
```

### 命令キャッシュの無効化

次のコード例を使用して、命令キャッシュ全体を無効化できます（プロセッサに含まれている場合）。この操作は、PPB メモリ領域の ICIALLU レジスタに書き込むことによって実行されます。

```
ICIALLU EQU 0xE000EF50
        MOV r0, #0x0
        LDR r11, =ICIALLU
        STR r0, [r11]
        DSB
        ISB
```

### データ・キャッシュおよび命令キャッシュの有効化

次のコード例を使用すると、データおよび命令キャッシュを初期化した後に有効にできます。この操作は、PPB メモリ領域の CCR.IC および CCR.DC フィールドを変更することによって実行されます。

```
CCR     EQU 0xE000ED14
        LDR r11, =CCR
        LDR r0, [r11]
        ORR r0, r0, #0x1:SHL:16     ; CCR.DC フィールドをセット
        ORR r0, r0, #0x1:SHL:17     ; CCR.IC フィールドをセット
        STR r0, [r11]
        DSB
        ISB
```

## 4.8.6 フォールト処理の考慮事項

キャッシュのメンテナンス操作によってバス・フォールトが発生することがあります。このようなフォールト・イベントは非同期です。

このタイプのバス・フォールトは、

- バス・フォールト・ハンドラが有効な場合に、ハード・フォルトに昇格しません。
- ロックアップすることはありません。

フォールト・イベントは非同期であるため、キャッシュのメンテナンス操作のソフトウェア・コードでは、DSB のようなメモリ・バリア命令を完了時に使用して、フォールト・イベントを直ちに監視できるようにする必要があります。

## 4.8.7 キャッシュ・メンテナンスの設計のヒントとコツ

キャッシュのメンテナンス操作の後に、必ず DSB および ISB 命令シーケンスを配置し、ソフトウェアのその後の命令により、効果が確認できるようにしなくてはなりません。

アドレスまたはセット／ウェイによるキャッシュのメンテナンス操作を使用する場合 DSB 命令は、前のロードまたはストアの後、メンテナンス操作の前に実行し、ロードまたはストアの効果が操作によって確認できるようにしなくてはなりません。例えば、DCCMVAC によってアクセスされるアドレスにストアが書き込む場合、DSB 命令によって、ダーティ・データがデータ・キャッシュから正しく一掃されることが保証されます。

1 つ以上のメンテナンス操作が実行されたときに、DSB 命令を使用することで、それらが完了したことと、メンテナンス操作後に続くロードまたはストア操作がすべて順に実行されることが保証されます。

キャッシュのメンテナンス操作は、常に順序どおりに完了します。これは、一連のメンテナンス操作の完了を保証するには、1 つの DSB 命令だけが必要であることを意味します。

次のコード・シーケンスは、キャッシュのメンテナンス操作を使用して、自己修正コードのデータ・キャッシュと命令キャッシュを同期させる方法を示します。このシーケンスは新しい 32 ビット命令を含む <Rx> で入力します。最初の行で 16 ビット命令に STR の代わりに STRH を使用します：

```
STR <Rx>, <inst_address1>
DSB                             ;データがキャッシュに書き込まれたことを
                                ; 確認します。
STR <inst_address1>, DCCMVAU    ; MVA により、point of unification（統合ポイント）
                                ; （PoU）でデータ・キャッシュを消去します。
STR <inst_address1>, ICIMVAU    ; MVA により PoU への命令キャッシュを
                                ; 無効にします。
```

```
DSB                              ;無効化が完了していることを確認します。
ISB                              ;フェッチした命令ストリームを同期します。
```

## 4.9 アクセス制御

L1 キャッシュ ECC の制御、属性のオーバーライド、AHB スレーブ・トラフィックの優先度、およびアクセスが TCM インタフェースと AXI マスタ・インタフェースのどちらにマップされるかは、アクセス制御レジスタで定義されます。アクセス制御レジスタは、次のとおりです。

**表 104. アクセス制御レジスタの概要**

| アドレス | 名前 | タイプ | 必要特権 | リセット値 | 説明 |
|---|---|---|---|---|---|
| 0xE000EF90 | ITCMCR | RW | 特権 | 0x00000000 | 242 ページの命令およびデータ密結合メモリの制御レジスタ |
| 0xE000EF94 | DTCMCR | RW | 特権 | 0x00000000 | |
| 0xE000EF98 | AHBPCR | RW | 特権 | 0x00000000 | 244 ページのAHBP 制御レジスタ |
| 0xE000EF9C | CACR | RW | 特権 | -(1) | 245 ページの補助キャッシュ制御レジスタ |
| 0xE000EFA0 | AHBSCR | RW | 特権 | 0x00000800 | 246 ページのAHB スレーブ制御レジスタ |
| 0xE000EFA8 | ABFSR | RW | 特権 | 0x00000000 | 247 ページの補助バス･フォールト･ステータス･レジスタ |

1. リセット値はインプリメンテーションおよび設定に依存し、シリコン・ベンダーが変更します。キャッシュ ECC が設定されている場合、リセット値は 0x00000000、キャッシュ ECC が設定されていない場合のリセット値は、0x00000002 です。

### 4.9.1 命令およびデータ密結合メモリの制御レジスタ

ITCMCR と DTCMCR は、アクセスを TCM インタフェースにマップするか、AXI マスタ・インタフェースにマップするかを制御します。ビット割当てを次に示します。

**図 60. ITCMR および DTCMR ビットの割当て**

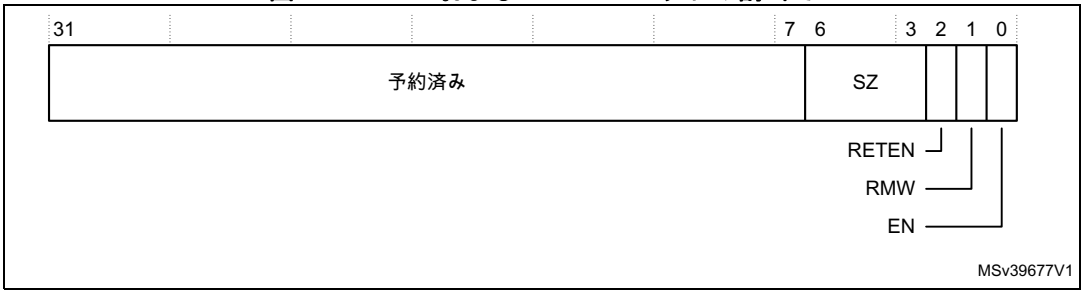



**表 105. ITCMCR および DTCMCR のビット割当て**

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [31:7] | - | - | 予約済み、RAZ/WI |
| [6:3] | SZ | RO | TCM サイズ関連する TCM のサイズを示します。<br>0b0000：TCM は実装されていません。<br>0b0011：4KB<br>0b0100：8KB<br>0b0101：16KB<br>0b0110：32KB<br>0b0111：64KB<br>0b1000：128KB<br>0b1001：256KB<br>0b1010：512KB<br>0b1011：1MB<br>0b1100：2MB<br>0b1101：4MB<br>0b1110：8MB<br>0b1111：16MB<br>その他すべてのエンコーディングは予約済みです。 |
| [2] | RETEN(1) | RW | 再試行フェーズは有効です。有効にすると、プロセッサは、対応する TCM インタフェースでの再試行出力を受け入れることを保証します。<br>0：再試行フェーズは無効です。<br>1：再試行フェーズは有効です。 |
| [1] | RMW(2) | RW | *Read-Modify-Write*（読出し／変更／書込み）（RMW）の有効化与えられた TCM へのすべてのサブチャンクの書込みで RMW シーケンスを使用することを示します。<br>0：RMW は無効です。<br>1：RMW は有効です。 |
| [0] | EN | RW | TCM が有効です。TCM を無効にすると、すべてのアクセスは AXI マスタに行われます。<br>0：TCM は無効です。<br>1：TCM は有効です。 |

1. ITCMCR および DTCMCR の RETEN フィールドは、TCM でのエラー検出／訂正をサポートするために使用されます。
2. ITCMCR および DTCMCR の RMW フィールドは、TCM でのエラー検出／訂正をサポートするために使用されます。

### TCM の有効化

TCM インタフェースは、プロセッサの外部信号によってシステム内のリセット時に有効にできます。リセット時に無効になっている場合、次のコード例を使用して、ソフトウェア内で命令およびデータ TCM インタフェースの両方を有効にすることができます。

```
ITCMCR  EQU 0xE000EF90
DTCMCR  EQU 0xE000EF94

        LDR r11, =ITCMCR
        LDR r0, [r11]
        ORR r0, r0, #0x1    ; ITCMCR.EN フィールドをセット
        STR r0, [r11]

        LDR r11, =DTCMCR
        LDR r0, [r11]
        ORR r0, r0, #0x1    ; DTCMCR.EN フィールドをセット
        STR r0, [r11]

        DSB
        ISB
```

### TCM 再試行および読出し／変更／書込みの有効化

プロセッサに接続された TCM がエラー検出および訂正をサポートする場合、TCM インタフェースを再試行および読出し／変更／書込み機能をサポートするように設定する必要があります。これらは、プロセッサの外部信号によってシステム内のリセット時に有効にできます。これらがリセット時に無効になっている場合、次のコード例を使用して、ソフトウェアでこれらを有効にすることができます。

```
ITCMCR  EQU 0xE000EF90
DTCMCR  EQU 0xE000EF94

        LDR r11, =ITCMCR
        LDR r0, [r11]
        ORR r0, r0, #0x1:SHL:1    ; ITCMCR.RMW フィールドをセット
        ORR r0, r0, #0x1:SHL:2    ; ITCMCR.RETEN フィールドをセット
        STR r0, [r11]

        LDR r11, =DTCMCR
        LDR r0, [r11]
        ORR r0, r0, #0x1:SHL:1    ; DTCMCR.RMW フィールドをセット
        ORR r0, r0, #0x1:SHL:2    ; DTCMCR.RETEN フィールドをセット
        STR r0, [r11]

        DSB
        ISB
```

### 4.9.2 AHBP 制御レジスタ

AHBPCR は、AHBP または AXI マスタ・インタフェース上のデバイスへのアクセスを制御します。ビット割当てを次に示します。

**図 61. AHBPCR のビット割当て**

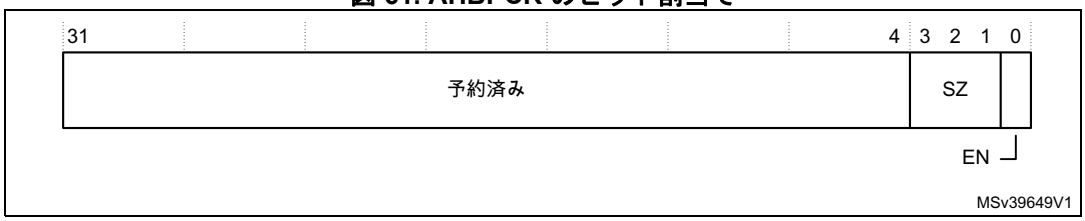



**表 106. AHBPCR ビット割当て**

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [31:4] | - | - | 予約済み、RAZ/WI |
| [3:1] | SZ | RO | AHBP サイズ：<br>0b001：64 MB<br>0b010：128 MB<br>0b011：256 MB<br>0b100：512 MB |
| [0] | EN | RW | AHBP 有効：<br>0：AHBP は無効です。無効にすると、すべてのアクセスは AXI マスタに対して行われます。<br>1：AHBP は有効です。 |

#### AHBP インタフェースの有効化

AHBP インタフェースは、プロセッサの外部信号によってシステム内のリセット時に有効にできます。リセット時に無効になっている場合、次のコード例を使用して、ソフトウェアから AHPB インタフェースを有効にできます。

```
AHBPCR  EQU 0xE000EF98

        LDR r11, =AHBPCR
        LDR r0, [r11]
        ORR r0, r0, #0x1          ; AHBPCR.EN フィールドをセット
        STR r0, [r11]

        DSB
        ISB
```

### 4.9.3 補助キャッシュ制御レジスタ

CACR は、L1 キャッシュ ECC と属性のオーバーライドを制御します。ビット割当てを次に示します。

**図 62. CACR のビット割当て**

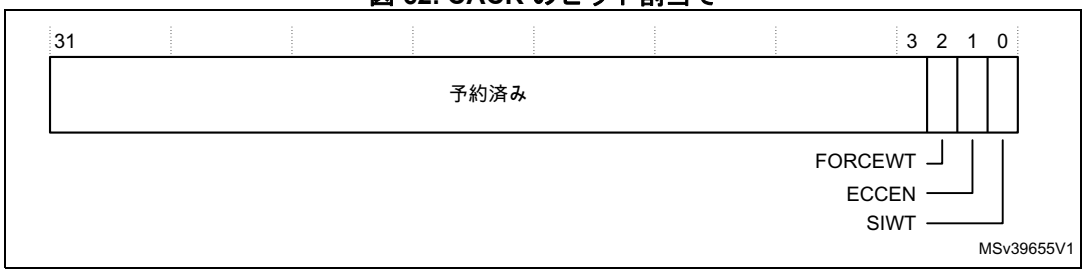



**表 107. CACR ビット割当て**

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [31:3] | - | - | 予約済み、RAZ/WI |
| [2] | FORCEWT | RW | データ・キャッシュでの強制ライトスルーを有効にします。<br>0：強制ライトスルーを無効にします。<br>1：強制ライトスルーを有効にします。キャッシュ可能なライトバック・メモリ領域は、ライトスルーとして処理されます。<br>データ・キャッシュを除く場合、このビットは RAZ です。 |
| [1] | ECCEN | RW | 命令およびデータ・キャッシュの ECC を有効にします。<br>0：命令およびデータ・キャッシュの ECC を有効にします。<br>1：命令およびデータ・キャッシュの ECC を無効にします。<br>データ・キャッシュと命令キャッシュの両方が除外されている場合、または ECC が設定されていない場合、このビットは WI です。ECC がプロセッサに搭載されている場合、ECCEN のリセット値は 0 です。ECC が搭載されていない場合、ECCEN のリセット値は 1 です。 |
| [0] | SIWT | RW | キャッシュのコヒーレンシの使用を有効にします。<br>0：通常キャッシュ可能な共有位置はキャッシュ不可として扱われます。プログラムされた内部キャッシュ可能属性は無視されます。これは、共有メモリのデフォルトの動作モードです。これらの位置でキャッシュはソフトウェアに対してトランスペアレントであるため、コヒーレンシを維持するためのソフトウェアのメンテナンスは必要ありません。<br>1：データ・キャッシュの場合、通常キャッシュ可能な共有位置はライトスルーとして扱われます。命令キャッシュの場合、共有位置はキャッシュ不可として扱われます。プログラムされた内部キャッシュ可能属性は無視されます。すべての書込みはグローバルに参照できます。その他のメモリ・エージェントの更新は、適切なキャッシュのメンテナンスを行わないと、Cortex®-M7 ソフトウェアからは見えません。<br>例えば、Cortex®-M7 プロセッサが、MP 対応プロセッサの *Accelerator Coherency Port*（アクセラレータ・コヒーレンシ・ポート）（ACP）インタフェースに内蔵されている場合など、異種の MP システムで有用。<br>データ・キャッシュを除く場合、このビットは RAZ です。 |

### キャッシュのエラー・チェックと訂正の無効化

キャッシュ・エラー・チェックと訂正がプロセッサに内蔵されている場合、それはリセットによりデフォルトで有効になります。次のコード例は、この機能を無効にするために使用できます。この操作は、PPB メモリ領域の CACR.ECCEN フィールドを変更することによって実行されます。

```
CACR    EQU 0xE000EF9C
        LDR r11, =CACR
        LDR r0, [r11]
        BFC r0, #0x1, #0x1     ; CACR.ECCEN のクリア
        STR r0, [r11]

        DSB
        ISB
```

ソフトウェアで CACR.ECCEN を変更する場合には注意が必要です。キャッシュにデータが含まれているときに CACR.ECCEN が変更された場合、キャッシュ内の ECC 情報は新しい設定に対して正しくなく、予期しないエラーやデータ喪失の原因となる可能性があります。したがって、両方のキャッシュがオフになっているときには、ソフトウェアはCACR.ECCEN のみを変更しなくてはならず、変更後は両方のキャッシュを無効化しなくてはなりません。

## 4.9.4 AHB スレーブ制御レジスタ

AHBSCR は、AHB スレーブ・トラフィックの優先順位を制御するために、ソフトウェアによって使用されます。ビット割当てを次に示します。

**図 63. AHBSCR のビット割当て**

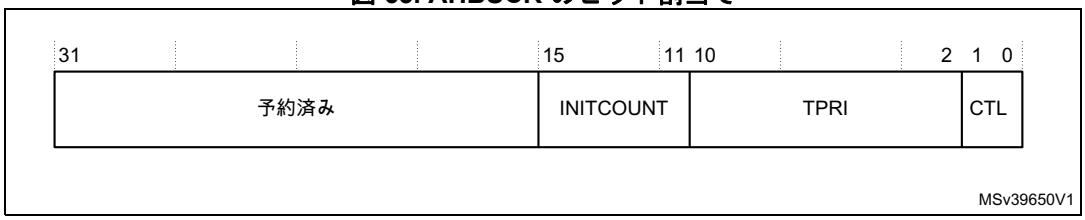



**表 108. AHBSCR ビット割当て**

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [31:16] | - | - | 予約済み。 |
| [15:11] | INITCOUNT | RW | 妥当性カウンタ初期化値。AHBSCR.CTL フィールドで選択されたリクエスタのアクセス優先順位を下げるために使用します。リセット値は 0b01 です。<br>ラウンドロビン・モードの場合、INITCOUNT を 0b01 および AHBSCR.CTL を 0b00 または 0b01 に設定します。<br>INITCOUNT を 0b00 に設定してはなりません。競合が発生した場合、降格されたリクエスタが常に優先され、ライブロックが発生する可能性があるからです。<br>AHBSCR.CTL が 0b11 の場合、INITCOUNT は使用されません。. |

表 108. AHBSCR ビット割当て （続き）

| ビット | 名前 | タイプ | 機能 |
|---|---|---|---|
| [10:2] | TPRI | RW | AHBS トラフィック降格の閾値実行優先度。<br>0b0xxxxxxx : 優先順位は TPRI[7:0] です。これは NVIC レジスタのエンコードと同じです。<br>0b11111111：優先度 -1。これはハード・フォルトの例外の優先順位です。<br>0b11111110：優先度 -2。これはNMIの例外の優先順位です。 |
| [1:0] | CTL | RW | AHBS 優先順位付け制御：<br>0b00：AHBS アクセス優先度の降格これがリセット値です。<br>0b01：ソフトウェアのアクセスの優先順位が降格されました。<br>0b10：ソフトウェア実行の優先順位が AHBSCR.TPRI でプログラムされた閾値レベル以上の場合、妥当性カウンタを AHBSCR.INITCOUNT 値に初期化することによって、AHBS アクセスの優先順位が降格されます。ソフトウェア実行優先順位がこの値を下回る場合、妥当性カウンタが 1 に初期化されます（ラウンドロビン）。<br>閾値レベルのエンコードは NVIC のエンコードと一致し、算術的に大きい数を使用して低い優先順位を表します。<br>0b11：**AHBSPRI** 信号でアクセス優先順位を制御します。 |

## 4.9.5 補助バス･フォールト･ステータス･レジスタ

ABFSR は、非同期バス・フォールトのソースに関する情報を格納します。ASBFSR ビット割当てを次に示します。

図 64. ABFSR ビット割当て

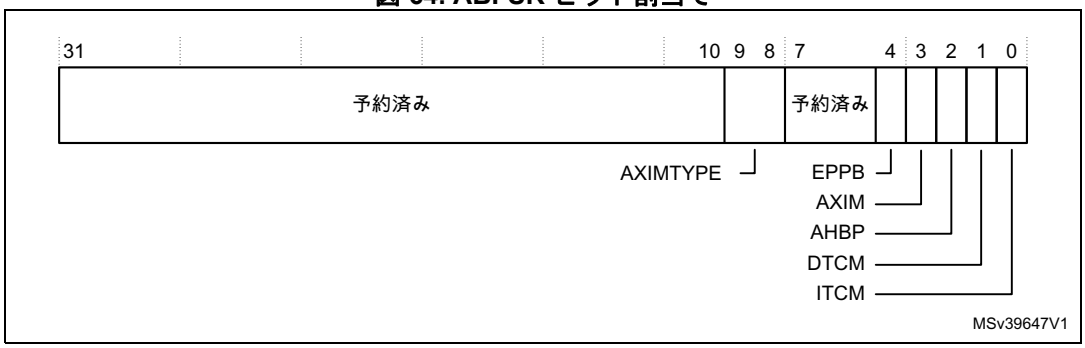



表 109. ABFSR ビット割当て

| ビット | 名前 | 機能 |
|---|---|---|
| [31:10] | - | 予約済みです。 |
| [9:8] | AXIMTYPE | AXIM インタフェースでのフォールトのタイプを示します。<br>b00：OKAY<br>b01：EXOKAY<br>b10：SLVERR<br>b11：DECERR<br>AXIM が 1 の場合のみ有効です。 |
| [7:5] | - | 予約済みです。 |
| [4] | EPPB | EPPB インタフェースでの非同期フォールト |
| [3] | AXIM | AXIM インタフェースでの非同期フォールト |
| [2] | AHBP | AHBP インタフェースでの非同期フォールト |

表 109. ABFSR ビット割当て （続き）

| ビット | 名前 | 機能 |
|---|---|---|
| [1] | DTCM | DTCM インタフェースでの非同期フォールト |
| [0] | ITCM | ITCM インタフェースでの非同期フォールト |

バス・フォールト・ハンドラで、ソフトウェアは BFSR を読み込み、非同期フォールトが発生した場合、影響を受けるインタフェースを決定するために ABFSR を読み込みます。ABFSR[4:0] フィールドは、ABFSR に任意の値を書き込むことによってクリアされるまで有効です。

BFSR の詳細については、204 ページのバス·フォールト·ステータス·レジスタを、参照してください。

# 5 改版履歴

表 110. 文書改版履歴

| 日付 | 版 | 変更内容 |
|---|---|---|
| 2015 年 12 月 18 日 | 1 | 初版発行 |
| 2016 年 4 月 22 日 | 2 | 表 11：STM32F746xx/STM32F756xx Cortex®-M7 の構成のタイトル更新。<br>表 12：STM32F76xxx/STM32F77xxx Cortex®-M7 の構成を追加。<br>セクション 2.3.2：メモリ･システムでのメモリ･アクセスの順序付けでセクション 2.3.4：*software ordering of memory accesses*（メモリアクセスのソフトウェア順序）へのリンクを更新。 |
| 2017 年 2 月 02 日 | 3 | 表 13：STM32F72xxx/STM32F73xxx Cortex®-M7 の構成を追加。 |
| 2017 年 11 月 14 日 | 4 | 資料全体に STM32H7 シリーズを追加。<br>表 14：STM32H7 シリーズ Cortex®-M7の構成を追加。 |
| 2019年 6月 15日 | 5 | 表 85：TYPE のビット割当て で DREGION 関数の説明を更新。<br>セクション 4.7.7：FPU 例外割込みの有効化とクリアを追加。 |

表 111. 日本語版文書改版履歴

| 日付 | 版 | 変更内容 |
|---|---|---|
| 2022 年 8 月 | 1 | 日本語版　初版発行 |

**重要なお知らせ（よくお読み下さい）**

STMicroelectronics NV およびその子会社（以下、ST）は、ST製品及び本書の内容をいつでも予告なく変更、修正、改善、改定及び改良する権利を留保します。購入される方は、発注前にST製品に関する最新の関連情報を必ず入手してください。ST 製品は、注文請書発行時点で有効なST の販売条件に従って販売されます。

ST製品の選択並びに使用については購入される方が全ての責任を負うものとします。購入される方の製品上の操作や設計に関してST は一切の責任を負いません。

明示又は黙示を問わず、STは本書においていかなる知的財産権の実施権も許諾致しません。

本書で説明されている情報とは異なる条件でST 製品が再販された場合、その製品についてSTが与えたいかなる保証も無効となります。

ST およびST ロゴはSTMicroelectronics の商標です。STの登録商標についてはSTウェブサイトをご覧ください。www.st.com/trademarks その他の製品またはサービスの名称は、それぞれの所有者に帰属します。

本書の情報は本書の以前のバージョンで提供された全ての情報に優先し、これに代わるものです。

この資料は、STMicroelectronics NV 並びにその子会社(以下ST)が英文で記述した資料（以下、「正規英語版資料」）を、皆様のご理解の一助として頂くためにSTマイクロエレクトロニクス㈱が英文から和文へ翻訳して作成したものです。この資料は現行の正規英語版資料の近時の更新に対応していない場合があります。この資料は、あくまでも正規英語版資料をご理解頂くための補助的参考資料のみにご利用下さい。この資料で説明される製品のご検討及びご採用にあたりましては、必ず最新の正規英語版資料を事前にご確認下さい。ST及びSTマイクロエレクトロニクス㈱は、現行の正規英語版資料の更新により製品に関する最新の情報を提供しているにも関わらず、当該英語版資料に対応した更新がなされていないこの資料の情報に基づいて発生した問題や障害などにつきましては如何なる責任も負いません。