PM0214
プログラミング·マニュアル

STM32F3 シリーズ、STM32F4 シリーズ、STM32L4 シリーズ、および STM32L4+ シリーズ Cortex®-M4 プログラミング·マニュアル

はじめに

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、STM32 Cortex®-M4 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。

STM32F3 シリーズ、STM32F4 シリーズ、STM32L4 シリーズ、および STM32L4+ シリーズの Cortex®-M4 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
- 内蔵スリープ·モードによる超低消費電力
- プラットフォーム·セキュリティ

参照文献

STMicroelectronics のウェブサイト（www.st.com）では、以下のドキュメントが入手可能です。

- STM32F3 シリーズ、STM32F4 シリーズ、STM32L4 シリーズ、および STM32L4+ シリーズのデータシート
- STM32F3 シリーズ、STM32F4 シリーズ、STM32L4 シリーズ、および STM32L4+ シリーズのリファレンス·マニュアル

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。

1.1 表記規則

本書の表記規則は次のとおりです。

< 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: LDRSB<cond> <Rt>, [<Rn>, #<offset>]
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で使用可能な略語を表します。コマンド名またはオプション名をすべて入力する代わりに、下線部分の文字だけを入力することができます。
`monospace`	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。

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可能です。
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。 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_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。 “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。 “1”を書き込んでも、ビットの値は変化しません。
トグル（t）	このビットは、ソフトウェアによって“1”を書き込んだ場合のみ反転できます。“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。

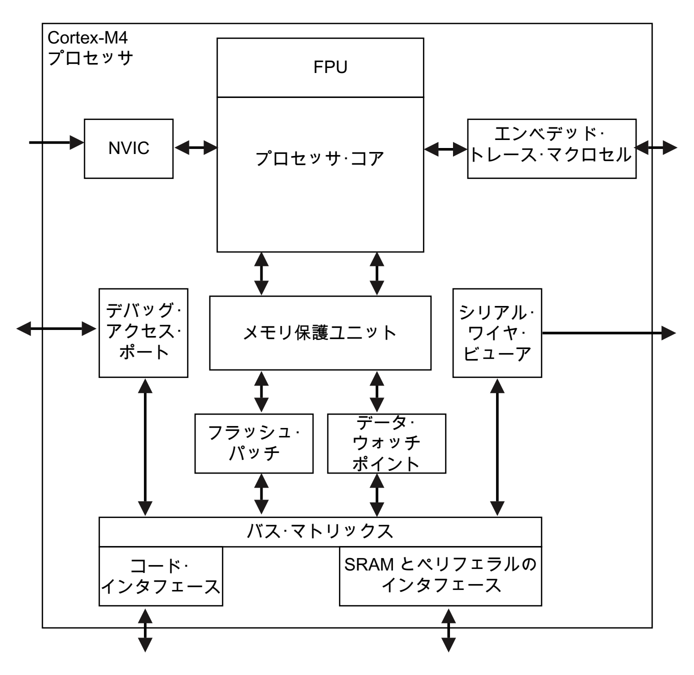
1.3 STM32 Cortex-M4 プロセッサおよびコア･ペリフェラルについて

Cortex-M4 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 モードによる超低消費電力
- プラットフォームに対する、内蔵メモリ保護ユニット（MPU）による堅牢なセキュリティ

Cortex-M4 プロセッサは、3 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。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。

図 1. STM32 Cortex-M4 の実装



Cortex-M4 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-M4 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-M4 プロセッサは、設定可能なネスト化されたベクタ割り込みコントローラ（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、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、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･スタブを一切必要としないため、

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。

省電力設計を最適化するため、NVIC に統合されている SLEEP モードの1つであるディープ･スリープ機能により、STM32 を STOP または STDBY モードに移行できます。

1.3.1 システム･レベル･インタフェース

Cortex-M4 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。アンアラインド･データ･アクセスに対応し、より高速なペリフェラル制御、システム･スピンロック、スレッドセーフなブール･データ処理を可能とする、アトミック･ビット操作を実装しています。

Cortex-M4 プロセッサは、きめ細かくメモリを制御するメモリ保護ユニット（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です。

1.3.2 設定可能な統合デバッグ機能

Cortex-M4 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 ポートまたは小型パッケージ･デバイスに最適な 2 ピンのシリアル･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。

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。

オプションのエンベデッド･トレース･マクロセル™（ETM）は、従来のトレース単位に比べてはるかに狭い領域を対象とした、他に類を見ない命令トレース･キャプチャ機能を提供します。

1.3.3 Cortex-M4 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
- すべての STM32 Cortex-M4 マイクロコントローラに搭載された IEEE754 準拠の単精度 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な乗算器
- 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
- セキュリティ重視のアプリケーション向けのメモリ保護ユニット（MPU）
- 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に必要なピン数が削減されます。

1.3.4 Cortex-M4 のコア･ペリフェラル

以下のペリフェラルを搭載しています。

ネスト化されたベクタ割込みコントローラ

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

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。

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。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

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

浮動小数点ユニット（FPU）は、32bit 単精度浮動小数点の値に対して IEEE754 準拠の演算を実行します。

2 Cortex-M4 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-M4 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･ソフトウェアの実行に使用します。 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。 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（24 ページのCONTROL レジスタを参照）。
ハンドラ･モード：	例外処理に使用します。 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常に特権が与えられます。

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：	非特権ソフトウェアは非特権レベルで実行され、次のような制約があります。 • MSR および MRS 命令へのアクセスが制限され、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。 • 特権ソフトウェアに制御を移すためのスーパバイザ･コールを実行するには、SVC 命令を使用する必要があります。
特権：	特権ソフトウェアは特権レベルで実行され、すべての命令を使用できるとともにすべてのリソースへのアクセスが可能です。 ソフトウェア実行に対する特権レベルを変更するため、CONTROL レジスタへの書込みが可能です。

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。これは、スタック･ポインタがスタック･メモリに最後にスタックされた項目を指すことを意味します。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、それぞれ独立したスタック･ポインタのコピーを持ちます（18 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（24 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、使用するスタックの概要

プロセッサ・モード	以下の実行に使用	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. 24 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

図 2. プロセッサのコア・レジスタ

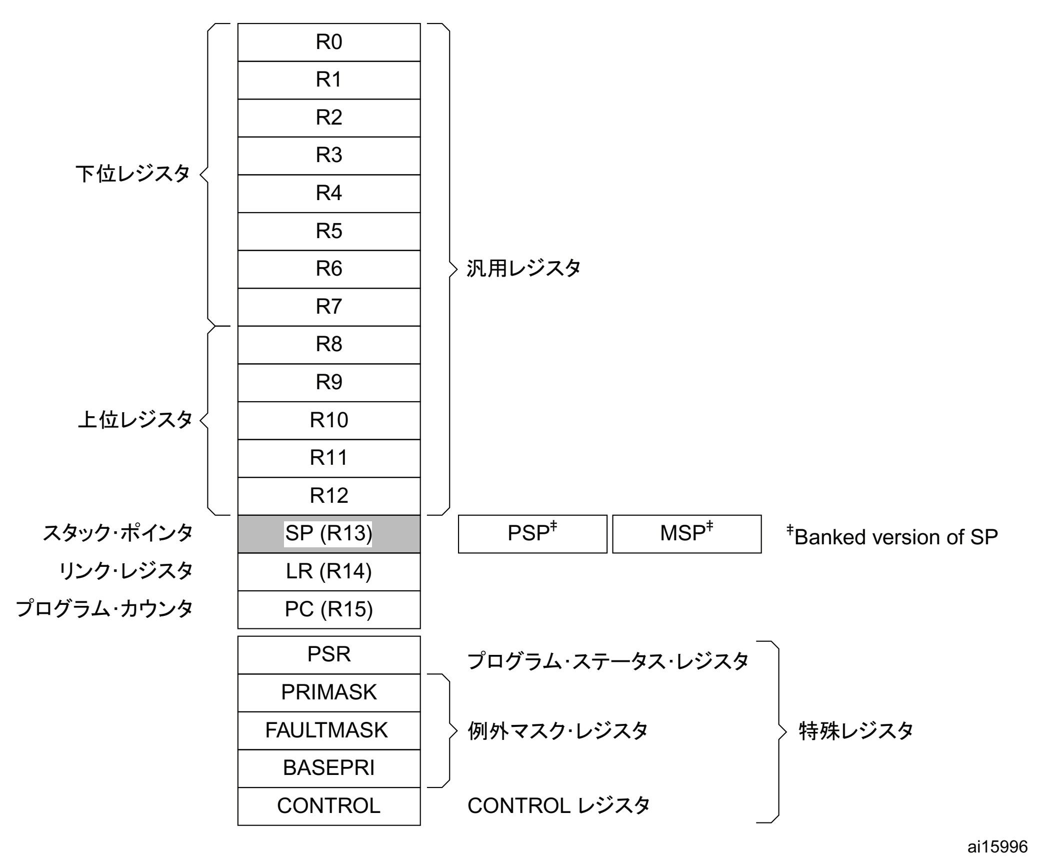


表 2. コア・レジスタ・セットの概要

名前	タイプ(1)	必要な特権(2)	リセット値	説明
R0-R12	読出し／書込み	両方	不明	18 ページの汎用レジスタ
MSP	読出し／書込み	特権	説明を参照	18 ページのスタック・ポインタ
PSP	読出し／書込み	両方	不明	18 ページのスタック・ポインタ
LR	読出し／書込み	両方	0xFFFFFFFF	18 ページのリンク・レジスタ
PC	読出し／書込み	両方	説明を参照	18 ページのプログラム・カウンタ

表 2. コア･レジスタ･セットの概要 （続き）

名前	タイプ[1]	必要な特権[2]	リセット値	説明
PSR	読出し／書込み	特権	0x01000000	18 ページのプログラム･ステータス･レジスタ
ASPR	読出し／書込み	両方	不明	20 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
IPSR	読出し専用	特権	0x00000000	21 ページの割込みプログラム･ステータス･レジスタ
EPSR	読出し専用	特権	0x01000000	21 ページの例外プログラム･ステータス･レジスタ
PRIMASK	読出し／書込み	特権	0x00000000	23 ページの優先度マスク･レジスタ
FAULTMASK	読出し／書込み	特権	0x00000000	23 ページのフォールト･マスク･レジスタ
BASEPRI	読出し／書込み	特権	0x00000000	24 ページのベース優先度マスク･レジスタ
CONTROL	読出し／書込み	特権	0x00000000	24 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。

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 : メイン･スタック･ポインタ（MSP）。これがリセット値です。
- 1 : 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 をロードします。

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

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。

- アプリケーション･プログラム･ステータス･レジスタ（APSR）
- 割込みプログラム･ステータス･レジスタ（IPSR）
- 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。このビット割当てを図 3 と図 4 に示します。

図 3. APSR、IPSR、および EPSR のビット割当て

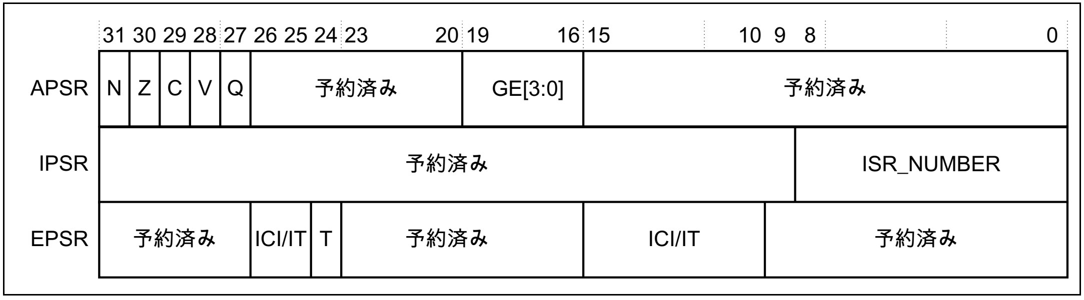
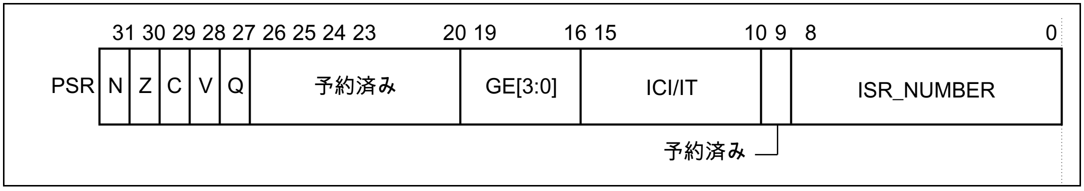


図 4. PSR のビット割当て



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。たとえば、次のように使用します。

- 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	読出し／書込み(1)、(2)	APSR、EPSR、および IPSR
IEPSR	読出し専用	EPSR および IPSR
IAPSR	読出し／書込み(1)	APSR および IPSR
EAPSR	読出し／書込み(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、186 ページのMRS および 187 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、17 ページの表 2 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 のビット定義

ビット	説明
ビット 31	**N** : 負または、より小さいフラグ : 0 : 演算結果が正、ゼロ、より大きい、等しいのいずれかでした。 1 : 演算結果が負または、より小さいでした。
ビット 30	**Z** : ゼロ･フラグ : 0 : 演算結果がゼロではありませんでした。 1 : 演算結果がゼロでした。
ビット 29	**C** : キャリーまたはボロー･フラグ : 0 : 加算によってキャリー･ビットが発生しなかったか、減算によってボロー･ビットが発生しました。 1 : 加算によってキャリー･ビットが発生したか、減算によってボロー･ビットが発生しませんでした。
ビット 28	**V** : オーバーフロー･フラグ : 0 : 演算でオーバーフローは発生しませんでした。 1 : 演算でオーバーフローが発生しました。
ビット 27	**Q** : DSP オーバーフローおよびサチュレーション･フラグ : スティッキー･サチュレーション･フラグ。 0 : リセット以降またはこのビットが最後にゼロにクリアされて以降、サチュレーションが発生していないことを示します。 1 : SSAT または USAT 命令でサチュレーションが発生したか、DSP オーバーフローが発生したことを示します。 このビットは、ソフトウェアで MRS 命令を使用してゼロにクリアされます。
ビット 26:20	予約済み
ビット 19:16	**GE[3:0]** : 以上フラグ。詳細は、105 ページのSEL を参照してください。
ビット 15:0	予約済み

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、17 ページの表 2 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 のビット定義

ビット	説明
ビット 31:9	予約済み
ビット 8:0	**ISR_NUMBER**： これは現在の例外の番号です。 0：スレッド･モード 1：予約済み 2：NMI 3：ハード･フォールト 4：メモリ管理フォールト 5：バス･フォールト 6：用法フォールト 7：予約済み .... 10：予約済み 11：SVCall 12：デバッグ用に予約済み 13：予約済み 14：PendSV 15：SysTick 16：IRQ0(1) .... 83：IRQ81(1) 詳細は、36 ページの例外のタイプを参照してください。

1. 割込みのマッピングの詳細については、STM32の製品リファレンス･マニュアル／データシート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、17 ページの表 2 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 のビット定義

ビット	説明
ビット 31:27	予約済み
ビット 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（22 ページの中断可能で中断後から継続可能な命令を参照）

表 6. EPSR のビット定義 （続き）

ビット	説明
ビット 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）。
ビット 24	T：Thumb 状態ビット
ビット 23:16	予約済み
ビット 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。フォールト･ハンドラは、スタックされた PSR 内の EPSR の値を調べることで、フォールト状態にある動作を示すことができます。41 ページのセクション 2.3.7: 例外の開始と復帰を参照してください。

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、16bit 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。詳細は、145 ページのIT を参照してください。

Thumb 状態

Cortex-M4 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。詳細は、46 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、186 ページのMRS、187 ページのMSR、および 182 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、17 ページの表 2 のレジスタ概要を参照してください。図 5 にビット割当てを示します。

図 5. PRIMASK のビット割当て

表 7. PRIMASK レジスタのビット定義

ビット	説明
ビット 31:1	予約済み
ビット 0	**PRIMASK** : 0 : 影響なし 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、ノンマスカブル割り込み（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、17 ページの表 2 のレジスタ概要を参照してください。図 6 にビット割当てを示します。

図 6. FAULTMASK のビット割当て

表 8. FAULTMASK レジスタのビット定義

ビット	機能
ビット 31:1	予約済み
ビット 0	**FAULTMASK** : 0 : 影響なし 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、17 ページの表 2 のレジスタ概要を参照してください。図 7 にビット割当てを示します。

図 7. BASEPRI のビット割当て

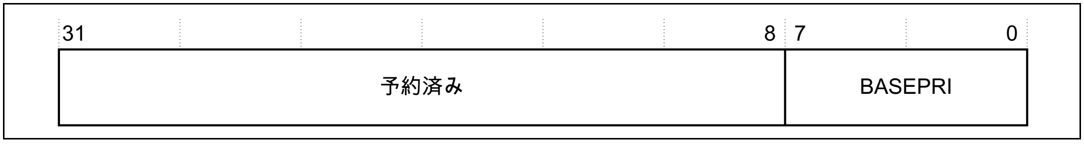


表 9. BASEPRI レジスタのビット割当て

ビット	機能
ビット 31:8	予約済み
ビット 7:4	**BASEPRI[7:4]** 優先度マスク･ビット(1) 0x00：影響なし ゼロ以外：例外処理のベース優先度を定義します。 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。
ビット 3:0	予約済み

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。詳細は、215 ページの割込み優先度レジスタ（NVIC_IPRx）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、17 ページの表 2 のレジスタ概要を参照してください。

表 10. CONTROL レジスタのビット定義

ビット	機能
ビット 31:3	予約済み
ビット 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。 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。 1：浮動小数点コンテキストがアクティブです。 Cortex-M4 は、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。
ビット 1	**SPSEL**：アクティブなスタック･ポインタの選択。現在のスタックを選択します。 0：MSP が現在のスタック･ポインタです。 1：PSP が現在のスタック･ポインタです。 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-M4 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。
ビット 0	**nPRIV**：スレッド･モードの特権レベル。スレッド･モードの特権レベルを定義します。 0：特権 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。CONTROL レジスタは、例外の開始または例外からの復帰のメカニズムによって更新されます。

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、アクティブなスタック･ポインタ･ビットを 1 にセットします（187 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（43 ページの例外からの復帰動作を参照）。

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 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。185 ページのISB を参照してください。

2.1.4 例外と割込み

Cortex-M4 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。プロセッサとネスト化されたベクタ割込みコントローラ（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、41 ページの例外の開始および 43 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。詳細は、208 ページのネスト化されたベクタ割込みコントローラ（NVIC）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。
  - 32bit ワード
  - 16bit ハーフワード
  - 8bit バイト
- すべてのメモリ･アクセスをリトルエンディアンとして管理します。詳細は、28 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 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 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-M4 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-M4 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。

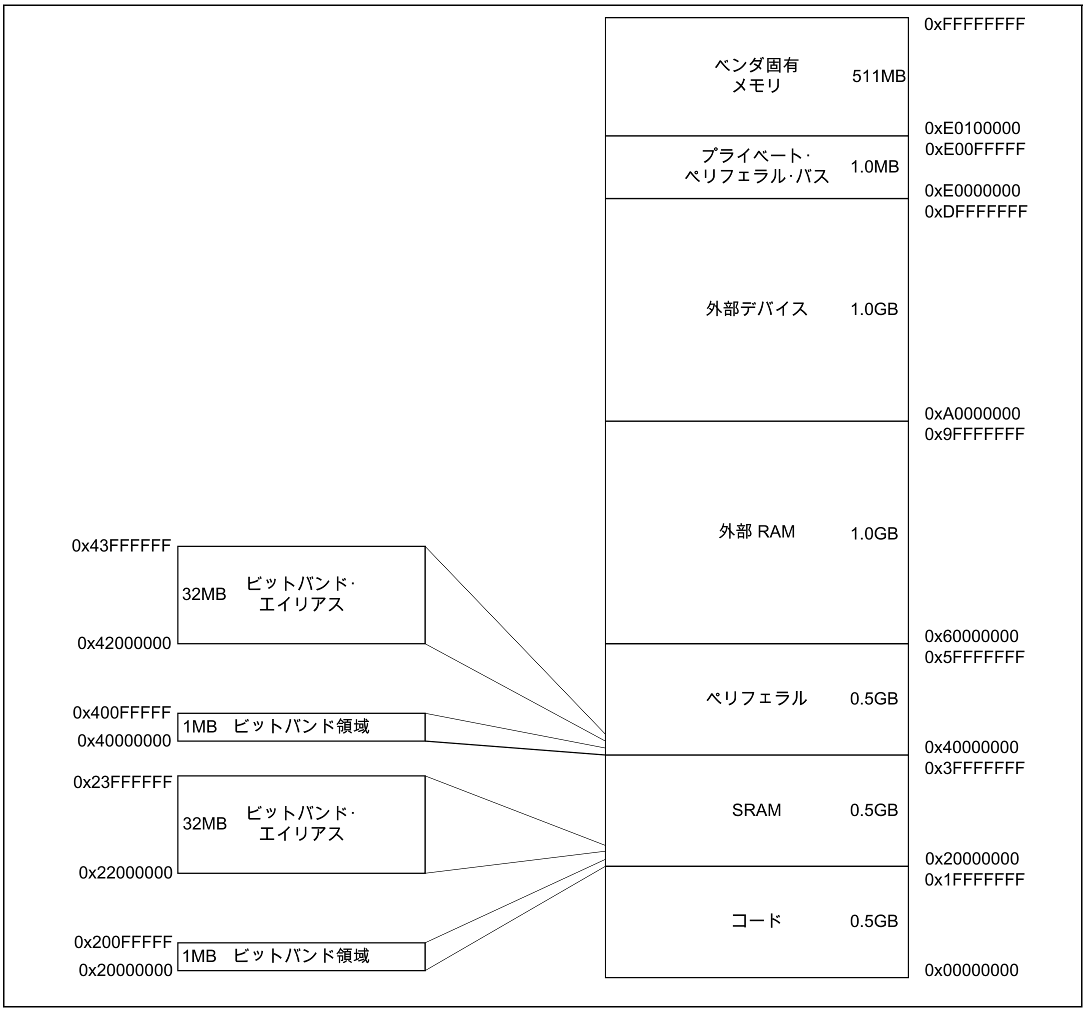
以降のセクションでは、CMSIS の詳細について説明します。

- 48 ページのセクション 2.5.4: 電源管理に関するプログラミングのヒント
- 57 ページのCMSIS 組込み関数
- 210 ページの割込みセット・イネーブル·レジスタ（NVIC_ISERx）
- 218 ページのNVIC のプログラミングのヒント

2.2 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、メモリ･アクセスの動作、およびビットバンド機能について説明します。プロセッサには、最大 4GB のアドレス指定可能なメモリを提供する固定されたメモリ･マップがあります。

図 8. メモリ･マップ



SRAM およびペリフェラルの領域にはビットバンド領域が含まれます。ビットバンドにより、ビット･データに対してアトミック操作を実行できます（30 ページのセクション 2.2.5: ビットバンドを参照）。

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（193 ページのセクション 4.1: STM32 Cortex-M4 のコア･ペリフェラルを参照）。

2.2.1 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	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。
Strongly-ordered	プロセッサは、他のすべて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。

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。XN 領域から命令をフェッチしようとすると、メモリ管理フォールト例外が発生します。

2.2.2 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、アクセスの順序が命令シーケンスの動作に影響を与えない限り、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。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（29 ページのセクション 2.2.4: 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 11. 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	-	<	-	<
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。

2.2.3 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 12. メモリ·アクセスの動作

アドレス範囲	メモリ領域	メモリ·タイプ	XN	説明
0x00000000 ~ 0x1FFFFFFF	コード	ノーマル(1)	-	プログラム·コードの実行可能領域。この領域にデータを配置することも可能です。
0x20000000 ~ 0x3FFFFFFF	SRAM	ノーマル(1)	-	データの実行可能領域。この領域にコードを配置することも可能です。 この領域には、ビットバンドおよびビットバンド·エイリアス領域も含まれます（31 ページの表 13 を参照）。
0x40000000 ~ 0x5FFFFFFF	ペリフェラル	デバイス(1)	XN(1)	この領域には、ビットバンドおよびビットバンド·エイリアス領域も含まれます（31 ページの表 14 を参照）。
0x60000000 ~ 0x9FFFFFFF	外部 RAM	ノーマル(1)	-	データの実行可能領域。
0xA0000000 ~ 0xDFFFFFFF	外部デバイス	デバイス(1)	XN(1)	外部デバイス·メモリ
0xED000000 ~ 0xED0FFFFF	専用ペリフェラル·バス	Strongly-ordered(1)	XN(1)	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。
0xED100000 ~ 0xFFFFFFFF	メモリ·マップ·ペリフェラル	デバイス(1)	XN(1)	この領域には、すべてのSTM32 標準ペリフェラルが含まれます。

1. 詳細は、28 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。ただし、プログラムには常にコード領域を使用することを推奨します。これは、プロセッサには、命令フェッチとデータ·アクセスを同時に実行できる個別のバスがあるためです。

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、193 ページのメモリ保護ユニット（MPU）を参照してください。

命令プリフェッチと分岐予測

Cortex-M4 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

2.2.4 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、以下の理由によります。

- 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。

28 ページのセクション 2.2.2: 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	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。183 ページのDMB を参照してください。
DSB	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。184 ページのDSB を参照してください。
ISB	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。185 ページのISB を参照してください。

メモリ･バリア命令は、たとえば次のような場合に使用します。

- **ベクタ･テーブル。**プログラムでベクタ･テーブル内のエントリを変更してから対応する例外を有効にする場合は、これらの操作の間に DMB 命令を使用します。これにより、例外が有効になった直後にその例外が取得された場合、プロセッサは確実に新しい例外ベクタを使用します。
- **自己修正コード。**プログラムに自己修正コードが含まれる場合、プログラムのコード修正直後に ISB 命令を使用します。これにより、後続の命令が必ず更新されたプログラムを使って実行されるようになります。
- **メモリ･マップの切り換え。**システムがメモリ･マップの切り換えメカニズムを備えている場合、プログラム内のメモリ･マップの切り換え後に DSB 命令を使用します。これにより、後続の命令が必ず更新されたメモリ･マップを使って実行されるようになります。
- **例外優先度の動的な変更。**例外が保留中またはアクティブな状態で、その例外の優先度を変更する必要がある場合は、変更後に DSB 命令を使用します。これにより、DSB 命令の完了時に変更が必ず有効になります。
- **マルチマスタ･システムにおけるセマフォの使用。**システムに複数のバス･マスタが存在する場合（たとえば、システム内に別のプロセッサが存在する場合）、各プロセッサはすべてのセマフォ命令後に DMB 命令を使用して、もう一方のバス･マスタが、メモリ･トランザクションをそれらが実行された順に認識できるようにする必要があります。

システム制御ブロックなどの Strongly-ordered メモリへのアクセスでは、DMB 命令を使用する必要はありません。

MPU のプログラミングでは、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。

2.2.5 ビットバンド

ビットバンド領域では、ビットバンド･エイリアス領域内の各ワードを、ビットバンド領域内の単一ビットにマッピングします。ビットバンド領域は、SRAM およびペリフェラル･メモリ領域の最下位 1MB を占有します。

メモリ･マップには 32MB のエイリアス領域が 2 つあり、次のように 1MB のビットバンド領域 2 つにマッピングされます。

- 32MB の SRAM エイリアス領域へのアクセスは、表 13 に示すように 1MB の SRAM ビットバンド領域にマッピングされます。
- 32MB のペリフェラル･エイリアス領域へのアクセスは、表 14 に示すように 1MB のペリフェラル･ビットバンド領域にマッピングされます。

表 13. SRAM メモリ･ビットバンド領域

アドレス範囲	メモリ領域	命令アクセスとデータ･アクセス
0x20000000 ~ 0x200FFFFF	SRAM ビットバンド領域	このメモリ範囲への直接アクセスは SRAM メモリ･アクセスのように動作しますが、ここはビットバンド･エイリアス経由でビットアドレス可能な領域でもあります。
0x22000000 ~ 0x23FFFFFF	SRAM ビットバンド･エイリアス	この領域へのデータ･アクセスは、ビットバンド領域に再マッピングされます。書込み操作は、読出し-変更-書込みの手順で実行されます。命令アクセスは再マッピングされません。

表 14. ペリフェラル･メモリ･ビットバンド領域

アドレス範囲	メモリ領域	命令アクセスとデータ･アクセス
0x40000000 ~ 0x400FFFFF	ペリフェラル ビットバンド領域	このメモリ範囲への直接アクセスはペリフェラル･メモリ･アクセスのように動作しますが、ここはビットバンド･エイリアス経由でビットアドレス可能な領域でもあります。
0x42000000 ~ 0x43FFFFFF	ペリフェラル ビットバンド･エイリアス	この領域へのデータ･アクセスは、ビットバンド領域に再マッピングされます。書込み操作は、読出し-変更-書込みの手順で実行されます。命令アクセスは許可されません。

注 : **SRAM またはペリフェラルのビットバンド･エイリアス領域へのワード･アクセスは、SRAM またはペリフェラルのビットバンド領域の単一ビットにマッピングされます。**

ビットバンド･アクセスには、バイト、ハーフワード、またはワード転送を使用できます。ビットバンド転送サイズは、ビットバンド･アクセスを実行する命令の転送サイズと一致します。

次の式は、エイリアス領域からビットバンド領域へのマッピングを示します。

```
bit_word_offset = (byte_offset x 32) + (bit_number x 4)

bit_word_addr = bit_band_base + bit_word_offset
```

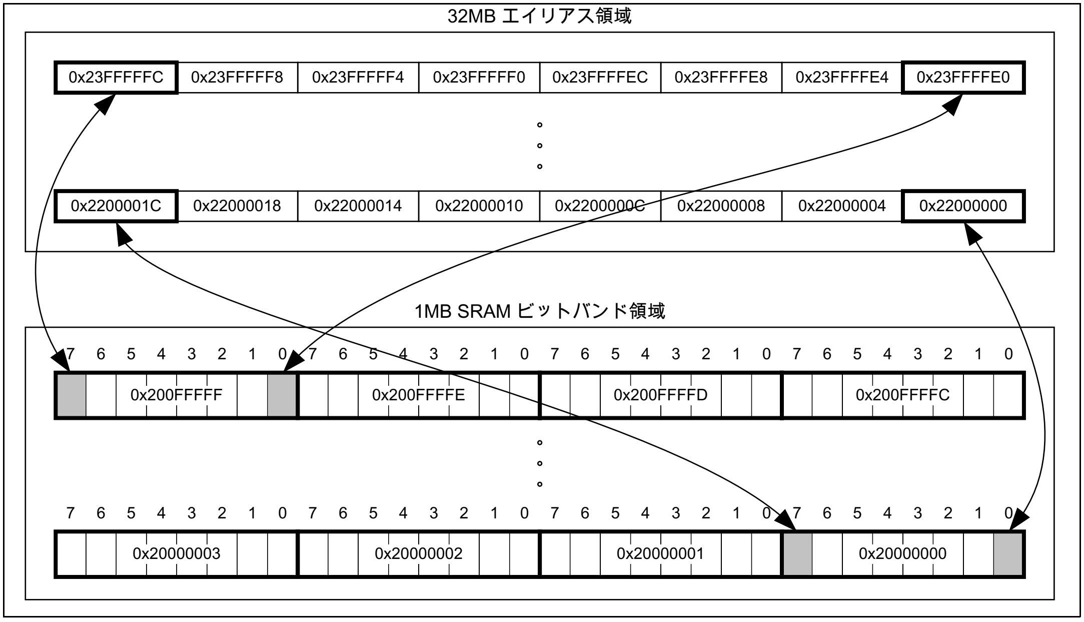
ここで、

- bit_word_offset は、ビットバンド･メモリ領域内の対象ビットの位置です。
- bit_word_addr は、対象ビットにマッピングされるエイリアス･メモリ領域内のワードのアドレスです。
- bit_band_base は、エイリアス領域の開始アドレスです。
- byte_offset は、対象ビットを含むビットバンド領域のバイト数です。
- bit_number は、対象ビットのビット位置（0 ~ 7）です。

32 ページの図 9 に、SRAM ビットバンド･エイリアス領域と SRAM ビットバンド領域間のビットバンド･マッピングの例を示します。

- 0x23FFFFED のエイリアス･ワードは、0x200FFFFF のビットバンド･バイトのビット [0] にマッピングされます。0x23FFFFED = 0x22000000 + (0xFFFFF*32) + (0*4)
- 0x23FFFFFC のエイリアス･ワードは、0x200FFFFF のビットバンド･バイトのビット [7] にマッピングされます。0x23FFFFFC = 0x22000000 + (0xFFFFF*32) + (7*4)
- 0x22000000 のエイリアス･ワードは、0x20000000 のビットバンド･バイトのビット [0] にマッピングされます。0x22000000 = 0x22000000 + (0*32) + (0 *4)
- 0x2200001C のエイリアス･ワードは、0x20000000 のビットバンド･バイトのビット [7] にマッピングされます。0x2200001C = 0x22000000+ (0*32) + (7*4)

図 9. ビットバンド･マッピング



エイリアス領域への直接アクセス

エイリアス領域内のワードに書き込むと、ビットバンド領域内の単一ビットが更新されます。

エイリアス領域内のワードに書き込まれた値のビット [0] によって、ビットバンド領域内の対象ビットに書き込まれる値が決まります。ビット [0] が 1 にセットされている値を書き込むと、ビットバンド･ビットに 1 が書き込まれ、ビット [0] が 0 にセットされている値を書き込むと、ビットバンド･ビットに 0 が書き込まれます。

エイリアス･ワードのビット [31:1] は、ビットバンド･ビットに影響を及ぼしません。0x01 の書込みは、0xFF の書込みと同じ効果があります。0x00 の書込みは、0x0E の書込みと同じ効果があります。

エイリアス領域内のワードの読み出しは、以下のことを意味します。

- 0x00000000 は、ビットバンド領域内の対象ビットが 0 にセットされていることを示します。
- 0x00000001 は、ビットバンド領域内の対象ビットが 1 にセットされていることを示します。

ビットバンド領域への直接アクセス

29 ページのメモリ･アクセスの動作は、ビットバンド領域への直接のバイト、ハーフワード、およびワードのアクセス動作を示します。

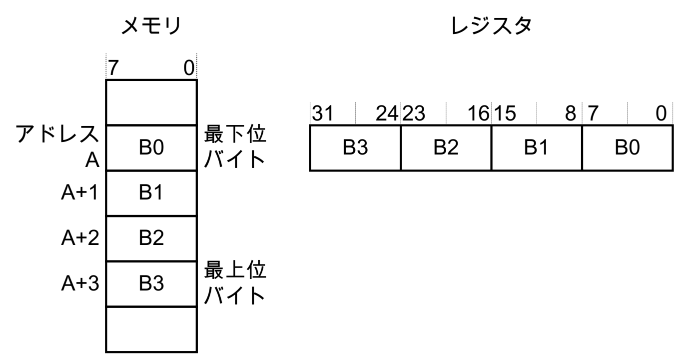
2.2.6 メモリのエンディアン形式

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。

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。例を図 10 に示します。

図 10. リトルエンディアン形式の例



2.2.7 同期プリミティブ

Cortex-M4 命令セットには、数組の同期プリミティブが存在します。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排他的アクセスを獲得し、書込みは成功です。

  1 : そのスレッドまたはプロセスは排他的アクセスを獲得できず、書込みは実行されません。

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。

メモリ位置の保証された読出し-変更-書込みを実行するには、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. 必要に応じて値を更新します。
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。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。

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-M4 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、マルチプロセッサ･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。
- 排他ストア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の場合、次の処理が行われます。

- CLREX 命令は、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。
- 排他ストア命令または例外は、ローカルな排他アクセス･タグ、およびプロセッサのグローバルな排他アクセス･タグを削除します。

同期プリミティブ命令の詳細については、79 ページのLDREX と STREX および 80 ページのCLREX を参照してください。

2.2.8 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 15. 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
```

2.3 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。

2.3.1 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。
保留中	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。

2.3.2 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。
ハード･フォールト	ハード･フォールトは、例外の処理中のエラーによって、または例外が他の例外メカニズムで管理できないことによって、発生する例外です。ハード･フォールトの優先度は -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。
メモリ管理フォールト	メモリ管理フォールトは、メモリ保護関連のフォールトが原因で発生する例外です。MPU または固定メモリ保護制約が、命令とデータの両方のメモリ･トランザクションについてこのフォールトを判断します。このフォールトは、実行不可（XN）メモリ領域への命令アクセスを中止するために使用されます。
バス･フォールト	バス･フォールトは、命令またはデータのメモリ･トランザクションでのメモリ関連のフォールトによって発生する例外です。メモリ･システムのバスで検出されたエラーが原因である場合もあります。

用法フォールト

用法フォールトは、命令実行フォールトが発生した場合に発生する例外です。これには、次の場合が含まれます。

- 未定義命令
- 不正なアンアラインド･アクセス
- 命令実行の無効な状態
- 例外復帰時のエラー

コアが用法フォールトをレポートするように設定されている場合は、以下によって用法フォールトが発生する可能性があります。

- ワードおよびハーフワード･メモリ･アクセスでの整列されていないアドレス
- 0 による除算

SVCall

スーパバイザ･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 16. 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)	例外タイプ	優先度	ベクタ･アドレスまたはオフセット(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	ハード･フォールト	-1	0x0000000C	-
4	-12	メモリ管理フォールト	設定可能(3)	0x00000010	同期
5	-11	バス･フォールト	設定可能(3)	0x00000014	正確なバス･フォールトの場合は同期、不正確なバス･フォールトの場合は非同期
6	-10	用法フォールト	設定可能(3)	0x00000018	同期
7 ~ 10	-	-	-	予約済み	-
11	-5	SVCall	設定可能(3)	0x0000002C	同期
12 ~ 13	-	-	-	予約済み	-
14	-2	PendSV	設定可能(3)	0x00000038	非同期
15	-1	SysTick	設定可能(3)	0x0000003C	非同期
16 以上	0 以上	割込み（IRQ）	設定可能(4)	0x00000040 以上(5)	非同期

1. ソフトウェア層を簡素化するため、CMSIS は IRQ 番号のみを使用します。そのため、割込み以外の例外には負値を使用します。IPSR は、例外番号を返します。詳細については、21 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、39 ページのベクタ･テーブルを参照してください。
3. 232 ページのシステム･ハンドラ優先度レジスタ（SHPRx）を参照してください。
4. 215 ページの割込み優先度レジスタ（NVIC_IPRx）を参照してください。
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、別の命令を実行できます。

特権ソフトウェアは、37 ページの表 16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。詳細については、次を参照してください。

- 234 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（SHCSR）
- 211 ページの割込みクリア・イネーブル･レジスタ（NVIC_ICERx）

ハード･フォールト、メモリ管理フォールト、バス･フォールト、および用例フォールトの詳細については、43 ページのセクション 2.4: フォールト処理を参照してください。

2.3.3 例外ハンドラ

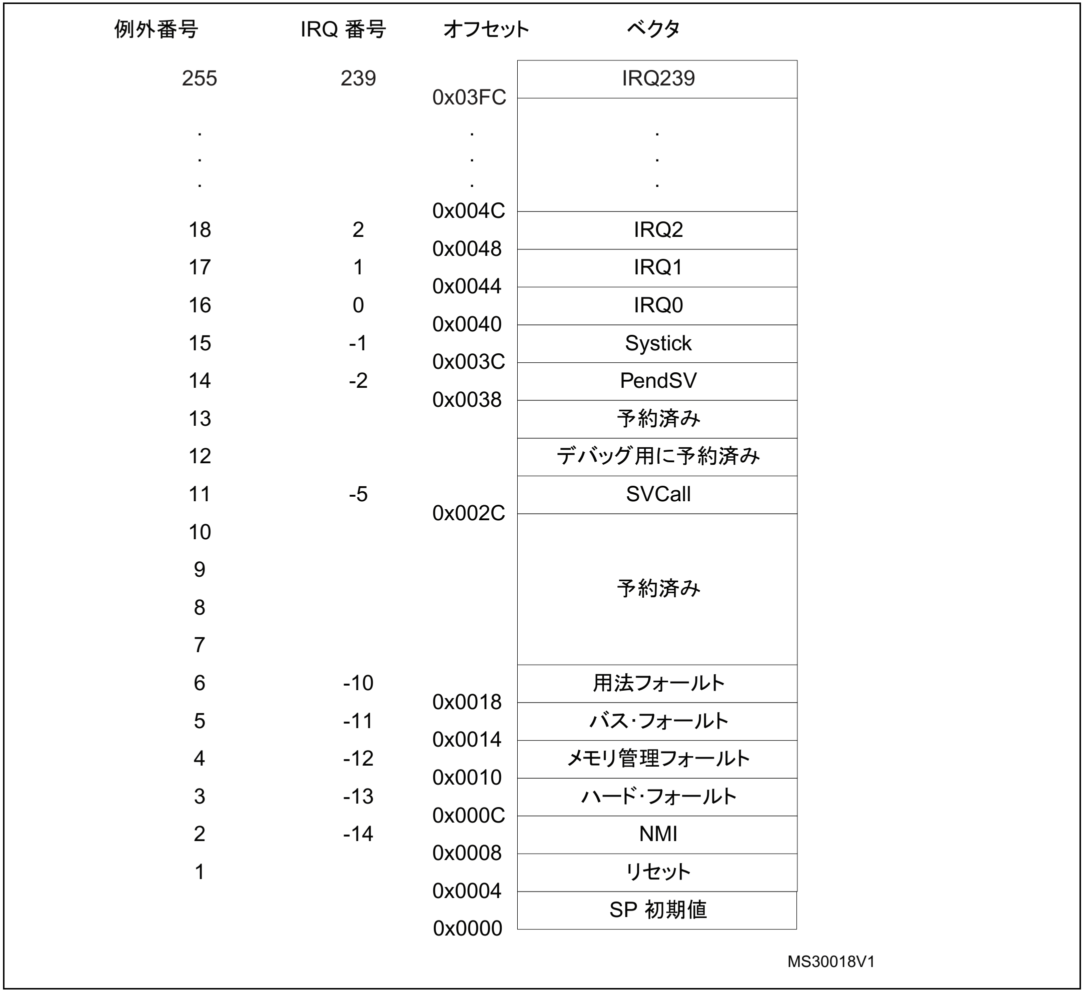
プロセッサは、以下を使用して例外を処理します。

割込みサービス･ルーチン（ISR）	IRQ0 ~ IRQ81 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･フォールト、メモリ管理フォールト、用例フォールト、およびバス･フォールトは、フォールト･ハンドラが処理す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。

2.3.4 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。39 ページの図 11 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。

図 11. ベクタ･テーブル



システム･リセット時に、ベクタ･テーブルはアドレス 0x00000000 に固定されます。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 0x00000080 ~ 0x3FFFFF80 の範囲の別のメモリ位置に再配置できます。詳細については、226 ページのベクタ･テーブル･オフセット･レジスタ（VTOR）を参照してください。

2.3.5 例外の優先度

37 ページの表 16 は、すべての例外に優先度が関連付けられていることを詳細に示しています。

- 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･フォールト、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、優先度を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。

- 232 ページのシステム･ハンドラ優先度レジスタ（SHPRx）
- 215 ページの割込み優先度レジスタ（NVIC_IPRx）

設定可能な優先度の値の範囲は、0 ~ 15 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･フォールト、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。

2.3.6 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、227 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（AIRCR）を参照してください。

2.3.7 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、セクション 2.3.6: 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、41 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、43 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）

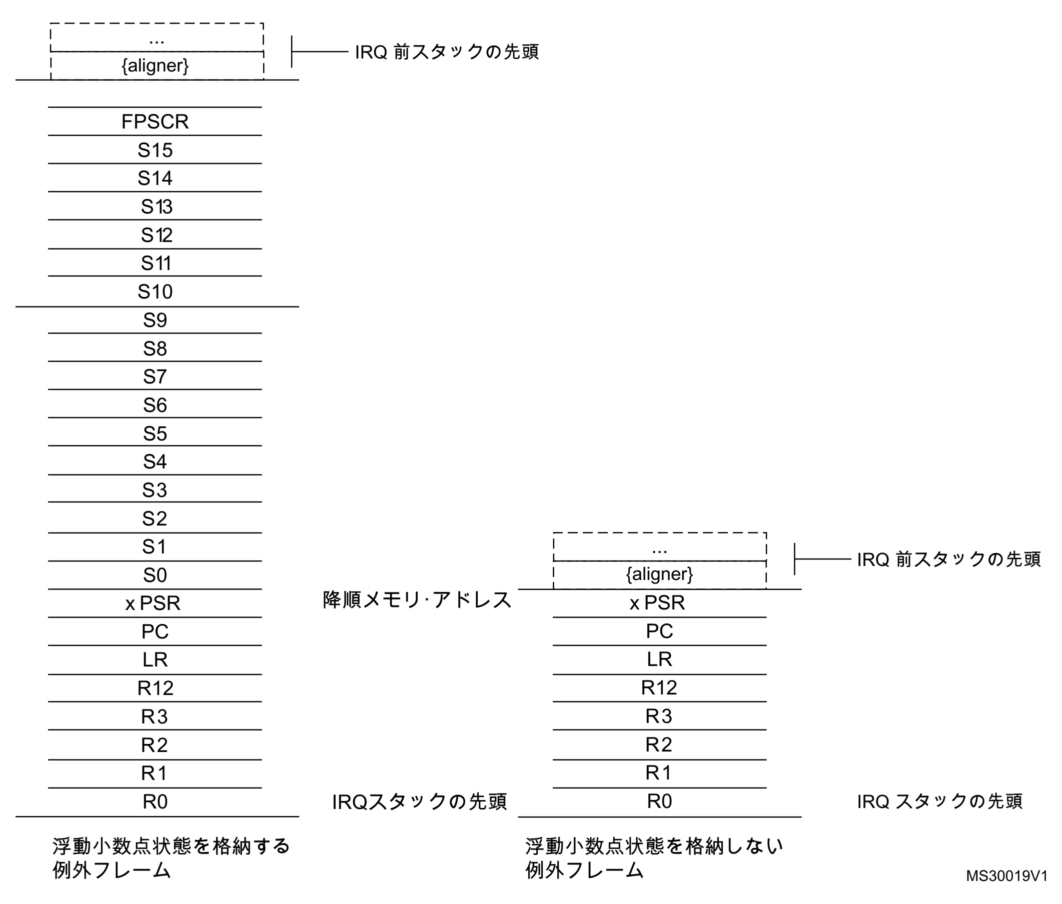
例外が別の例外を横取りする場合、これらの例外はネストされています。

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。詳細については、22 ページの例外マスク･レジスタを参照してください。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-M4 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。42 ページの図 12 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-M4 スタック･フレーム･レイアウトを示します。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。42 ページの図 12 には、このスタック･フレームも示されています。

図 12. Cortex-M4 スタック·フレームのレイアウト



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、構成制御レジスタ（CCR）の STKALIGN ビットによって制御されます。

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 17 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 17. 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。

2.4 フォールト処理

フォールトは、例外のサブセットです。詳細については、36 ページの例外モデルを参照してください。フォールトを生成する要素を次に示します。

- 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
- 実行不可（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

2.4.1 フォールト･タイプ

表 18 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、236 ページの設定可能フォールト･ステータス･レジスタ（CFSR：UFSR+BFSR+MMFSR）を参照してください。

表 18. 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	ハード･フォールト	VECTTBL	240 ページのハード･フォールト･ステータス･レジスタ（HFSR）
ハード･フォールトに昇格されたフォールト		FORCED	
MPU またはデフォルトのメモリ･マップの不一致：	メモリ管理	-	241 ページのメモリ管理フォールト･アドレス･レジスタ（MMFAR）
– 命令アクセス時		IACCVIOL(1)	
– 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：	バス･フォールト	-	-
– 例外スタッキング中		STKERR	241 ページのバス･フォールト･アドレス･レジスタ（BFAR）
– 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	用法フォールト	NOCP	236 ページの設定可能フォールト･ステータス･レジスタ（CFSR：UFSR+BFSR+MMFSR）
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. MPU が無効な場合であっても、XN 領域へのアクセスで発生します。
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

2.4.2 フォールトの昇格とハード･フォールト

ハード･フォー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（232 ページのシステム･ハンドラ優先度レジスタ（SHPRx）を参照）。これらのフォールトのハンドラの実行は、ソフトウェアで無効化できます（234 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（SHCSR）を参照）。

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（36 ページのセクション 2.3: 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。これは優先度の昇格と呼ばれ、フォールトはハード･フォールトに昇格されたと言われます。ハード･フォールトへの昇格は、次の場合に発生します。

- 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･フォールトへの昇格が発生します。
- 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･ハンドラを開始する際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、このバス･フォールトはハード･フォールトに昇格しません。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。

固定優先度のハード･フォー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･フォールトを除くすべての例外を横取りできます。

2.4.3 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。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 19 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 19. 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
ハード･フォールト	HFSR	-	240 ページのハード･フォールト･ステータス･レジスタ（HFSR）
メモリ管理フォールト	MMFSR	MMFAR	241 ページのメモリ管理フォールト･アドレス･レジスタ（MMFAR）
バス･フォールト	BFSR	BFAR	241 ページのバス･フォールト･アドレス･レジスタ（BFAR）
用法フォールト	UFSR	-	236 ページの設定可能フォールト･ステータス･レジスタ（CFSR：UFSR+BFSR+MMFSR）

2.4.4 ロックアップ

NMI またはハード･フォールトのハンドラを実行している間にハード･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

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。

2.5 電源管理

STM32 と Cortex-M4 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。他のすべてのシステムおよびペリフェラル･クロックは、動作し続ける可能性があります。
- DEEP SLEEP モードでは、STM32 システムとペリフェラルのクロックの大部分が停止します。製品レベルでは、これは STOP モードまたは STANDBY モードのどちらかに対応します。詳細については、STM32 リファレンス･マニュアルの「Low-power modes」のセクションを参照してください。

SCR の SLEEPDEEP ビットにより、使用する SLEEP モードを選択します（229 ページのシステム制御レジスタ（SCR）を参照）。SLEEP モードの動作の詳細については、STM32 製品リファレンス･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。

2.5.1 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。

割込み待機

割込み待機命令 WFI は、ただちに SLEEP モードに移行させます（ウェイクアップ条件が真の場合を除く。47 ページのWFI または sleep-on-exit からのウェイクアップを参照）。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、192 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。

詳細は、191 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 189 ページのSEV のような SEV 命令が実行されているためです。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。

2.5.2 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、22 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
- 例外イベント信号を検出した場合（セクション 2.5.3: 外部イベント入力／拡張割込みおよび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、229 ページのシステム制御レジスタ（SCR）を参照してください。

2.5.3 外部イベント入力／拡張割込みおよびイベント入力

プロセッサは外部イベント入力信号を供給します。

この信号は、非同期イベントの検出（外部入力ピンまたは非同期ペリフェラル･イベントより）時に外部または拡張割込み／イベント･コントローラ（EXTI）により生成されます。

この信号は、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 (47 ページのイベント待機を参照)。詳細については、STM32 リファレンス･マニュアルの低電力モードのセクションを参照してください。

2.5.4 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 STM32 Cortex-M4 命令セット

この章は、ユーザ･ガイドにある Cortex-M4 命令セットの説明の参照資料です。以下のセクションでは全般的な情報を提供します。

以下の各セクションでは、Cortex-M4 命令の機能グループについて説明します。これらによって、Cortex-M4 プロセッサでサポートされるすべての命令が説明されています。

3.1 命令セットの概要

プロセッサは Thumb 命令セット･バージョンを実装しています。表 20 にサポートされる命令を示します。

表 20 では、次のように表記されています。

- 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 20. Cortex-M4 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	83 ページの 3.5.1
ADD、ADDS	{Rd}、Rn、Op2	加算	N、Z、C、V	83 ページの 3.5.1
ADD、ADDW	{Rd}、Rn、#imm12	加算	N、Z、C、V	83 ページの 3.5.1
ADR	Rd、label	PC 相対アドレスのロード	—	69 ページの 3.4.1

表 20. Cortex-M4 命令 （続き）

ニーモニック	オペランド	概要	フラグ	ページ
AND、ANDS	{Rd}、Rn、Op2	論理積	N、Z、C	85 ページの 3.5.2
ASR、ASRS	Rd、Rm、<Rs|#n>	算術右シフト	N、Z、C	86 ページの 3.5.3
B	label	分岐	—	142 ページの 3.9.5
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	85 ページの 3.5.2
BKPT	#imm	ブレークポイント	—	181 ページの 3.11.1
BL	label	リンク付き分岐	—	142 ページの 3.9.5
BLX	Rm	リンク付き間接分岐	—	142 ページの 3.9.5
BX	Rm	間接分岐	—	142 ページの 3.9.5
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 3.9.6
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 3.9.6
CLREX	—	排他をクリア	—	80 ページの 3.4.9
CLZ	Rd、Rm	先行ゼロ・カウント	—	87 ページの 3.5.4
CMN	Rn、Op2	比較否定	N、Z、C、V	88 ページの 3.5.5
CMP	Rn、Op2	比較	N、Z、C、V	88 ページの 3.5.5
CPSID	iflags	プロセッサ状態の変更、 割込みを無効にする	—	182 ページの 3.11.2
CPSIE	iflags	プロセッサ状態の変更、 割込みを有効にする	—	182 ページの 3.11.2
DMB	—	データ・メモリ・バリア	—	184 ページの 3.11.4
DSB	—	データ同期バリア	—	184 ページの 3.11.4
EOR、EORS	{Rd}、Rn、Op2	排他的論理和	N、Z、C	85 ページの 3.5.2
ISB	—	命令同期バリア	—	185 ページの 3.11.5
IT	—	If-Then 条件ブロック	—	145 ページの 3.9.7
LDM	Rn{!}、reglist	多重レジスタ・ロード、 ポスト・インクリメント	—	76 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	—	76 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	—	76 ページの 3.4.6
LDR	Rt、[Rn、#offset]	レジスタ・ロード（ワード）	—	68 ページの 3.4
LDRB、LDRBT	Rt、[Rn、#offset]	レジスタ・ロード（バイト）	—	68 ページの 3.4
LDRD	Rt、Rt2、[Rn、#offset]	レジスタ・ロード（2 バイト）	—	70 ページの 3.4.2
LDREX	Rt、[Rn、#offset]	排他的レジスタ・ロード	—	79 ページの 3.4.8
LDREXB	Rt、[Rn]	排他的レジスタ・ロード （バイト）	—	79 ページの 3.4.8

表 20. Cortex-M4 命令 （続き）

ニーモニック	オペランド	概要	フラグ	ページ
LDREXH	Rt、[Rn]	排他的レジスタ・ロード （ハーフワード）	—	79 ページの 3.4.8
LDRH、LDRHT	Rt、[Rn、#offset]	レジスタ・ロード （ハーフワード）	—	68 ページの 3.4
LDRSB、 LDRSBT	Rt、[Rn、#offset]	レジスタ・ロード （符号付きバイト）	—	68 ページの 3.4
LDRSH、 LDRSHT	Rt、[Rn、#offset]	レジスタ・ロード （符号付きハーフワード）	—	68 ページの 3.4
LDRT	Rt、[Rn、#offset]	レジスタ・ロード （ワード）	—	68 ページの 3.4
LSL、LSLS	Rd、Rm、<Rs|#n>	論理左シフト	N、Z、C	86 ページの 3.5.3
LSR、LSRS	Rd、Rm、<Rs|#n>	論理右シフト	N、Z、C	86 ページの 3.5.3
MLA	Rd、Rn、Rm、Ra	積和、32 ビットの結果	—	110 ページの 3.6.1
MLS	Rd、Rn、Rm、Ra	積差、32 ビットの結果	—	110 ページの 3.6.1
MOV、MOVS	Rd、Op2	転送	N、Z、C	89 ページの 3.5.6
MOVT	Rd、#imm16	上位に転送	—	91 ページの 3.5.7
MOVW、MOV	Rd、#imm16	16 ビット定数を転送	N、Z、C	89 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	—	186 ページの 3.11.6
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	187 ページの 3.11.7
MUL、MULS	{Rd}、Rn、Rm	乗算、32 ビットの結果	N、Z	110 ページの 3.6.1
MVN、MVNS	Rd、Op2	転送して否定	N、Z、C	89 ページの 3.5.6
NOP	—	何もしない	—	188 ページの 3.11.8
ORN、ORNS	{Rd}、Rn、Op2	否定論理和	N、Z、C	85 ページの 3.5.2
ORR、ORRS	{Rd}、Rn、Op2	論理和	N、Z、C	85 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
POP	reglist	レジスタをスタックから ポップ	—	78 ページの 3.4.7
PUSH	reglist	レジスタをスタックに プッシュ	—	78 ページの 3.4.7
QADD	{Rd}、Rn、Rm	飽和倍演算および加算		128 ページの 3.7.3
QADD16	{Rd}、Rn、Rm	16 ビット飽和加算		128 ページの 3.7.3
QADD8	{Rd}、Rn、Rm	8 ビット飽和加算		128 ページの 3.7.3
QASX	{Rd}、Rn、Rm	飽和加算および減算、交換付き		129 ページの 3.7.4
QDADD	{Rd}、Rn、Rm	飽和加算		130 ページの 3.7.5
QDSUB	{Rd}、Rn、Rm	飽和倍演算および減算		130 ページの 3.7.5
QSAX	{Rd}、Rn、Rm	飽和減算および加算、交換付き		129 ページの 3.7.4
QSUB	{Rd}、Rn、Rm	飽和減算		128 ページの 3.7.3
QSUB16	{Rd}、Rn、Rm	16 ビット飽和減算		129 ページの 3.7.4

表 20. Cortex-M4 命令 （続き）

ニーモニック	オペランド	概要	フラグ	ページ
QSUB8	{Rd}、Rn、Rm	8 ビット飽和減算		129 ページの 3.7.4
RBIT	Rd、Rn	ビット順序反転	—	129 ページの 3.7.4
REV	Rd、Rn	ワード内のバイト順序反転	—	92 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	—	92 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	—	92 ページの 3.5.8
ROR、RORS	Rd、Rm、<Rs|#n>	右ローテート	N、Z、C	86 ページの 3.5.3
RRX、RRXS	Rd、Rm	右ローテート、拡張付き	N、Z、C	86 ページの 3.5.3
RSB、RSBS	{Rd}、Rn、Op2	反転減算	N、Z、C、V	83 ページの 3.5.1
SADD16	{Rd}、Rn、Rm	16 ビット符号付き加算		93 ページの 3.5.9
SADD8	{Rd}、Rn、Rm	8 ビット符号付き加算		93 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		98 ページの 3.5.14
SBC、SBCS	{Rd}、Rn、Op2	キャリー付き減算	N、Z、C、V	83 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
SDIV	{Rd}、Rn、Rm	符号付き除算	—	112 ページの 3.6.3
SEV	—	イベント送信	—	189 ページの 3.11.9
SHADD16	{Rd}、Rn、Rm	結果を半減する 16 ビット符号付き加算	—	94 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	—	94 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	—	95 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	—	95 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	—	96 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	—	96 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	112 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	—	111 ページの 3.6.2
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

表 20. Cortex-M4 命令 （続き）

ニーモニック	オペランド	概要	フラグ	ページ
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	112 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	—	111 ページの 3.6.2
SSAT	Rd、#n、Rm、{shift #s}	符号付き飽和	Q	126 ページの 3.7.1
SSAT16	Rd、#n、Rm	16 ビット符号付き飽和	Q	127 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	98 ページの 3.5.14
SSUB16	{Rd}、Rn、Rm	16 ビット符号付き減算	—	97 ページの 3.5.13
SSUB8	{Rd}、Rn、Rm	8 ビット符号付き減算	—	97 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	—	76 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	—	76 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	—	76 ページの 3.4.6
STR	Rt、[Rn、#offset]	レジスタ･ストア（ワード）	—	68 ページの 3.4
STRB、STRBT	Rt、[Rn、#offset]	レジスタ･ストア（バイト）	—	68 ページの 3.4
STRD	Rt、Rt2、[Rn、#offset]	レジスタ･ストア（ダブルワード）	—	70 ページの 3.4.2
STREX	Rd、Rt、[Rn、#offset]	排他的レジスタ･ストア	—	79 ページの 3.4.8
STREXB	Rd、Rt、[Rn]	排他的レジスタ･ストア（バイト）	—	79 ページの 3.4.8
STREXH	Rd、Rt、[Rn]	排他的レジスタ･ストア（ハーフワード）	—	79 ページの 3.4.8
STRH、STRHT	Rt、[Rn、#offset]	レジスタ･ストア（ハーフワード）	—	68 ページの 3.4
STRT	Rt、[Rn、#offset]	レジスタ･ストア（ワード）	—	68 ページの 3.4

表 20. Cortex-M4 命令 （続き）

ニーモニック	オペランド	概要	フラグ	ページ
SUB、SUBS	{Rd}、Rn、Op2	減算	N、Z、C、V	83 ページの 3.5.1
SUB、SUBW	{Rd}、Rn、#imm12	減算	N、Z、C、V	83 ページの 3.5.1
SVC	#imm	スーパバイザ・コール	—	190 ページの 3.11.10
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）	—	136 ページの 3.8.2
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 3.9.8
TBH	[Rn、Rm、LSL #1]	テーブル分岐（ハーフワード）	—	147 ページの 3.9.8
TEQ	Rn、Op2	等価テスト	N、Z、C	93 ページの 3.5.9
TST	Rn、Op2	テスト	N、Z、C	93 ページの 3.5.9
UADD16	{Rd}、Rn、Rm	16 ビット符号なし加算	GE	100 ページの 3.5.16
UADD8	{Rd}、Rn、Rm	8 ビット符号なし加算	GE	100 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	101 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	—	102 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	—	102 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	—	103 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	—	103 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	—	104 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	—	104 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	—	112 ページの 3.6.3
UMAAL	RdLo、RdHi、Rn、Rm	符号なし Long 積和 (32 x 32 + 32 +32)、64 ビットの結果	—	111 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	—	111 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	—	111 ページの 3.6.2

表 20. Cortex-M4 命令 （続き）

ニーモニック	オペランド	概要	フラグ	ページ
UQADD16	{Rd}、Rn、Rm	16 ビット符号なし飽和加算	—	132 ページの 3.7.7
UQADD8	{Rd}、Rn、Rm	8 ビット符号なし飽和加算	—	132 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	—	131 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	—	131 ページの 3.7.6
UQSUB16	{Rd}、Rn、Rm	16 ビット符号なし飽和減算	—	132 ページの 3.7.7
UQSUB8	{Rd}、Rn、Rm	8 ビット符号なし飽和減算	—	132 ページの 3.7.7
USAD8	{Rd}、Rn、Rm	符号なし絶対差の和	—	106 ページの 3.5.22
USADA8	{Rd}、Rn、Rm、Ra	絶対差の符号なし和の累算	—	107 ページの 3.5.23
USAT	Rd、#n、Rm、{shift #s}	符号なし飽和	Q	126 ページの 3.7.1
USAT16	Rd、#n、Rm	16 ビット符号なし飽和	Q	127 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	101 ページの 3.5.17
USUB16	{Rd}、Rn、Rm	16 ビット符号なし減算	GE	108 ページの 3.5.24
USUB8	{Rd}、Rn、Rm	8 ビット符号なし減算	GE	108 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）	—	136 ページの 3.8.2
UXTB16	{Rd}、Rm、{ROR #n}	16 ビット値に符号なし拡張（バイト）	—	136 ページの 3.8.2
UXTH	{Rd}、Rm、{ROR #n}	ゼロ拡張（ハーフワード）	—	136 ページの 3.8.2
VABS.F32	Sd、Sm	浮動小数点の絶対値	—	151 ページの 3.10.1
VADD.F32	{Sd}、Sn、Sm	浮動小数点加算	—	152 ページの 3.10.2
VCMP.F32	Sd、<Sm | #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	153 ページの 3.10.3
VCMPE.F32	Sd、<Sm | #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	153 ページの 3.10.3
VCVT.S32.F32	Sd、Sm	浮動小数点数と整数との変換	—	154 ページの 3.10.4
VCVT.S16.F32	Sd、Sd、#fbits	浮動小数点数と固定小数点数との変換	—	154 ページの 3.10.4
VCVTR.S32.F32	Sd、Sm	丸めを使った浮動小数点数と整数との変換	—	154 ページの 3.10.4

表 20. Cortex-M4 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|H>.F32 .F16	Sd、Sm	半精度値から単精度値への変換	—	155 ページの 3.10.5
VCVTT<B|T>. F32.F16	Sd、Sm	単精度レジスタから半精度レジスタへの変換	—	156 ページの 3.10.6
VDIV.F32	{Sd}、Sn、Sm	浮動小数点除算	—	157 ページの 3.10.7
VFMA.F32	{Sd}、Sn、Sm	浮動小数点結合積和	—	158 ページの 3.10.8
VFNMA.F32	{Sd}、Sn、Sm	浮動小数点結合積和の結果を符号反転	—	159 ページの 3.10.9
VFMS.F32	{Sd}、Sn、Sm	浮動小数点結合積差	—	158 ページの 3.10.8
VFNMS.F32	{Sd}、Sn、Sm	浮動小数点結合乗算を減算した結果を符号反転	—	159 ページの 3.10.9
VLDM.F<32|64>	Rn{!}、list	拡張レジスタの多重ロード	—	160 ページの 3.10.10
VLDR.F<32|64>	<Dd|Sd>、[Rn]	メモリからの拡張レジスタ･ロード	—	161 ページの 3.10.11
VLMA.F32	{Sd}、Sn、Sm	浮動小数点積和	—	162 ページの 3.10.12
VLMS.F32	{Sd}、Sn、Sm	浮動小数点積差	—	162 ページの 3.10.12
VMOV.F32	Sd、#imm	浮動小数点のイミディエート値の転送	—	163 ページの 3.10.13
VMOV	Sd、Sm	浮動小数点レジスタ間の転送	—	164 ページの 3.10.14
VMOV	Sn、Rt	ARM コア･レジスタ値を単精度レジスタにコピー	—	168 ページの 3.10.18
VMOV	Sm、Sm1、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	—	167 ページの 3.10.17
VMOV	Dd[x]、Rt	ARM コア･レジスタ値をスカラにコピー	—	165 ページの 3.10.15
VMOV	Rt、Dn[x]	スカラ値を ARM コア･レジスタにコピー	—	166 ページの 3.10.16
VMRS	Rt、FPSCR	FPSCR から ARM コア･レジスタまたは APSR に転送	N、Z、C、V	169 ページの 3.10.19
VMSR	FPSCR、Rt	ARM コア･レジスタから FPSCR に転送	FPSCR	170 ページの 3.10.20
VMUL.F32	{Sd}、Sn、Sm	浮動小数点乗算	—	171 ページの 3.10.21
VNEG.F32	Sd、Sm	浮動小数点数無効	—	172 ページの 3.10.22
VNMLA.F32	Sd、Sn、Sm	浮動小数点積和	—	173 ページの 3.10.23
VNMLS.F32	Sd、Sn、Sm	浮動小数点積差	—	173 ページの 3.10.23
VNMUL	{Sd}、Sn、Sm	浮動小数点乗算	—	173 ページの 3.10.23
VPOP	list	拡張レジスタをポップ	—	174 ページの 3.10.24
VPUSH	list	拡張レジスタをプッシュ	—	175 ページの 3.10.25
VSQRT.F32	Sd、Sm	浮動小数点平方根	—	176 ページの 3.10.26

表 20. Cortex-M4 命令 （続き）

ニーモニック	オペランド	概要	フラグ	ページ
VSTM	Rn{!}、list	浮動小数点レジスタの多重ストア	—	177 ページの 3.10.27
WFE	—	イベント待機	—	191 ページの 3.11.11
WFI	—	割込み待機	—	192 ページの 3.11.12

3.2 CMSIS 組込み関数

ISO/IEC C コードは、一部の Cortex-M4 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 では、ANSI で直接アクセスできない命令を生成するため、表 21 にリストされている組込み関数を提供しています。

表 21. 一部の Cortex-M4 命令を生成するための CMSIS 組込み関数

命令	CMSIS 組込み関数
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多数用意しています（表 22 を参照）。

表 22. 特殊レジスタにアクセスするための CMSIS 組込み関数

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。

- 58 ページのオペランド
- 59 ページのPC または SP を使用した場合の制限事項
- 59 ページのフレキシブル第 2 オペランド
- 60 ページのシフト演算
- 63 ページのアドレスのアライメント
- 64 ページのPC 相対式
- 64 ページの条件付き実行
- 67 ページの命令の幅の選択

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（フレキシブル第 2 オペランドを参照）。

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･レジスタにプログラム･カウンタ（PC）とスタック･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。

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-M4 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
- レジスタと任意に指定できるシフト

定数

Operand2 定数は #constant の形式で指定できます。constant には次の値を指定できます。

- 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

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指定することができます。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

否定が許可されている 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、Rm {, shift} の形式で指定できます。

- Rm は、第 2 オペランドのデータを保持するレジスタです。
- Shift は、Rm に適用される任意のシフト量です。以下の値を指定できます。
  - 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、シフトは 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、直接実行できます。結果はデスティネーション･レジスタに書き込まれます。
- 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（59 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（61 ページの図 13: ASR #3 を参照）。

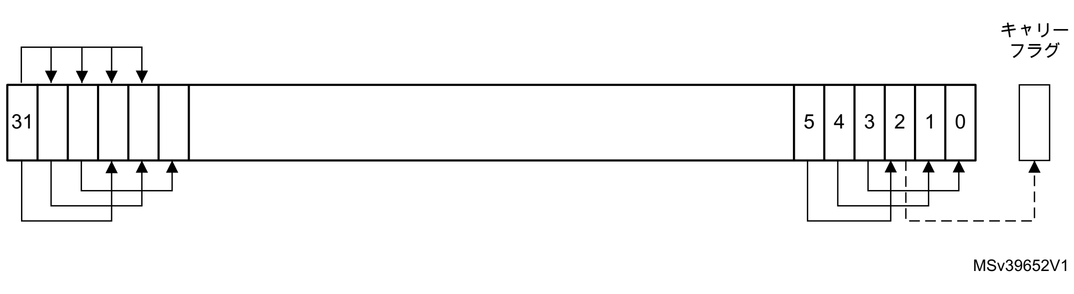
ASR #n 演算を使用すると、レジスタ Rm の値を 2^n で除算し、その結果を負の無限大に丸めることができます。

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。

注： 1 **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。**

2 **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 13. ASR #3



LSR

n ビットの論理右シフトです。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 14 を参照）。

LSR #n 演算を使用すると、レジスタ Rm の値を 2^n で除算できます（値が符号なし整数と見なされる場合）。

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。

注： 1 **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。**

2 **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. LSR #3

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 n ビットを 0 にセットします（図 15: LSL #3 を参照）。

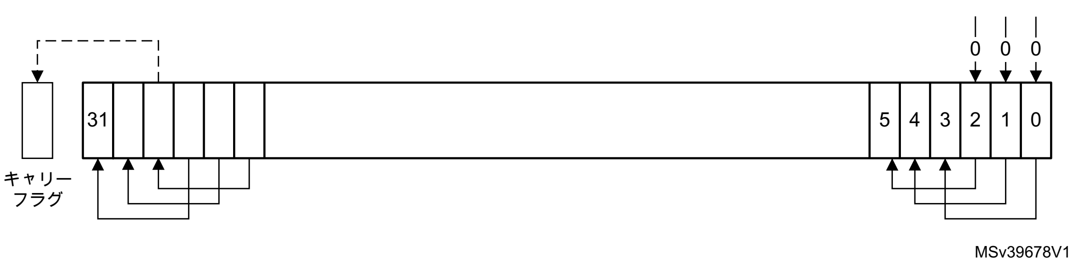
値が符号なしの整数または 2 の補数となる符号付き整数と解釈される場合、LSL #n 演算を使用して、レジスタ Rm の値を 2^n 倍することができます。このとき警告なしでオーバーフローが発生する場合があります。

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 はゼロ以外）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。

注：　1　**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。**

　　　2　**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 15. LSL #3



ROR

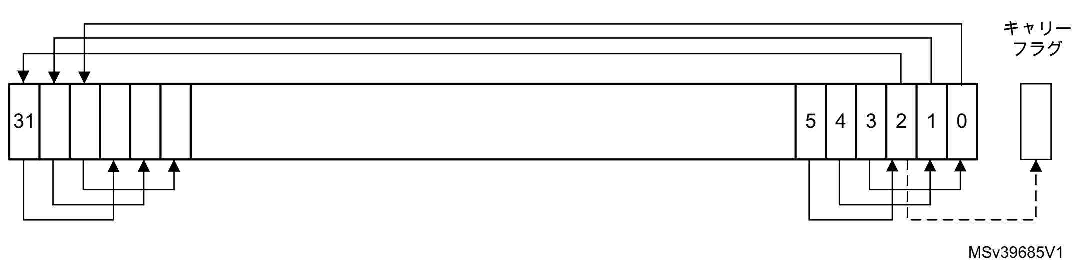
n ビットの左ローテートです。レジスタ Rm の左側 32-n ビットを右に n 桁転送し、結果の右側 32-n ビットが得られます。さらに、レジスタの右 n ビットを結果の左 n ビットに転送します（図 16 を参照）。

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 の最後のビット･ローテーション（ビット [n-1]）に更新されます。

注：　1　**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。**

　　　2　**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 16. ROR #3

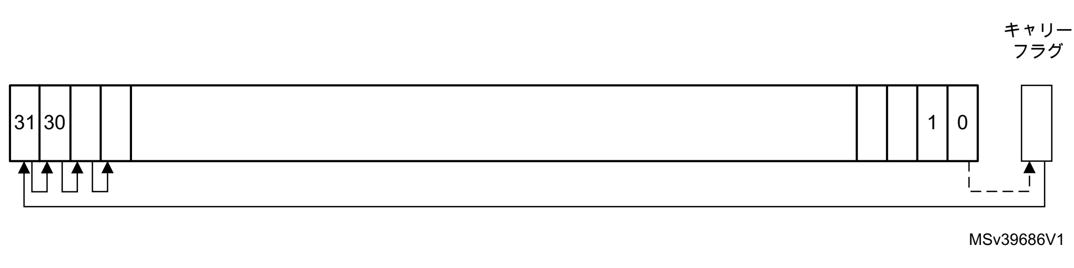


RRX

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（図 17 を参照）。

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 17. RRX #3



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-M4 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、43 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 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防ぐには、設定で UNALIGN_TRP ビットを使用し、レジスタを制御して、すべてのアンアラインド･アクセスをトラップします（230 ページの設定および制御レジスタ（CCR）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、必要に応じて、操作の結果に従って、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（20 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、66 ページの表 23: 条件コードの接尾文字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。

- 条件フラグ
- 66 ページの条件コードの接尾文字

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 : 演算によってオーバーフロ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。

APSR の詳細については、18 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。

キャリーは以下の場合に発生します。

- 加算の結果が 2^{32} 以上の場合。
- 減算の結果が正またはゼロの場合。
- 転送命令または論理命令によるインライン･バレル･シフタ演算の結果として。

オーバーフローは、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 23 に、使用できる条件コードを示します。

IT 命令で条件付き実行を使用して、コード内の分岐命令の数を削減できます。

表 23 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 23. 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	負
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。

具体例 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)。

具体例 1 : 絶対値

```
MOVSR0, R1; R0 = R1、     フラグをセット
IT MI;                    負の場合の IT 命令
RSBMIR0, R1, #0;          負の場合、R0 = -R1
```

具体例 2 : 値の比較と更新に、符号付きの値 R0 および R2 がそれぞれ R1 および R3 より大きい場合に R4 の値を更新する条件付き命令の使用方法を示します。

具体例 2 : 値の比較と更新

```
CMP R0, R1 ;         R0 と R1 を比較し、フラグをセット
ITT GT ;             2 つの GT 条件の IT 命令
CMPGT R2, R3;        'より大きい' の場合、R2 と R3 を比較し、フラグをセット
MOVGT R4, R5 ;       また 'より大きい' である場合、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。このような命令の中には、命令の幅の接尾文字を使用することで、特定の命令サイズを強制できるものがあります。
.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。具体例 3 : 命令の幅の選択に、命令の幅の接尾文字を使用した命令を示します。

具体例 3：命令の幅の選択

```
BCS.W label;          短い分岐でも 32 ビット命令を作成
ADDS.W R0, R0, R1;    16 ビット命令で同じ演算を実行できる場合でも
                      ; 32 ビット命令を作成
```

3.4 メモリ･アクセス命令

表 24 に、メモリ･アクセス命令を示します。

表 24. メモリ･アクセス命令

ニーモニック	概要	参照先
ADR	PC 相対アドレスのロード	69 ページのADR
CLREX	排他をクリア	80 ページのCLREX
LDM{mode}	レジスタの多重ロード	76 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	70 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	72 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	73 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	74 ページのLDR （PC 相対）
LDRD	レジスタ･ロード（デュアル）	70 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	79 ページのLDREX と STREX
POP	レジスタをスタックからポップ	78 ページのPUSH と POP
PUSH	レジスタをスタックにプッシュ	78 ページのPUSH と POP
STM{mode}	レジスタの多重ストア	76 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	70 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	72 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	73 ページのLDR と STR （非特権）
STREX{type}	排他的レジスタ･ストア	79 ページのLDREX と STREX

3.4.1 ADR

PC 相対アドレスのロード。

構文

`ADR{cond} Rd, label`

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- label はPC 相対式です（64 ページのPC 相対式を参照）。

動作

ADR は、イミディエート値を PC に追加することによって、アドレスを決定します。結果をデスティネーション･レジスタに書き込みます。

ADR は PC 相対アドレスであるため、位置独立コードを生成します。

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。

注 : **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（67 ページの命令の幅の選択を参照）。**

制限事項

Rd には SP または PC のどちらも使用できません。

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 は LDR （レジスタ·ロード）または STR （レジスタ·ストア）を指定できます。
- type には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rt はロードまたはストアするレジスタです。
- Rn はメモリ·アドレスのベースとなるレジスタです。
- offset 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 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（63 ページのアドレスのアライメントを参照）。

表 25 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 25. イミディエート、プレインデクス、およびポストインデクスのオフセット範囲

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 をロードする。
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する。
STR R2, [R9,#const-struc]; const-struc は、
                     ; 0 ~ 4095 の範囲内の定数に評価される式。
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
                     ; ストアし、R4 を 4 バイト分インクリメントする。
LDRD R8, R9, [R3, #0x20]; R3 のアドレスの 32 バイト上のワードから
                     ; R8 をロードし、R3 のアドレスの 36 バイト上のワードから
                     ; R9 をロードする。
STRD R0, R1, [R8], #-16; R8 のアドレスに R0 をストアし、R8 のアドレスの
                     ; 4 バイト上のワードに R1 をストアして、
                     ; R8 を 16 バイト分デクリメントする。
```

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 は LDR （レジスタ·ロード）または STR （レジスタ·ストア）を指定できます。
- type には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照)。
- Rt はロードまたはストアするレジスタです。
- Rn はメモリ·アドレスのベースとなるレジスタです。
- Rm 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 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（63 ページのアドレスのアライメントを参照)。

制限事項

これらの命令の場合：

- Rn は PC にはできません。
- Rm には SP または PC のどちらも使用できません。
- 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
                ; ストア
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
                          ; バイト値を読み込み、ワード値に符号拡張して
                          ; 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
                          ; アドレスにストア
```

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 は LDR （レジスタ・ロード）または STR （レジスタ・ストア）を指定できます。
- type には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rt はロードまたはストアするレジスタです。
- Rn はメモリ・アドレスのベースとなるレジスタです。
- offset 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（70 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 :

- Rn は PC にはできません。
- Rt には SP または PC のどちらも使用できません。

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]   ; 非特権アクセスを使用して、R4 の最下位バイトを R7 のアドレスに
                   ; 条件付きでストアする
LDRHT R2, [R2, #8] ; 非特権アクセスを使用して、R2 と 8 の合計に等しいアドレスから
                   ; ハーフワード値を R2 にロードする
```

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、

- type には以下のいずれかを指定します。
  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 ビットに符号拡張）
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rt はロードまたはストアするレジスタです。
- Rt2 は 2 番目にロードまたはストアするレジスタです。
- label はPC 相対式です（64 ページのPC 相対式を参照）。

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（63 ページのアドレスのアライメントを参照）。

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 26 に label と PC 間で可能なオフセットを示します。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（67 ページの命令の幅の選択を参照）。

表 26. label-PC 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

制限事項

これらの命令の場合：

- Rt2 には SP または PC のどちらも使用できません。
- Rt は Rt2 とは異なる値である必要があります。
- 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。
- ワードでのロード命令で Rt が PC のとき、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･データを
                   ; R0 にロードする
LDRSB R7, localdata; localdata とラベル付けされたアドレスからのバイト値を
                   ; ロードし、ワード値に符号拡張して
                   ; R7 に入れる
```

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 は LDM （複数レジスタ･ロード）または STM （複数レジスタ･ストア）を指定できます。
- addr_mode には以下のいずれかを指定します。

  IA : 各アクセスの後で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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rn はメモリ･アドレスのベースとなるレジスタです。
- ! 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 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（77 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が指定されている場合、Rn - 4 * (n) の値が Rn にライトバックされます。

PUSH 命令と POP 命令は、この形式で表すことができます（詳細は、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：

- Rn 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ている場合、reglist に Rn を含めることはできません。

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 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。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。

構文

```
PUSH{cond} reglist
POP{cond} reglist
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- reglist は空でないレジスタ（またはレジスタ範囲）のリストを中かっこで囲んで指定します。レジスタのリストまたは範囲はコンマで区切る必要があります（77 ページの例を参照）。

動作

- PUSH はレジスタ番号の降順にレジスタをスタックにストアし、最大の番号のレジスタが最上位のメモリ･アドレスを使用して、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。
- POP はレジスタ番号の昇順にレジスタをスタックからロードし、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。詳細は、76 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
```

3.4.8 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd 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 はロードまたはストアするレジスタです。
- Rn はメモリ・アドレスのベースとなるレジスタです。
- offset 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（33 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。

注： **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                   ; ロック取得値を初期化する try
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
```

3.4.9 CLREX

排他をクリア。

構文

`CLREX{cond}`

ここで、

cond は任意の条件コードです（64 ページの条件付き実行を参照）。

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。

詳細は、33 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

これら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 27 にデータ処理命令を示します。

表 27. 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	83 ページのADD、ADC、SUB、SBC、および RSB
ADD	加算	83 ページのADD、ADC、SUB、SBC、および RSB
ADDW	加算	83 ページのADD、ADC、SUB、SBC、および RSB
AND	論理積	85 ページのAND、ORR、EOR、BIC、および ORN
ASR	算術右シフト	86 ページのASR、LSL、LSR、ROR、および RRX
BIC	ビット·クリア	85 ページのAND、ORR、EOR、BIC、および ORN
CLZ	先行ゼロ·カウント	87 ページのCLZ
CMN	比較否定	88 ページのCMP および CMN
CMP	比較	88 ページのCMP および CMN
EOR	排他的論理和	85 ページのAND、ORR、EOR、BIC、および ORN
LSL	論理左シフト	86 ページのASR、LSL、LSR、ROR、および RRX
LSR	論理右シフト	86 ページのASR、LSL、LSR、ROR、および RRX
MOV	転送	89 ページのMOV および MVN
MOVT	上位に転送	91 ページのMOVT
MOVW	16 ビット定数を転送	89 ページのMOV および MVN
MVN	転送して否定	89 ページのMOV および MVN
ORN	否定論理和	85 ページのAND、ORR、EOR、BIC、および ORN
ORR	論理和	85 ページのAND、ORR、EOR、BIC、および ORN
RBIT	ビット順序反転	92 ページのREV、REV16、REVSH、および RBIT
REV	ワード内のバイト順序反転	92 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	92 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	92 ページのREV、REV16、REVSH、および RBIT
ROR	右ローテート	86 ページのASR、LSL、LSR、ROR、および RRX
RRX	右ローテート、拡張付き	86 ページのASR、LSL、LSR、ROR、および RRX
RSB	反転減算	83 ページのADD、ADC、SUB、SBC、および RSB
SADD16	16 ビット符号付き加算	93 ページのSADD16 および SADD8
SADD8	8 ビット符号付き加算	93 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	98 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	98 ページのSASX および SSAX
SBC	キャリー付き減算	83 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	94 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	94 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	95 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	95 ページのSHASX および SHSAX

表 27. 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	96 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	96 ページのSHSUB16 および SHSUB8
SSUB16	16 ビット符号付き減算	97 ページのSSUB16 および SSUB8
SSUB8	8 ビット符号付き減算	97 ページのSSUB16 および SSUB8
SUB	減算	83 ページのADD、ADC、SUB、SBC、および RSB
SUBW	減算	83 ページのADD、ADC、SUB、SBC、および RSB
TEQ	等価テスト	93 ページのSADD16 および SADD8
TST	テスト	93 ページのSADD16 および SADD8
UADD16	16 ビット符号なし加算	100 ページのUADD16 および UADD8
UADD8	8 ビット符号なし加算	100 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	101 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	101 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	102 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	102 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	103 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	103 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	104 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	104 ページのUHSUB16 および UHSUB8
USAD8	符号なし絶対差の和	106 ページのUSAD8
USADA8	絶対差の符号なし和の累算	107 ページのUSADA8
USUB16	16 ビット符号なし減算	108 ページのUSUB16 および USUB8
USUB8	8 ビット符号なし減算	108 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 には以下のいずれかを指定します。
  ADD：加算
  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 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（64 ページの条件付き実行を参照）。
- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd 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 は第 1 オペランドを保持するレジスタです。
- Operand2 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、59 ページのフレキシブル第 2 オペランドを参照）。
- Imm12 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（84 ページのマルチワード算術演算の例および 69 ページのADR を参照）。

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 には SP または PC のどちらも使用できません。
- 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、以下の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。
  - S 接尾文字は指定できません。
  - Rm は PC または SP のいずれにもできません。
  - 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、以下の追加制限事項があります。
  - S 接尾文字は指定できません。
  - 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。

注： 1 **加算または減算に PC を使用すると、PC のビット [1:0] が b00 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。**

2 **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 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にのみ実行
```

マルチワード算術演算の例

具体例 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。

具体例 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。具体例 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。

具体例 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 には以下のいずれかを指定します。

  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 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（64 ページの条件付き実行を参照）。
- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn は第 1 オペランドを保持するレジスタです。
- Operand2 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、59 ページのフレキシブル第 2 オペランドを参照）。

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 または PC はどちらも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（59 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

```
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 には以下のいずれかを指定します。
  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 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（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はシフトされる値を保持するレジスタです。
- Rs 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 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

注 : **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、60 ページのシフト演算を参照してください。

制限事項

SP または PC はどちらも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（60 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rm はオペランド･レジスタです。

動作

CLZ 命令は、Rm の値に含まれる先行ゼロの数をカウントし、結果を Rd に返します。ソース･レジスタにビットがセットされていない場合の結果値は 32 となり、ビット [31] がセットされている場合の結果はゼロになります。

制限事項

SP または PC はどちらも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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rn は第 1 オペランドを保持するレジスタです。
- Operand2 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、59 ページのフレキシブル第 2 オペランドを参照）。

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。

CNS 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 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（64 ページの条件付き実行を参照）。
- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Operand2 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、59 ページのフレキシブル第 2 オペランドを参照）。
- imm16 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。

86 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。

注： **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。
- S 接尾文字を指定することはできません。

Rd が MOV 命令の PC の場合 :

- PC に書き込まれた値のビット [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。

注 : **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（59 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 （0xF のビット単位の反転）の値を
                       ; R2 に書き込んで、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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Imm16 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 には SP または PC のどちらも使用できません。

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      ; 0xF123 を R3 の上位ハーフワードに書き込む。
                         ; 下位ハーフワードと APSR は変更されない。
```

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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション・レジスタです。
- Rn 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：以下のいずれかの変換を行います。
  - 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：以下のいずれかの変換を行います。
  - 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：以下のいずれかの変換を行います。
  - 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 または PC はどちらも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 には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はオペランドを保持するレジスタです。
- Rm 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 には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はオペランドを保持するレジスタです。
- Rm 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 には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はオペランドを保持するレジスタです。
- Rn、Rm 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、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、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、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 ; 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、
                   ; 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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はオペランドを保持するレジスタです。
- Rm 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      ; R0 のハーフワードを R1 の対応するハーフワードから減算し、
                    ; 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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はオペランドを保持するレジスタです。
- Rm 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       ; R0 のハーフワードを R1 の対応するハーフワードから減算し、
                    ; 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 には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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下位ハーフワードを第 2 オペランドの符号付き上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込む
                  ; R5 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、
                  ; R0 の下位ハーフワードに書込む
SSAX R7, R3, R2   ; R2 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、
                  ; R7 の上位ハーフワードに書込む
                  ; R3 の下位ハーフワードを R2 の上位ハーフワードに加算し、
```

```
                                ; R7 の下位ハーフワードに書込む
```

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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rn は第 1 オペランドを保持するレジスタです。
- Operand2 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、59 ページのフレキシブル第 2 オペランドを参照）。

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 または PC はどちらも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（59 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実行
                    ; APSR が更新されるが、結果は破棄
TEQEQ R10, R9       ; R10 の値が R9 の値と等しいか条件付きテストを実行
                    ; APSR が更新されるが、結果は破棄
```

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 には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はオペランドを保持する 1 番目のレジスタです。
- Rm はオペランドを保持する 2 番目のレジスタです。

演算

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。

UADD8 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       ; 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
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 には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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
                   ; R5 の下位ハーフワードを R0 の上位ハーフワードから減算し、
                   ; R0 の下位ハーフワードに書き込む
USAX R7, R3, R2    ; R2 の上位ハーフワードを R3 の下位ハーフワードから減算し、
                   ; 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 には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn は第 1 オペランドを保持するレジスタです。
- Rm 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。

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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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。

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 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    ; 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、
                      ; 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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はオペランドを保持する 1 番目のレジスタです。
- Rm 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。

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. 符号なしのハーフワードの各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   ; R0 のハーフワードを R1 の対応するハーフワードから減算し、
                 ; 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 ; R5 のバイトを R0 の対応するバイトから減算し、
                   ; 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> は標準のアセンブラ構文フィールドです。
- <Rd> はデスティネーション･レジスタです。
- <Rn> は第 1 オペランド･レジスタです。
- <Rm> 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 オペランド･レジスタまたは第 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn は第 1 オペランド･レジスタです。
- Rm 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 または PC はどちらも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        ; R0 の各バイトを R4 の対応するバイトから減算し、
                        ; その差分を加算し、R1 に書き込む
USAD8  R0, R5           ; R5 のバイトを R0 の対応するバイトから減算し、
                        ; その差分を加算し、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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn は第 1 オペランド･レジスタです。
- Rm は第 2 オペランド･レジスタです。
- Ra 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  ; R0 のバイトを R1 の対応するバイトから減算し、
                   ; 差分を加算し、R6 の値を加算し、R1 に書き込む
USADA8  R4, R0, R5, R2 ; R5 のバイトを R0 の対応するバイトから減算し、
                   ; 差分を加算し、R2 の値を加算し、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 には以下のいずれかを指定します。

  USUB16：16 ビット符号なし減算

  USUB8：8 ビット符号なし減算
- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はオペランドを保持するレジスタです。
- Rm 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 または PC はどちらも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    ; R0 のハーフワードを R1 の対応するハーフワードから減算し、
                 ; R1 の対応するハーフワードに書き込みます
USUB8  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
```

3.6 乗算および除算命令

表 28 は乗算および除算命令を示しています。

表 28. 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	110 ページのMUL、MLA、および MLS
MLS	積差、32 ビットの結果	110 ページのMUL、MLA、および MLS
MUL	乗算、32 ビットの結果	110 ページのMUL、MLA、および MLS
SDIV	符号付き除算	124 ページのSDIV および UDIV
SMLA[B,T]	符号付き積和（ハーフワード）	112 ページのSMLA および SMLAW
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	115 ページのSMLAL および SMLALD
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）	112 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	120 ページのSMMUL
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	112 ページのSMLA および SMLAW
UMAAL	符号なし Long 積和累算（32 x 32 + 32 + 32）、64 ビットの結果	111 ページのUMULL、UMAAL および UMLAL
UMLAL	符号なし積和（32 x 32 + 64)、64 ビットの結果	111 ページのUMULL、UMAAL および UMLAL
UMULL	符号なし乗算（32 x 32)、64 ビットの結果	111 ページのUMULL、UMAAL および 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- S 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（64 ページの条件付き実行を参照）。
- Rd 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 は乗算される値を保持するレジスタです。
- Ra 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。

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 または PC はどちらも使用しないでください。

S 接尾文字を MUL 命令で指定する場合 :

- 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 接尾文字は使用できません。

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 および 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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- RdHi、RdLo はデスティネーション･レジスタです。これらは、累算値も保持します。
- Rn、Rm 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

UMULL 命令：

1. 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。
2. 結果の下位 32 ビットを RdLo に書き込みます。
3. 結果の上位 32 ビットを RdHi に書き込みます。

UMAAL 命令：

1. 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。
2. 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。
3. 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。
4. 結果の上位 32 ビットを RdHi に書き込みます。
5. 結果の下位 32 ビットを RdLo に書き込みます。

UMLAL 命令：

1. 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。
2. 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。
3. 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 または PC はどちらも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 を加算して、
```

```
                          ; 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 には以下のいずれかを指定します。

  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 Rm レジスタ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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd 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 は乗算される値を保持するレジスタです。
- Ra 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：

1. 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。
2. Ra の値を 32 ビットの乗算結果に加算します。
3. 積和の結果を Rd に書き込みます。
4. 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：

1. Rn の 32 ビットの符号付き値を以下と乗算します。
   a) Rm の符号付き上位ハーフワード（T 接尾文字付き）。
   b) Rm の符号付き下位ハーフワード（B 接尾文字付き）。
2. 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。
3. 積和の結果を Rd に書き込みます。
4. 48 ビットの乗算結果の下位 16 ビットは無視されます。
5. 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 または 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と R4 の上位ハーフワードを乗算し、
                         ; 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 をその結果に加算して、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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn 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 は第 2 オペランド･レジスタです。
- Ra 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 :

1. 以下のいずれかを実行します。
   a) 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。
   b) 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. 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。
3. 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 または PC はどちらも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と
                        ; R1 の対応するハーフワードを乗算し、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 には以下のいずれかを指定します。
  - SMLAL：符号付き Long 積和。
  - SMLAL：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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- RdHi、RdLo 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 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：

1. 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。
2. 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。
3. 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：

1. 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。
2. 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。
3. 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。
- 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 または PC はどちらも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と R7 の上位ハーフワードを乗算し、
                        ; 32 ビットに符号拡張し、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、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と 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と R1 の下位
                        ; ハーフワードを、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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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 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 :

1. 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。
2. 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。
3. 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。
4. 符号付き累算値を減算結果に加算します。
5. 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 :

1. 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。
2. 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。
3. 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。
4. RdHi と RdLo の 64 ビット値を減算結果に加算します。
5. 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 または PC はどちらも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、
                        ; 2 つ目の乗算結果を 1 つ目の結果から減算し、
                        ; 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、
                        ; 2 つ目の乗算結果を 1 つ目の結果から減算し、
                        ; 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、
                        ; 2 つ目の乗算結果を 1 つ目の結果から減算し、
                        ; R6:R3 を加算して、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、
                        ; 2 つ目の乗算結果を 1 つ目の結果から減算し、
                        ; 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 には以下のいずれかを指定します。

  SMMLA：符号付き上位ワードの積和
  SMMLS：符号付き上位ワードの積差
- R 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、0x80000000 が乗算結果に加算されてから、上位ワードが抽出されます。
- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション・レジスタです。
- Rn、Rm 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 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。

1. Rn と Rm の値を乗算します。
2. 任意で、0x80000000 を加算して、値を丸めます。
3. 結果の上位 32 ビットを抽出します。
4. Ra の値を抽出した符号付き値に加算します。
5. 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。

1. Rn と Rm の値を乗算します。
2. 任意で、0x80000000 を加算して、値を丸めます。
3. 結果の上位 32 ビットを抽出します。
4. 結果から抽出した値を Ra の値から減算します。
5. 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 または PC はどちらも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、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、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 には以下のいずれかを指定します。

  SMMUL : 符号付き上位ワード乗算。

  R : 丸め誤差フラグ。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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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 :

1. Rn と Rm の値を乗算します。
2. 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。
3. 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 または PC はどちらも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、
                       ; 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、
                       ; 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 には以下のいずれかを指定します。

  SMUAD : デュアル符号付き積和。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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 は第 1 オペランドの値と第 2 オペランドの値を 2 つの符号付きハーフワードとして解釈します。

1. 第 2 オペランドのハーフワードを任意にローテートします。
2. 符号付き 16 x 16 ビットの乗算を 2 つ実行します。
3. 2 つの乗算結果を加算します。
4. 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 は第 1 オペランドの値と第 2 オペランドの値を 2の補数の符号付き整数として解釈します。

1. 第 2 オペランドのハーフワードを任意にローテートします。
2. 符号付き 16 x 16 ビットの乗算を 2 つ実行します。
3. 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。
4. 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 または PC はどちらも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
                   ; 乗算結果を加算して、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
                   ; 乗算結果を加算して、R3 に書き込む
```

```
SMUSD   R3, R6, R2 ; 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の
                   ; 乗算結果を減算して、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
                   ; 乗算結果を減算して、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
```

- op には以下のいずれかを指定します。

  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 Rm レジスタ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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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：

1. 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。
2. 32 ビットの乗算結果を 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：

1. 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。
2. 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

SP または PC はどちらも使用しないでください。

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 ; R5 と R3 の上位ハーフワードを乗算し、
                   ; 上位 32 ビットを抽出して、R4 に書き込む
SMULWB  R4, R5, R3 ; R5 と R3 の下位ハーフワードを乗算し、
                   ; 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 には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- RdHi、RdLo 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 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 または PC はどちらも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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd 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 は除算される値を保持するレジスタです。
- Rn は除数を保持するレジスタです。

動作

SDIV は、Rn の値を Rm の値で除算する、符号付き整数除算です。

UDIV は、Rn の値を Rm の値で除算する、符号なし整数除算です。

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 または PC はどちらも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

このセクションでは、飽和命令について説明します。

表 29. 飽和命令

ニーモニック	概要	参照先
SSAT	符号付き飽和	126 ページのSSAT および USAT
SSAT16	符号付きハーフワード飽和	127 ページのSSAT16 および USAT16
USAT	符号なし飽和	126 ページのSSAT および USAT
USAT16	符号なしハーフワード飽和	127 ページのSSAT16 および USAT16
QADD	飽和加算	128 ページのQADD および QSUB
QSUB	飽和減算	128 ページのQADD および QSUB
QSUB16	16 ビット飽和減算	128 ページのQADD および QSUB
QASX	飽和加算および減算、交換付き	129 ページのQASX および QSAX
QSAX	飽和減算および加算、交換付き	129 ページのQASX および QSAX
QDADD	飽和倍演算および加算	130 ページのQDADD および QDSUB
QDSUB	飽和倍演算および減算	130 ページのQDADD および QDSUB
UQADD16	16 ビット符号なし飽和加算	132 ページのUQADD および UQSUB
UQADD8	8 ビット符号なし飽和加算	132 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	131 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	131 ページのUQASX および UQSAX
UQSUB16	16 ビット符号なし飽和減算	132 ページのUQADD および UQSUB
UQSUB8	8 ビット符号なし飽和減算	132 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} 未満の場合、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 命令を使用します（187 ページのMSR を参照）。

Q フラグの状態を読み出すには、MRS 命令を使用します（186 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 には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- n は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 は飽和する値を保持するレジスタです。
- shift #s 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：s の範囲は 1 ~ 31 です。
  LSL #s：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、符号付き範囲内 $-2^{n-1} \leq x \leq 2^{n-1}-1$ で飽和します。

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 $0 \leq x \leq 2^{n-1}$ 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 ビット左に論理シフトし、
                              ; その結果を符号付き 16 ビット値として飽和して、
                              ; 結果を R7 にライトバック
USATNE  R0, #7, R5            ; R5 の値を符号なし 7 ビット値として
                              ; 条件付きで飽和し、結果を 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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- n は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。
- Rn 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 の上位および下位のハーフワードを 9 ビット値として
                            ; 飽和し、R7 の対応するハーフワードに
                            ; 書き込む
USAT16NE  R0, #13, R5       ; R5 の各ハーフワードを 13 ビット値として
                            ; 条件付きで飽和し、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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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 $-2^{n-1} \leq x \leq 2^{n-1}-1$ 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。そうでない場合は、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 命令を使用します（187 ページのMSR を参照）。
Q フラグの状態を読み出すには、MRS 命令を使用します（186 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
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 ; R3 のハーフワードを R2 の対応するハーフワードから減算し、
```

```
                    ; 16 ビットに飽和して、R4 の対応するハーフワードに
                    ; 書き込む
QSUB8   R4, R2, R5 ; R5 のバイトを R2 の対応するバイトから減算し、
                    ; 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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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、
5. $-2^{15} \le x \le 2^{15} - 1$（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、
4. $-2^{15} \le x \le 2^{15} - 1$（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。
5. 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    ; 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
                   ; 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 には以下のいずれかを指定します。
  QDADD：飽和倍演算と加算。
  QDSUB：飽和倍演算と減算。
- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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：

1. 第 2 オペランドの値を 2 倍にします。
2. 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。
3. 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：

1. 第 2 オペランドの値を 2 倍にします。
2. 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。
3. 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、$–2^{31} \leq x \leq 2^{31}– 1$ 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 を加算し、
                      ; 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 32 ビットに飽和した値を 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、

- op には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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、
4. $0 \le x \le 2^{16} - 1$（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。
5. 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
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 には以下のいずれかを指定します。

  UQADD8：4 つの 8 ビット符号なし整数の加算。

  UQADD16：2 つの 16 ビット符号なし整数の加算。

  UDSUB8：4 つの 8 ビット符号なし整数の減算。

  UQSUB16：2 つの 16 ビット符号なし整数の減算。
- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn、Rm 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：

1. 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。
2. 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：

1. 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。
2. 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：

1. 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。
2. 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：

1. 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。
2. デスティネーション･レジスタに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 ; R0 のハーフワードを R3 の対応するハーフワードから減算し、
                   ; 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 ; R6 のバイトを R5 の対応するバイトから減算し、
                   ; 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 30 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 30. 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）	136 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	136 ページのSXT および UXT
SXTH	符号拡張（ハーフワード）	136 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）	136 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	136 ページのSXT および UXT
UXTH	ゼロ拡張（ハーフワード）	136 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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn は第 1 オペランド･レジスタです。
- Rm 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 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 : シフト長が 1 ~ 32 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 および 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み、
                           ; 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  ; 1 ビット右にシフトした R2 を 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 には以下のいずれかを指定します。

  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション・レジスタです。
- Rm は拡張する値を保持するレジスタです。
- ROR #n は、以下のいずれかを指定できます。

  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 には以下のいずれかを指定します。

  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。
  SXTAB : 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。
  UXTAB : 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn は第 1 オペランド･レジスタです。
- Rm 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 は、以下のいずれかを指定できます。

  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 16 ビット右に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、R8 を加算して、
                               ; 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 ビットフィールド命令

表 31 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 31. 隣接するビット･セットで動作する命令

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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はソース･レジスタです。
- lsb 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 の範囲とする必要があります。
- width 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 ~ 11 に置換
```

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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rn はソース･レジスタです。
- lsb 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 の範囲とする必要があります。
- width 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 からビット 20 ~ 23 （4 ビット）を抽出し、
                     ; 32 ビットに符号拡張して、その結果を R0 に書き込む
UBFX R8, R11, #9, #10; R11 からビット 9 ~ 18（10 ビット）を抽出し、
                     ; 32 ビットにゼロ拡張して、その結果を 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

```
SXTextend{cond} {Rd,} Rm {, ROR #n}
UXTextend{cond} {Rd}, Rm {, ROR #n}
```

ここで、

- extend には以下のいずれかを指定します。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- Rm は拡張する値を保持するレジスタです。
- ROR #n には以下のいずれかを指定します。
  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、
                           ; その結果を R4 に書き込む
UXTB   R3, R10             ; R10 の値の最下位バイトを抽出し、
                           ; ゼロ拡張して、その結果を R3 に書き込む
```

3.9.4 分岐命令と制御命令

表 32 は分岐命令と制御命令を示しています。

表 32. 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

3.9.5 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 は分岐（イミディエート）です。
- BL はリンク付き分岐（イミディエート）です。
- BX は間接分岐（レジスタ）です。
- BLX はリンク付き間接分岐（レジスタ）です。
- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- label はPC 相対式です。64 ページのPC 相対式を参照してください。
- Rm 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 33 は、各種分岐命令の範囲を示しています。

表 33. 分岐範囲

命令	分岐範囲
B label	-16MB ~ +16MB
Bcond label （IT ブロック外）	-1MB ~ +1MB
Bcond 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。67 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。

Bcond 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
                ; （呼出）
```

3.9.6 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 はオペランドを保持するレジスタです。
- label 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 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
```

3.9.7 IT

If-Then 条件付き命令。

構文

`IT{x{y{z}}} cond`

ここで、

- x 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 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 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 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。

a) 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。
- Bcond 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。Bcond 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
                      ; 使用可能

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
```

3.9.8 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 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 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 はベースとして
                                ; 使用される
BranchTable_H
    DCI     ((CaseA - BranchTable_H)/2)   ; CaseA オフセットの計算
    DCI     ((CaseB - BranchTable_H)/2)   ; CaseB オフセットの計算
    DCI     ((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
```

3.10 浮動小数点命令

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、256 ページのFPU の有効化を参照してください。

表 34. 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	151 ページのVABS
VADD	浮動小数点加算	152 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	153 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	153 ページのVCMP、VCMPE
VCVT	浮動小数点数と整数との変換	154 ページのVCVT、VCVTR（浮動小数点数と整数間）
VCVT	浮動小数点数と固定小数点数との変換	155 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	154 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	156 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	156 ページのVCVTB、VCVTT
VDIV	浮動小数点除算	157 ページのVDIV
VFMA	結合浮動小数点積和	158 ページのVFMA、VFMS
VFNMA	浮動小数点結合積和の結果を符号反転	159 ページのVFNMA、VFNMS
VFMS	浮動小数点結合積差	158 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	159 ページのVFNMA、VFNMS
VLDM	拡張レジスタの多重ロード	160 ページのVLDM
VLDR	メモリからの拡張レジスタ･ロード	161 ページのVLDR
VLMA	浮動小数点積和	162 ページのVLMA、VLMS
VLMS	浮動小数点積差	162 ページのVLMA、VLMS
VMOV	浮動小数点のイミディエート値の転送	163 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	164 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	165 ページのVMOV （スカラから ARM コア･レジスタへ）
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	166 ページのVMOV （ARM コア･レジスタと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	167 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	スカラ値を ARM コア･レジスタにコピー	168 ページのVMOV （ARM コア･レジスタからスカラへ）
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	169 ページのVMRS

表 34. 浮動小数点命令

ニーモニック	概要	参照先
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	170 ページのVMSR
VMUL	浮動小数点乗算	171 ページのVMUL
VNEG	浮動小数点数無効	172 ページのVNEG
VNMLA	浮動小数点積和	173 ページのVNMLA、VNMLS、VNMUL
VNMLS	浮動小数点積差	173 ページのVNMLA、VNMLS、VNMUL
VNMUL	浮動小数点乗算	173 ページのVNMLA、VNMLS、VNMUL
VPOP	拡張レジスタをポップ	174 ページのVPOP
VPUSH	拡張レジスタをプッシュ	175 ページのVPUSH
VSQRT	浮動小数点平方根	176 ページのVSQRT
VSTM	拡張レジスタの多重ストア	177 ページのVSTM
VSTR	拡張レジスタをメモリにストア	178 ページのVSTR
VSUB	浮動小数点減算	179 ページのVSUB

3.10.1 VABS

浮動小数点の絶対値。

構文

`VABS{cond}.F32 Sd, Sm`

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照)。
- Sd、Sm 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

この浮動小数点命令は符号ビットをクリアします。

例

```
VABS.F32 S4, S6
```

3.10.2 VADD

浮動小数点加算

構文

```
VADD{cond}.F32 {Sd,} Sn, Sm
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd はデスティネーションの浮動小数点値です。
- Sn、Sm 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
```

3.10.3 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}.F32 Sd, Sm
VCMP{E}{cond}.F32 Sd, #0.0
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- E 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 は比較元の浮動小数点オペランドです。
- Sm 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. 以下を比較します。

   2 つの浮動小数点レジスタ。

   1 つの浮動小数点レジスタとゼロ。

1. 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（169 ページのVMRS を参照）。

例

```
VCMP.F32    S4, #0.0
VCMP.F32    S4, S2
```

3.10.4 VCVT、VCVTR（浮動小数点数と整数間）

レジスタの値を浮動小数点数から 32 ビット整数に変換。

構文

```
VCVT{R}{cond}.Tm.F32 Sd, Sm
VCVT{cond}.F32.Tm Sd, Sm
```

ここで、

- R は以下のようになります。
  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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Tm 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、Sm 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. 以下のいずれかの処理を実行します。
- 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。

3.10.5 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.F32 Sd, Sd, #fbits
VCVT{cond}.F32.Td Sd, Sd, #fbits
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Td 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 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 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

これらの命令の場合：

以下のいずれかの処理を実行します。

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。

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。

3.10.6 VCVTB、VCVTT

半精度値と単精度値間の変換。

構文

```
VCVT{y}{cond}.F32.F16 Sd, Sm
VCVT{y}{cond}.F16.F32 Sd, Sm
```

ここで、

- y 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd はデスティネーション･レジスタです。
- Sm はオペランド･レジスタです。

動作

この命令に .F16.F32 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変換します。
2. 結果を単精度レジスタに書き込みます。

この命令に .F32.F16 接尾文字が指定されている場合 :

1. 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。

3.10.7 VDIV

浮動小数点値を除算します。

構文

`VDIV{cond}.F32 {Sd,} Sn, Sm`

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd はデスティネーション･レジスタです。
- Sn、Sm 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。

3.10.8 VFMA、VFMS

結合浮動小数点積和および積差。

構文

```
VFMA{cond}.F32 {Sd,} Sn, Sm
VFMS{cond}.F32 {Sd,} Sn, Sm
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd はデスティネーション･レジスタです。
- Sn、Sm 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。
3. 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。
5. 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。

3.10.9 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}.F32 {Sd,} Sn, Sm
VFNMS{cond}.F32 {Sd,} Sn, Sm
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd はデスティネーション･レジスタです。
- Sn、Sm 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。

3.10.10 VLDM

浮動小数点多重ロード

構文

`VLDM{mode}{cond}{.size} Rn{!}, list`

ここで、

- mode 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 Rn で指定されているアドレスの直前で終了します。
- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Size は任意のデータ･サイズ指定子です。
- Rn はベース･レジスタです。SP を使用できます。
- ! 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 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。

3.10.11 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}{.64} Dd, [Rn{#imm}]
VLDR{cond}{.64} Dd, label
VLDR{cond}{.64} Dd, [PC, #imm}]
VLDR{cond}{.32} Sd, [Rn {, #imm}]
VLDR{cond}{.32} Sd, label
VLDR{cond}{.32} Sd, [PC, #imm]
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- 64, 32 は任意のデータ･サイズ指定子です。
- Dd 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 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 はベース･レジスタです。SP を使用できます。
- imm 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 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。

3.10.12 VLMA、VLMS

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

```
VLMA{cond}.F32 Sd, Sn, Sm
VLMS{cond}.F32 Sd, Sn, Sm
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd はデスティネーションの浮動小数点値です。
- Sn、Sm 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。

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。

3.10.13 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}.F32 Sd, #imm`

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd は分岐のデスティネーションです。
- imm 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。

3.10.14 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}.F64 Dd, Dm
VMOV{cond}.F32 Sd, Sm
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Dd 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 はダブルワード演算のソース･レジスタです。
- Sd 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 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。

3.10.15 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rt はデスティネーション ARM コア･レジスタです。
- Dn は 64 ビット･ダブルワード･レジスタです。
- x はダブルワード･レジスタのどちらの半分を使用するかを指定します。
  x が 0 の場合は下位半分を使用し、x が 1 の場合は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。

3.10.16 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sn は単精度浮動小数点レジスタです。
- Rt 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。

3.10.17 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, Sm
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sm は 1 番目の単精度レジスタです。
- Sm1 は 2 番目の単精度レジスタです（Sm の次の単精度レジスタ）。
- Rt は Sm との間で転送を行う ARM コア･レジスタです。
- Rt2 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。

1. 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. 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。

3.10.18 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- 32 は任意のデータ･サイズ指定子です。
- Dd[x] 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 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。

3.10.19 VMRS

浮動小数点システム･レジスタから ARM コア･レジスタに転送します。

構文

```
VMRS{cond} Rt, FPSCR
VMRS{cond} APSR_nzcv, FPSCR
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rt 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 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。

1. FPSCR の値を汎用レジスタにコピーします。
2. 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。

3.10.20 VMSR

ARM コア･レジスタから 浮動小数点システム･レジスタに転送します。

構文

```
VMSR{cond} FPSCR, Rt
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rt 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。詳細は、254 ページの浮動小数点ステータス制御レジスタ（FPSCR）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。

3.10.21 VMUL

浮動小数点乗算。

構文

```
VMUL{cond}.F32 {Sd,} Sn, Sm
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照)。
- Sd はデスティネーションの浮動小数点値です。
- Sn、Sm 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。

3.10.22 VNEG

浮動小数点符号反転。

構文

`VNEG{cond}.F32 Sd, Sm`

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd はデスティネーションの浮動小数点値です。
- Sm 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。
3. 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。

3.10.23 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}.F32 Sd, Sn, Sm
VNMLS{cond}.F32 Sd, Sn, Sm
VNMUL{cond}.F32 {Sd,} Sn, Sm
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd はデスティネーションの浮動小数点値です。
- Sn、Sm はオペランドの浮動小数点値です。

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。

3.10.24 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- size 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 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。

3.10.25 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- size 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 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。

3.10.26 VSQRT

浮動小数点平方根。

構文

`VSQRT{cond}.F32 Sd, Sm`

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd はデスティネーションの浮動小数点値です。
- Sm 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. 浮動小数点レジスタの値の平方根を計算します。
2. 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。

3.10.27 VSTM

浮動小数点多重ストア。

構文

`VSTM{mode}{cond}{.size} Rn{!}, list`

ここで、

- mode 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- size 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 はベース･レジスタです。SP を使用できます。
- ! 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 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。

3.10.28 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- 32, 64 は任意のデータ･サイズ指定子です。
- Sd はシングルワード･ストアのソース･レジスタです。
- Dd はダブルワード･ストアのソース･レジスタです。
- Rn はベース･レジスタです。SP を使用できます。
- imm 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 は省略でき、その場合はオフセットが +0 であることを意味します。

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。

3.10.29 VSUB

浮動小数点減算。

構文

`VSUB{cond}.F32 {Sd,} Sn, Sm`

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- Sd はデスティネーションの浮動小数点値です。
- Sn、Sm 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 その他の命令

表 35 に、Cortex-M4 のその他の命令を示します。

表 35. 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	181 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	182 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	182 ページのCPS
DMB	データ･メモリ･バリア	183 ページのDMB
DSB	データ同期バリア	184 ページのDSB
ISB	命令同期バリア	185 ページのISB
MRS	特殊レジスタからレジスタへの転送	186 ページのMRS
MSR	レジスタから特殊レジスタへの転送	187 ページのMSR
NOP	何もしない	188 ページのNOP
SEV	イベント送信	189 ページのSEV
SVC	スーパバイザ･コール	190 ページのSVC
WFE	イベント待機	191 ページのWFE
WFI	割込み待機	192 ページのWFI

3.11.1 BKPT

ブレークポイント。

構文

`BKPT #imm`

ここで、

- imm 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 0xAB    ; イミディエート値が 0xAB に設定されているブレークポイント
                 ; （デバッガは PC を使用して配置することでイミディエート値を抽出可能）
```

3.11.2 CPS

プロセッサ状態を変更します。

構文

`CPSeffect iflags`

ここで、

- effect は次のいずれかを指定できます。

  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 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、22 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。

- CPS は特権ソフトウェアでのみ使用でき、非特権ソフトウェアで使用した場合は作用しません。
- 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）
```

3.11.3 DMB

データ・メモリ・バリア。

構文

`DMB{cond}`

ここで、cond は任意の条件コードです（64 ページの条件付き実行を参照)。

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
```

3.11.4 DSB

データ同期バリア。

構文

`DSB{cond}`

ここで、cond は任意の条件コードです（64 ページの条件付き実行を参照）。

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
```

3.11.5 ISB

命令同期バリア。

構文

`ISB{cond}`

ここで、cond は任意の条件コードです（64 ページの条件付き実行を参照）。

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
```

3.11.6 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rd はデスティネーション･レジスタです。
- spec_reg は、APSR、IPSR、EPSR、IEPSR、IAPSR、EAPSR、PSR、MSP、PSP、PRIMASK、BASEPRI、BASEPRI、MAX、FAULTMASK、または CONTROL のいずれかにできます。

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。187 ページのMSR を参照してください。
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。

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
```

3.11.7 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 は任意の条件コードです（64 ページの条件付き実行を参照）。
- Rn はソース･レジスタです。
- spec_reg は、APSR、IPSR、EPSR、IEPSR、IAPSR、EAPSR、PSR、MSP、PSP、PRIMASK、BASEPRI、BASEPRI、MAX、FAULTMASK、または CONTROL のいずれかにできます。

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（20 ページの表 4: APSR のビット定義を参照）。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

186 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
```

3.11.8 NOP

何もしない。

構文

`NOP{cond}`

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。

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
```

3.11.9 SEV

イベントを送信します。

構文

`SEV{cond}`

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。

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（46 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
```

3.11.10 SVC

スーパバイザ·コール。

構文

```
SVC{cond} #imm
```

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。
- imm は 0 ~ 255 （8 ビット値）の範囲の整数に相当する式です。

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    ; スーパバイザ·コール（SVC ハンドラはスタックされている
              ;PC を配置することでイミディエート値を抽出可能）
```

3.11.11 WFE

イベント待機。WFE はヒント命令です。

構文

`WFE{cond}`

ここで、cond は任意の条件コードです（64 ページの条件付き実行を参照）。

動作

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、46 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
```

3.11.12 WFI

割込み待機。

構文

`WFI{cond}`

ここで、

- cond は任意の条件コードです（64 ページの条件付き実行を参照）。

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。

- 例外
- デバッグ･エントリ要求（デバッグが有効かどうかは無関係）

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 コア·ペリフェラル

4.1 STM32 Cortex-M4 のコア·ペリフェラル

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 36. STM32 のコア·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
0xE000E010-0xE000E01F	システム·タイマ	250 ページの表 54
0xE000E100-0xE000E4EF	ネスト化されたベクタ割込みコントローラ	219 ページの表 48
0xE000ED00-0xE000ED3F	システム制御ブロック	243 ページの表 52
0xE000ED88-0xE000ED8B	浮動小数点ユニット·コプロセッサ·アクセス制御	251 ページの表 55
0xE000ED90-0xE000EDB8	メモリ保護ユニット	206 ページの表 43
0xE000EF00-0xE000EF03	ネスト化されたベクタ割込みコントローラ	219 ページの表 48
0xE000EF30-0xE000EF44	浮動小数点ユニット	251 ページの表 55

レジスタの説明では、次のように表記されています。

- 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。

4.2 メモリ保護ユニット（MPU）

このセクションでは、一部の STM32 マイクロコントローラに実装されているメモリ保護ユニット（MPU）について説明します。使用する STM 32 タイプに MPU が存在するかどうかを調べるには、対応するデバイスのデータシートを参照してください。

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。以下の機能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-M4 MPU は次を定義します。

- 0 ~ 7 の 8 つの個別メモリ領域
- 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-M4 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（28 ページのセクション 2.2.1: メモリの領域、タイプ、および属性を参照）。

表 37 に、使用可能な MPU 領域属性を示します。

表 37. 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 ライトスルー·キャッシュ可能 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 ライトスルー·キャッシュ可能 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。

4.2.1 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。MPU_RASR レジスタの 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 38 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 38. 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
b000	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	
b001	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	
b010	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		-
b1BB	A	A	0	ノーマル	共有不可	キャッシュ可能メモリ(2)、BB = 外部ポリシー、AA = 内部ポリシー .
			1		共有可能	

1. MPU はこのビットの値を無視します。
2. AA ビットと BB ビットのエンコードについては、表 39 を参照してください。

表 39 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 39. 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 40 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 40. AP エンコード

AP[2:0]	特権アクセス許可	非特権アクセス許可	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
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

4.2.2 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（25 ページのセクション 2.1.4: 例外と割込みを参照）。MMFSR は、フォールトの原因を示します。詳細は、241 ページのセクション 4.4.15: メモリ管理フォールト·アドレス·レジスタ（MMFAR）を参照してください。

4.2.3 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 と MPU_RASR の各エイリアスを使用して、STM 命令で最大 4 つの領域を同時にプログラムできます。

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]           ; 領域サイズと有効化
STRH R3, [R0, #0xA]           ; 領域属性
```

変更する領域をすでに有効にしている場合、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。次に例を示します。

```
                              ; 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

ただし、例外ハンドラに入ることよってMPU セットアップ･プロセスが開始する場合、またはその後に例外から復帰する場合は、メモリ･バリア命令は必要ありません。例外に入る、および、例外から復帰するメカニズムでメモリ･バリアの動作が行われるためです。

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。例外からの復帰を使用して、または例外を取得することによって、プログラミング･シーケンスに入る場合は、ISB は必要ありません。

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、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（203 ページのMPU 領域

ベース･アドレス･レジスタ（MPU_RBAR）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
                            ; 領域番号、VALID （ビット 4）を 1 にセット
STR R2, [R0, #0x4]          ; 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 = 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
STM R0, {R1-R2}               ; 領域ベース･アドレス、領域番号、および VALID ビット、
                              ; および領域属性、サイズ、および有効化
```

サブ領域

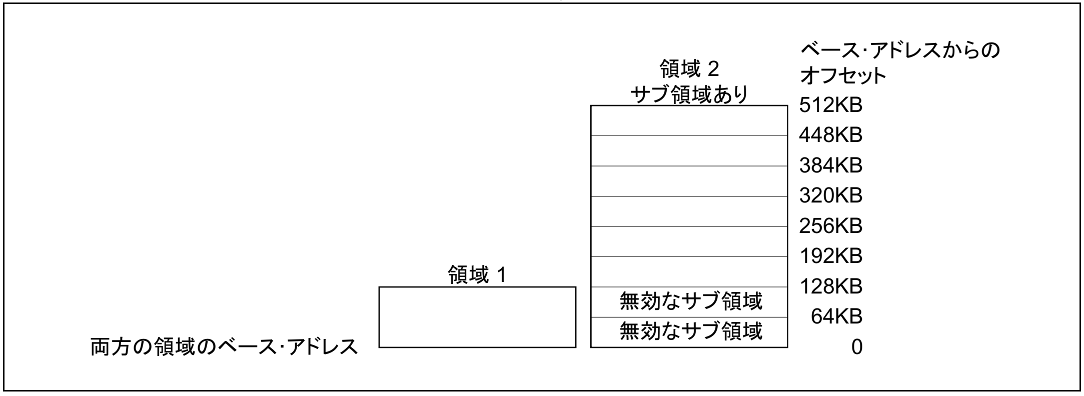
256 バイト以上の領域は、8 つの等サイズのサブ領域に分割されます。RASR の SRD フィールドで対応するビットをセットして、サブ領域を無効にします（204 ページのセクション 4.2.9: MPU 領域属性およびサイズ･レジスタ（MPU_RASR）を参照）。SRD の最下位ビットが最初のサブ領域を制御し、最上位ビットが最後のサブ領域を制御します。あるサブ領域を無効にすると、その無効な範囲とオーバーラップするもう 1 つの領域が代わりに一致をとります。無効なサブ領域とオーバーラップする有効なサブ領域が存在しない場合、MPU はフォールトを発生させます。

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 b00000011 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 18. サブ領域の例



4.2.4 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。

ソフトウェアがMPU レジスタへのアクセスに適したサイズのアラインド･アクセスを使用するようにします。

- RASR 以外は、ワードでのアラインド･アクセスを使用する必要があります。
- RASR 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ンド･アクセスを使用できます。

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 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。

MPU の推奨設定

STM32 マイクロコントローラ･システムにはプロセッサが 1 つしかないため、MPU は次のようにプログラムする必要があります。

表 41. STM32 のメモリ領域属性

メモリ領域	TEX	C	B	S	メモリ･タイプと属性
Flash メモリ	b000	1	0	0	ノーマル･メモリ、共有不可、ライトスルー
内部 SRAM	b000	1	0	1	ノーマル･メモリ、共有可能、ライトスルー
外部 SRAM	b000	1	1	1	ノーマル･メモリ、共有可能、ライトバック、書込み割当て
ペリフェラル	b000	0	1	1	デバイス･メモリ、共有可能

STM32 実装では、共有可能性とキャッシュ･ポリシーの属性は、システムの動作に影響を及ぼしません。しかし、MPU 領域にこれらの設定を使用すると、アプリケーション･コードの移植性が高まります。示されている値は、典型的な状況を想定しています。

注 : **MPU 属性は、メモリ／ペリフェラルのアドレス空間への DMA データ･アクセスには影響を及ぼしません。したがって、メモリ領域を想定外の DMA アクセスから保護するため、MPU は SW/CPU による DMA レジスタへのアクセスを制御する必要があります。**

4.2.5 MPU タイプ･レジスタ（MPU_TYPER）

アドレス･オフセット：0x00

リセット値：0x0000 0800

必要な特権：特権

MPU_TYPER 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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み								IREGION[7:0]							
								r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DREGION[7:0]							予約済み								SEPA RATE
r	r	r	r	r	r	r									r

ビット 31:24 **予約済み。**

ビット 23:16 **IREGION[7:0]**：MPU 命令領域の数。

サポートされている MPU 命令領域の数を示します。

常に 0x00 です。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。

ビット 15:8 **DREGION[7:0]**：MPU データ領域の数。

サポートされている MPU データ領域の数を示します。

0x08：MPU 領域は 8 個

0x00：MPU なし

ビット 7:1 **予約済み。**

ビット 0 SEPARATE：個別フラグ。

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。

0 = 統合

1 = 個別

4.2.6 MPU 制御レジスタ（MPU_CTRL）

アドレス·オフセット：0x04

リセット値：0x0000 0000

必要な特権：特権

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、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。

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 27 ページのセクション 2.2: メモリ·モデル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（29 ページの表 12: メモリ·アクセスの動作を参照）。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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み													PRIVDEFENA	HFNMIENA	ENABLE
													rw	rw	rw

ビット 31:3 **予約済み、ハードウェアによって 0 に固定されています。**

ビット 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。

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。

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。

注： **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。**

MPU が無効な場合、プロセッサはこのビットを無視します。

ビット 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。

MPU が有効な場合：

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。

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。

注： **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。**

ビット 0 **ENABLE**：MPU を有効にします。

0：MPU は無効

1：MPU は有効

4.2.7 MPU 領域番号レジスタ（MPU_RNR）

アドレス･オフセット：0x08

リセット値：0x0000 0000

必要な特権：特権

MPU_RNR レジスタは、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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み								REGION[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

ビット 31:8 **予約済み、ハードウェアによって 0 に固定されています。**

ビット 7:0 **REGION[7:0]**：MPU 領域

これらのビットは、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。MPU は 8 つ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 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 レジスタ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（MPU_RBAR）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。

4.2.8 MPU 領域ベース·アドレス·レジスタ（MPU_RBAR）

アドレス·オフセット：0x0C

リセット値：0x0000 0000

必要な特権：特権

MPU_RBAR レジスタは、MPU_RNR レジスタによって選択されている MPU 領域のベース·アドレスを定義します。このレジスタを使用して、MPU_RNR レジスタの値を更新できます。

VALID ビットを 1 にセットして MPU_RBAR レジスタに書き込むことによって、現在の領域番号を変更し、MPU_RNR レジスタを更新します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADDR[31:N]...															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
....ADDR[31:N]											VALID	REGION[3:0]			
											rw	rw	rw	rw	rw

ビット 31:N **ADDR[31:N]**：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。

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 = Log2（領域サイズ（バイト単位））

MPU_RASR レジスタで領域サイズが 4GB 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。

ビット N-1:5 **予約済み、ハードウェアによって 0 に固定されています。**

ビット 4 **VALID**：MPU 領域番号有効

書込み：

0：MPU_RNR レジスタは変更されません。プロセッサは次のように動作します。

- 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。
- REGION フィールドの値を無視します。

1：プロセッサは次のように動作します。

- MPU_RNR の値を REGION フィールドの値に更新します。
- 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。

読出し：

常に 0 が読み出されます。

ビット 3:0 **REGION[3:0]**：MPU 領域フィールド

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。

読出し時は、MPU_RNR レジスタで指定されている現在の領域番号を返します。

4.2.9 MPU 領域属性およびサイズ･レジスタ（MPU_RASR）

アドレス･オフセット：0x10

リセット値：0x0000 0000

必要な特権：特権

MPU_RASR レジスタは、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。

MPU_RASR は、ワードアクセスまたはハーフ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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み			XN	予約済み	AP[2:0]			予約済み		TEX[2:0]			S	C	B
			rw		rw	rw	rw			rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRD[7:0]								予約済み		SIZE					EN ABLE
rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

ビット 31:29 **予約済み、ハードウェアによって 0 に固定されています。**

ビット 28 **XN**：命令アクセス･ディセーブル･ビット

0：命令フェッチは有効

1：命令フェッチは無効

ビット 27 **予約済み、ハードウェアによって 0 に固定されています。**

ビット 26:24 **AP[2:0]**：アクセス許可

アクセス許可については、セクション 4: コア･ペリフェラルを参照してください。

AP ビットのエンコードについては、196 ページの表 40 を参照してください。

ビット 23:22 **予約済み、ハードウェアによって 0 に固定されています。**

ビット 21:19 **TEX[2:0]：メモリ属性**

TEX ビットのエンコードについては、195 ページの表 38

ビット 18 **S：共有可能メモリ属性**

S ビットのエンコードについては、195 ページの表 38

ビット 17 **C：メモリ属性**

ビット 16 **B：メモリ属性**

ビット 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以下を示します。

0：対応するサブ領域が有効

1：対応するサブ領域が無効

詳細は、198 ページのサブ領域を参照してください。

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。

ビット 7:6 **予約済み、ハードウェアによって 0 に固定されています。**

ビット 5:1 **SIZE**：MPU 保護領域のサイズ。
最小許容値は 3 （b00010）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。

ビット 0 **ENABLE**：領域イネーブル･ビット。

SIZE フィールドの値

SIZE フィールドは、MPU_RNR レジスタで指定されている 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 42 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 42. SIZE フィールドの値の例

SIZE 値	領域サイズ	N 値(1)	備考
b00100 (4)	32 バイト	5	最小許容サイズ
b01001 (9)	1KB	10	-
b10011 (19)	1MB	20	-
b11101 (29)	1GB	30	-
b11111 (31)	4GB	b01100	最大許容サイズ

1. MPU_RBAR レジスタ内（203 ページのセクション 4.2.8 を参照）。

4.2.10 MPU レジスタ・マップ

表 43. MPU レジスタ・マップとリセット値

オフセット	レジスタ	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	MPU_TYPER	予約済み								IREGION[7:0]								DREGION[7:0]								予約済み							SEPARATE
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0x04	MPU_CTRL	予約済み																													PRIVDEFENA	HFNMIENA	ENABLE
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	MPU_RNR	予約済み																								REGION[7:0]							
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	MPU_RBAR	ADDR[31:N]...																											VALID	REGION[3:0]			
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	MPU_RASR	予約済み			XN	予約済み	AP[2:0]			予約済み		TEX[2:0]			S	C	B	SRD[7:0]								予約済み		SIZE					EN ABLE
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	MPU_RBAR_A1(1)	ADDR[31:N]...																											VALID	REGION[3:0]			
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	MPU_RASR_A1(2)	予約済み			XN	予約済み	AP[2:0]			予約済み		TEX[2:0]			S	C	B	SRD[7:0]								予約済み		SIZE					EN ABLE
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	MPU_RBAR_A2(1)	ADDR[31:N]...																											VALID	REGION[3:0]			
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	MPU_RASR_A2(2)	予約済み			XN	予約済み	AP[2:0]			予約済み		TEX[2:0]			S	C	B	SRD[7:0]								予約済み		SIZE					EN ABLE
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	MPU_RBAR_A3(1)	ADDR[31:N]...																											VALID	REGION[3:0]			
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

表 43. MPU レジスタ·マップとリセット値 （続き）

オフセット	レジスタ	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x20	MPU_RASR_A3[2]	予約済み			XN	予約済み	AP[2:0]			予約済み		TEX[2:0]			S	C	B	SRD[7:0]								予約済み		SIZE					EN ABLE
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. MPU_RBAR レジスタのエイリアス
2. MPU_RASR レジスタのエイリアス

4.3 ネストト化されたベクタ割込みコントローラ（NVIC）

このセクションでは、ネスト化されたベクタ割込みコントローラ（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。

- 最大 81 個の割込み（割込み番号は STM32 デバイス･タイプに依存します。データシートを参照）。
- 割込みごとにプログラム可能な優先度レベル（0 ~ 15）。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 44. NVIC レジスタの概要

アドレス	名前	タイプ	必要な特権	リセット値	説明
0xE000E100 ~ 0xE000E10B	NVIC_ISER0 ~ NVIC_ISER2	RW	特権	0x00000000	210 ページの表 4.3.2: 割込みセット・イネーブル･レジスタ（NVIC_ISERx）
0XE000E180 ~ 0xE000E18B	NVIC_ICER0 ~ NVIC_ICER2	RW	特権	0x00000000	211 ページの表 4.3.3: 割込みクリア・イネーブル･レジスタ（NVIC_ICERx）
0XE000E200 ~ 0xE000E20B	NVIC_ISPR0 ~ NVIC_ISPR2	RW	特権	0x00000000	212 ページの表 4.3.4: 割込みセット・ペンディング･レジスタ（NVIC_ISPRx）
0XE000E280 ~ 0xE000E29C	NVIC_ICPR0 ~ NVIC_ICPR2	RW	特権	0x00000000	213 ページの表 4.3.5: 割込みクリア・ペンディング･レジスタ（NVIC_ICPRx）
0xE000E300 ~ 0xE000E31C	NVIC_IABR0 ~NVIC_IABR2	RW	特権	0x00000000	214 ページの表 4.3.6: 割込みアクティブ･ビット･レジスタ（NVIC_IABRx）
0xE000E400 ~ 0xE000E503	NVIC_IPR0 ~ NVIC_IPR20	RW	特権	0x00000000	215 ページの表 4.3.7: 割込み優先度レジスタ（NVIC_IPRx）
0xE000EF00	STIR	WO	設定可能	0x00000000	216 ページの表 4.3.8: ソフトウェア･トリガ割込みレジスタ（NVIC_STIR）

4.3.1 CMSIS を使用した Cortex-M4 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 45. CMSIS の NVIC アクセス関数

CMSIS 関数(1)	説明
void NVIC_EnableIRQ(IRQn_Type IRQn)	割込みまたは例外を有効にします。
void NVIC_DisableIRQ(IRQn_Type IRQn)	割込みまたは例外を無効にします。
void NVIC_SetPendingIRQ(IRQn_Type IRQn)	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)	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)	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)	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)	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。

4.3.2 割込みセット・イネーブル·レジスタ（NVIC_ISERx）

アドレス·オフセット：0x00 ~ 0x0B

リセット値：0x0000 0000

必要な特権：特権

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETENA[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETENA[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

ビット 31:0 **SETENA**：割込みセット・イネーブル·ビット。

書込み：

0：影響なし。
1：割込みを有効にします。

読出し：

0：割込みは無効です。
1：割込みは有効です。

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。

4.3.3 割込みクリア・イネーブル·レジスタ（NVIC_ICERx）

アドレス·オフセット：0x00 ~ 0x0B

リセット値：0x0000 0000

必要な特権：特権

ICER0 ~ ICER2 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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLRENA[31:16]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLRENA[15:0]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

ビット 31:0 **CLRENA**：割込みクリア・イネーブル·ビット。

書込み：

0：影響なし。
1：割込みを無効にします。

読出し：

0：割込みは無効です。
1：割込みは有効です。

4.3.4 割込みセット・ペンディング･レジスタ（NVIC_ISPRx）

アドレス･オフセット：0x00 ~ 0x0B

リセット値：0x0000 0000

必要な特権：特権

ISPR0 ~ ISPR2 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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETPEND[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETPEND[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

ビット 31:0 **SETPEND**：割込みセット・ペンディング･ビット

書込み：

0：影響なし。

1：割込み状態を保留中に変更します。

読出し：

0：割込みは保留中ではありません。

1：割込みは保留中です。

保留中の割込みに対応する ISPR ビットへの 1 の書込み：

– 影響なし。

無効な割込みに対応する ISPR ビットへの 1 の書込み：

– その割込みの状態を保留中に設定します。

4.3.5 割込みクリア・ペンディング·レジスタ（NVIC_ICPRx）

アドレス·オフセット：0x00 ~ 0x0B

リセット値：0x0000 0000

必要な特権：特権

ICPR0 ~ ICPR2 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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLRPEND[31:16]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLRPEND[15:0]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

ビット 31:0 **CLRPEND**：割込みクリア・ペンディング·ビット

書込み：

0：影響なし。

1：割込みの保留状態を解除します。

読出し：

0：割込みは保留中ではありません。

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。

4.3.6 割込みアクティブ·ビット·レジスタ（NVIC_IABRx）

アドレス·オフセット：0x00 ~ 0x0B

リセット値：0x0000 0000

必要な特権：特権

IABR0 ~ IABR2 の各レジスタは、どの割込みがアクティブかを示します。

ビット割当てを次に示します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ACTIVE[31:16]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACTIVE[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

ビット 31:0 **ACTIVE**：割込みアクティブ·フラグ

0：割込みはアクティブではありません。

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。

4.3.7 割込み優先度レジスタ（NVIC_IPRx）

アドレス·オフセット：0x00 ~ 0x0B

リセット値：0x0000 0000

必要な特権：特権

NVIC_IPR0 ~ IPR20 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。各レジスタは、図 19 に示すように、4 つの優先度フィールドを保持し、それらが CMSIS 割込み優先度配列 IP[0] ~ IP[80] の 4 つの要素にマッピングされます。

図 19. NVIC_IPRx レジスタ·マッピング

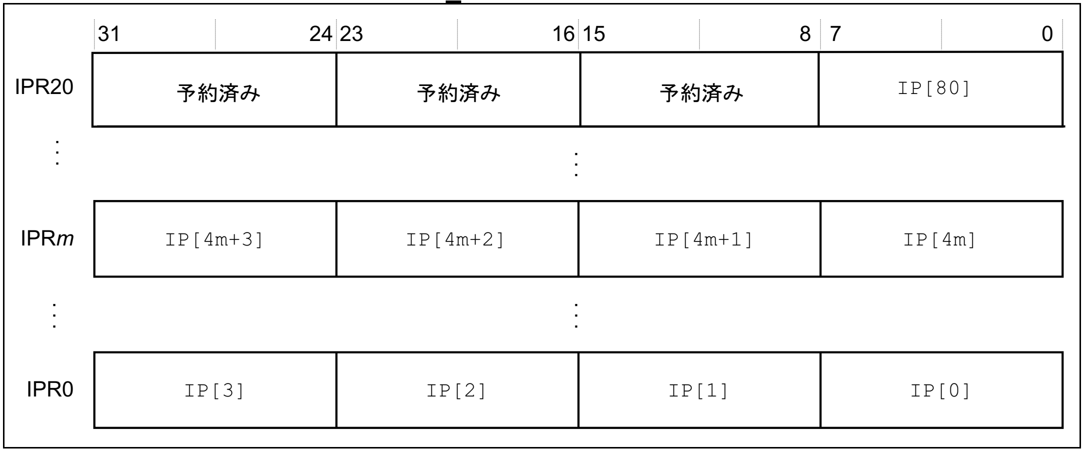


表 46. IPR ビット割当て

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 [7:4] のみ実装し、ビット [3:0] 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の詳細については、210 ページの割込みセット・イネーブル·レジスタ（NVIC_ISERx）を参照してください。

割込み N 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 M は、M = N 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 N 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。

4.3.8 ソフトウェア・トリガ割込みレジスタ（NVIC_STIR）

アドレス・オフセット：0xE00

リセット値：0x0000 0000

必要な特権 : 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（セクション 4.4.6: システム制御レジスタ（SCR）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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み							INTID[8:0]								
							w	w	w	w	w	w	w	w	w

ビット 31:9	予約済み、クリア状態を保つ必要があります。
ビット 8:0	**INTID** ソフトウェア生成割込み ID STIRに書き込むと、ソフトウェア生成割込み（SGI）が生成されます。書き込む値は、必要なSGIの割込みID （0 ~ 239の範囲）です。たとえば、値0x03は、割込みIRQ3を指定します。

4.3.9 レベル割込みとパルス割込み

STM32 は、レベル検出割込みとパルス検出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（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-M4 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・ペンディング・レジスタの対応するビットに書き込む（セクション 4.3.4: 割込みセット・ペンディング·レジスタ（NVIC_ISPRx）を参照）か、または STIR に書き込んで SGI を保留中にする（セクション 4.3.8: ソフトウェア·トリガ割込みレジスタ（NVIC_STIR）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。その後で、次の動作が行われます。
  - 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む。

  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

4.3.10 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、226 ページのセクション 4.4.4: ベクタ･テーブル･オフセット･レジスタ（VTOR）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 47. 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。
void NVIC_SystemReset (void)	システムをリセットします。

入力パラメータ IRQn は IRQ 番号です（37 ページの表 16: さまざまな例外タイプのプロパティを参照）。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。

4.3.11 NVIC レジスタ·マップ

次の表に、NVIC レジスタ·マップとリセット値を示します。メインの NVIC レジスタ·ブロックのベース·アドレスは、0xE000E100 です。NVIC_STIR レジスタは、0xE000EF00 にある独立したブロックに配置されています。

表 48. NVIC レジスタ·マップとリセット値

オフセット	レジスタ	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000	NVIC_ISER0	SETENA[31:0]																															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x004	NVIC_ISER1	SETENA[63:32]																															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x008	NVIC_ISER2	予約済み															SETENA[80:64]																
	リセット値																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x080	NVIC_ICER0	CLRENA[31:0]																															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x084	NVIC_ICER1	CLRENA[63:32]																															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x088	NVIC_ICER2	予約済み															CLRENA [80:64]																
	リセット値																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x100	NVIC_ISPR0	SETPEND[31:0]																															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x104	NVIC_ISPR1	SETPEND[63:32]																															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x108	NVIC_ISPR2	予約済み															SETPEND [80:64]																
	リセット値																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x180	NVIC_ICPR0	CLRPEND[31:0]																															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x184	NVIC_ICPR1	CLRPEND[63:32]																															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x188	NVIC_ICPR2	予約済み															CLRPEND [80:64]																
	リセット値																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x200	NVIC_IABR0	ACTIVE[31:0]																															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x204	NVIC_IABR1	ACTIVE[63:32]																															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x208	NVIC_IABR2	予約済み															ACTIVE [80:64]																
	リセット値																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x300	NVIC_IPR0	IP[3]								IP[2]								IP[1]								IP[0]							
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
:	:	:																															

表 48. NVIC レジスタ・マップとリセット値 （続き）

オフセット	レジスタ	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x368	NVIC_IPR25	予約済み																IP[101]								IP[100]							
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SCB レジスタ																																	
予約済み																																	
0xE00	NVIC_STIR	予約済み																							INTID[8:0]								
	リセット値																								0	0	0	0	0	0	0	0	0

4.4 システム制御ブロック（SCB）

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。

表 49. 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	必要な特権	リセット値	説明
0xE000E008	ACTLR	RW	特権	0x00000000	222 ページの表 4.4.1: 補助制御レジスタ（ACTLR）
0xE000ED00	CPUID	RO	特権	0x410FC241	223 ページの表 4.4.2: CPUID ベース･レジスタ（CPUID）
0xE000ED04	ICSR	RW(1)	特権	0x00000000	224 ページの表 4.4.3: 割込み制御およびステート・レジスタ（ICSR）
0xE000ED08	VTOR	RW	特権	0x00000000	226 ページの表 4.4.4: ベクタ･テーブル･オフセット･レジスタ（VTOR）
0xE000ED0C	AIRCR	RW(1)	特権	0xFA050000	227 ページの表 4.4.5: 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）
0xE000ED10	SCR	RW	特権	0x00000000	229 ページの表 4.4.6: システム制御レジスタ（SCR）
0xE000ED14	CCR	RW	特権	0x00000200	230 ページの表 4.4.7: 設定および制御レジスタ（CCR）
0xE000ED18	SHPR1	RW	特権	0x00000000	232 ページの表 4.4.8: システム･ハンドラ優先度レジスタ（SHPRx）
0xE000ED1C	SHPR2	RW	特権	0x00000000	
0xE000ED20	SHPR3	RW	特権	0x00000000	
0xE000ED24	SHCRS	RW	特権	0x00000000	234 ページの表 4.4.9: システム･ハンドラ制御およびステート・レジスタ（SHCSR）
0xE000ED28	CFSR	RW	特権	0x00000000	236 ページの表 4.4.10: 設定可能フォールト･ステータス･レジスタ（CFSR：UFSR+BFSR+MMFSR）
0xE000ED28	MMSR(2)	RW	特権	0x00	メモリ管理フォールト･ステータス･レジスタ 236 ページの表 4.4.10
0xE000ED29	BFSR(2)	RW	特権	0x00	バス･フォールト･ステータス･レジスタ 236 ページの表 4.4.10
0xE000ED2A	UFSR(2)	RW	特権	0x0000	用法フォールト･ステータス･レジスタ 236 ページの表 4.4.10
0xE000ED2C	HFSR	RW	特権	0x00000000	240 ページの表 4.4.14: ハード･フォールト･ステータス･レジスタ（HFSR）
0xE000ED34	MMAR	RW	特権	予測不能	241 ページの表 4.4.15: メモリ管理フォールト･アドレス･レジスタ（MMFAR）
0xE000ED38	BFAR	RW	特権	予測不能	241 ページの表 4.4.16: バス･フォールト･アドレス･レジスタ（BFAR）
0xE000ED3C	AFSR	RW	特権	0x00000000	242 ページの表 4.4.17: 補助フォールト･ステータス･レジスタ（AFSR）

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

4.4.1 補助制御レジスタ（ACTLR）

アドレス·オフセット：0x00 （ベース·アドレス = 0xE000 E008）

リセット値：0x0000 0000

必要な特権：特権

このレジスタは、デフォルトで Cortex-M4 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 レジスタには、次のプロセッサ機能のディセーブル·ビットがあります。

- IT フォールディング
- デフォルトのメモリ·マップへのアクセスでの書込みバッファの使用
- マルチサイクル命令の割込み

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み						DISOOFP	DISFPCA						DISFOLD	DISDEFWBUF	DISMCYCINT
						rw	rw						rw	rw	rw

ビット 31:10 予約済み

ビット 9 DISOOFP

浮動小数点命令が整数命令に対して順不同に完了することを無効にします。

ビット 8 DISFPCA

CONTROL.FPCA の自動更新を無効にします。

このビットの値は、0 として書き込まれる、または保持される（SBZP）必要があります。

ビット 7:3 予約済み

ビット 2 DISFOLD

IT 命令のフォールディングを無効にします。

0：IT 命令のフォールディングを有効にします。

1：IT 命令のフォールディングを無効にします。

状況によっては、プロセッサは、IT 命令をまだ実行している間に、IT ブロックの最初の命令の実行を開始できます。この動作は IT フォールディングと呼ばれ、性能が向上しますが、ルーピングでジッタを引き起こす可能性があります。タスクでジッタを避ける必要がある場合は、タスクを実行する前に DISFOLD ビットを 1 にセットして、IT フォールディングを無効にします。

ビット 1 DISDEFWBUF

このビットは、Cortex-M4 プロセッサに実装されている書込みバッファにのみ影響します。

デフォルトのメモリ·マップへのアクセスでの書込みバッファの使用を無効にします。これにより、すべてのバス·フォールトは正確なバス·フォールトになりますが、プロセッサが次の命令を実行する前にメモリへのすべてのストアが完了する必要があるので、性能は低下します。

0：書込みバッファの使用を有効にします。

1：書込みバッファの使用を無効にします。メモリへのストアは、次の命令の前に完了します。

ビット 0 DISMCYCINT

マルチサイクル命令の割込みを無効にします。1 にセットされている場合、多重ロード命令と多重ストア命令の割込みを無効にします。これにより、プロセッサが現在の状態をスタックして割込みハンドラに入る前にすべての LDM または STM が完了する必要があるため、プロセッサの割込み遅延は増加します。

0：プロセッサの割込み遅延（ロード／ストア動作と乗算／除算動作）を有効にします。

1：割込み遅延を無効にします。

4.4.2 CPUID ベース·レジスタ（CPUID）

アドレス·オフセット：0x00

リセット値：0x410F C241

必要な特権：特権

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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
実装者								バリアント				定数			
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
部品番号												リビジョン			
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

ビット 31:24 **実装者**：実装者コード

0x41：ARM

ビット 23:20 **バリアント**：バリアント番号

製品リビジョン識別子 rnpn の r の値

0x0：リビジョン 0

ビット 19:16 **定数**：0xF として読み出されます。

ビット 15:4 **部品番号**：プロセッサの部品番号

0xC24：= Cortex-M4

ビット 3:0 **リビジョン**：リビジョン番号

製品リビジョン識別子 rnpn の p の値で、パッチ·リリースを示します。

0x1：= パッチ 1

4.4.3 割込み制御およびステート・レジスタ（ICSR）

アドレス·オフセット：0x04

リセット値：0x0000 0000

必要な特権：特権

ICSRの内容は以下のとおりです。

- ビット：
  - 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

注意： **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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NMIPENDSET	予約済み		PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	予約済み		ISRPENDING	予約済み			VECTPENDING[6:4]		
rw			rw	w	rw	w			r				r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VECTPENDING[3:0]				RETOBASE	予約済み		VECTACTIVE[8:0]								
r	r	r	r	r			rw	rw	rw	rw	rw	rw	rw	rw	rw

ビット 31 **NMIPENDSET**：NMI セット・ペンディング·ビット。

書込み：

0：影響なし。

1：NMI 例外の状態を保留中に変更します。

読出し：

0：NMI 例外は保留中ではありません。

1：NMI 例外は保留中です。

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。

ビット 30:29 予約済み

ビット 28 **PENDSVSET**：PendSV セット・ペンディング·ビット。

書込み：

0：影響なし。
1：PendSV 例外の状態を保留中に変更します。

読出し：

0：PendSV 例外は保留中ではありません。
1：PendSV 例外は保留中です。

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。

ビット 27 **PENDSVCLR**：PendSV クリア・ペンディング·ビット。このビットは書き込み専用です。読み出したときの値は予測不能です。

0：影響なし。
1：PendSV 例外の保留状態を解除します。

ビット 26 **PENDSTSET**：SysTick 例外セット・ペンディング·ビット。

書込み：

0：影響なし。
1：SysTick 例外の状態を保留中に変更します。

読出し：

0：SysTick 例外は保留中ではありません。
1：SysTick 例外は保留中です。

ビット 25 **PENDSTCLR**：SysTick 例外クリア・ペンディング·ビット。書込み専用。読み出したときの値は予測不能です。

0：影響なし。
1：SysTick 例外の保留状態を解除します。

ビット 24 予約済み、クリア状態を保つ必要があります。

ビット 23 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。

ビット 22 **ISRPENDING**：割込み保留中フラグ。NMI とフォールトは除外します。

0：割込みは保留中ではありません。
1：割込みが保留中です。

ビット 21:19 予約済み、クリア状態を保つ必要があります。

ビット 18:12 **VECTPENDING**：保留中のベクタ。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。

0：保留中の例外なし。
その他の値：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。

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。

ビット 11 **RETTOBASE**：ベース·レベルに戻ります。横取りされたアクティブな例外の有無を示します。

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。
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。

ビット 10:9 予約済み

ビット 8:0 VECTACTIVE アクティブ·ベクタ。アクティブな例外の番号が格納されます。

0：スレッド·モード
その他の値：現在アクティブな例外の例外番号(1)。

注：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クスになります（21 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（21 ページの割込みプログラム·ステータス·レジスタを参照）。

4.4.4 ベクタ･テーブル･オフセット･レジスタ（VTOR）

アドレス･オフセット：0x08

リセット値：0x0000 0000

必要な特権：特権

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み		TBLOFF[29:16]													
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TBLOFF[15:9]							予約済み								
rw	rw	rw	rw	rw	rw	rw									

ビット 31:30 予約済み、クリア状態を保つ必要があります。

ビット 29:9 **TBLOFF**：ベクタ･テーブル･ベース･オフセット･フィールド。

テーブル･ベースのメモリ･アドレス 0x00000000 からのオフセットのビット [29:9] が格納されています。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せる必要があります。最小アラインメントは 128 ワードです。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。

ビット 29 は、ベクタ･テーブルがコードまたは SRAM のどちらのメモリ領域にあるかを示します。

0：コード

1：SRAM

注： ビット 29 は TBLBASE ビットと呼ばれることもあります。

ビット 8:0 予約済み、クリア状態を保つ必要があります。

4.4.5 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）

アドレス·オフセット：0x0C

リセット値：0xFA05 0000

必要な特権：特権

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。

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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VECTKEYSTAT[15:0]（読出し時）/ VECTKEY[15:0]（書込み時）															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ENDIANESS	予約済み				PRIGROUP			予約済み					SYS RESET REQ	VECT CLR ACTIVE	VECT RESET
r					rw	rw	rw						w	w	w

ビット 31:16 **VECTKEYSTAT/VECTKEY**：レジスタ·キー

0xFA05 として読み出されます。

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。

ビット 15 ENDIANESS：データのエンディアン形式ビット

0 として読み出されます。

0：リトルエンディアン

ビット 14:11 予約済み、クリア状態を保つ必要があります。

ビット 10:8 **PRIGROUP**：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（228 ページの2 進小数点を参照）。

ビット 7:3 予約済み、クリア状態を保つ必要があります。

ビット 2 **SYSRESETREQ**：システム·リセット要求

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。

このビットは 0 として読み出されます。

0：システム·リセット要求なし。

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。

ビット 1 VECTCLRACTIVE

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。

ビット 0 VECTRESET

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 50 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 50. 優先度のグループ化

PRIGROUP [2:0]	割込み優先度レベル値、PRI_*N*[7:4]			数	
	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
0b0xx	0bxxxx	[7:4]	なし	16	なし
0b100	0bxxx.y	[7:5]	[4]	8	2
0b101	0bxx.yy	[7:6]	[5:4]	4	4
0b110	0bx.yyy	[7]	[6:4]	2	8
0b111	0b.yyyy	なし	[7:4]	なし	16

1. PRI_n[7:4] 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。

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（40 ページのセクション 2.3.6: 割込み優先度のグループ化を参照）。

4.4.6 システム制御レジスタ（SCR）

アドレス･オフセット：0x10

リセット値：0x0000 0000

必要な特権：特権

SCR は、低電力状態の開始と終了の機能を制御します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み											SEVONPEND	予約済み	SLEEPDEEP	SLEEP ON EXIT	予約済み
											rw		rw	rw	

ビット 31:5 予約済み、クリア状態を保つ必要があります。

ビット 4 **SEVEONPEND**：保留時イベント送信ビット。

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。

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。

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。

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。

ビット 3 予約済み、クリア状態を保つ必要があります。

ビット 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。

0：スリープ

1：ディープ･スリープ

ビット 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 を設定します。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：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。

1：割込みサービス･ルーチン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。

ビット 0 予約済み、クリア状態を保つ必要があります。

4.4.7 設定および制御レジスタ（CCR）

アドレス·オフセット：0x14

リセット値：0x0000 0200

必要な特権：特権

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（216 ページのソフトウェア·トリガ割込みレジスタ（NVIC_STIR）を参照）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み						STKALIGN	BFHFNMIGN	予約済み			DIV_0_TRP	UNALIGN_TRP	予約済み	USERSETMPEND	NONBASETHRDENA
						rw	rw				rw	rw		rw	rw

ビット 31:10 予約済み、クリア状態を保つ必要があります。

ビット 9 STKALIGN

例外の開始時のスタック·アライメントを設定します。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。

0：4 バイトに整列

1：8 バイトに整列

ビット 8 BFHFNMIGN

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。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。

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。

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。

ビット 7:5 予約済み、クリア状態を保つ必要があります。

ビット 4 DIV_0_TRP

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。

0：0 による除算をトラップしません。

1：0 による除算をトラップします。

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。

ビット 3 UNALIGN_ TRP

アンアラインド·アクセスのトラップを有効にします。

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。

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。

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。

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。

ビット 2 予約済み、クリア状態を保つ必要があります。

ビット 1 USERSETMPEND

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（216 ページのソフトウェア·トリガ割込みレジスタ（NVIC_STIR）を参照）。

0：無効にします。

1：有効にします。

ビット 0 NONBASETHRDENA

プロセッサがスレッド·モードに移行する方法を設定します。

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。

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（43 ページの例外からの復帰を参照）。

4.4.8 システム·ハンドラ優先度レジスタ（SHPRx）

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。

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 51. 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
メモリ管理フォールト	PRI_4	システム·ハンドラ優先度レジスタ 1 （SHPR1）
バス·フォールト	PRI_5	
用法フォールト	PRI_6	
SVCall	PRI_11	233 ページのシステム·ハンドラ優先度レジスタ 2 （SHPR2）
PendSV	PRI_14	233 ページのシステム·ハンドラ優先度レジスタ 3 （SHPR3）
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 [7:3] のみ実装し、ビット [3:0] は 0 として読み出され、ここへの書込みは無視されます（M=4 の場合）。

システム·ハンドラ優先度レジスタ 1 （SHPR1）

アドレス·オフセット：0x18

リセット値：0x0000 0000

必要な特権：特権

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み								PRI_6[7:4]				PRI_6[3:0]			
								rw	rw	rw	rw	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRI_5[7:4]				PRI_5[3:0]				PRI_4[7:4]				PRI_4[7:4]			
rw	rw	rw	rw	r	r	r	r	rw	rw	rw	rw	r	r	r	r

ビット 31:24	予約済み、クリア状態を保つ必要があります。
ビット 23:16	PRI_6：システム·ハンドラ 6、用法フォールトの優先度
ビット 15:8	PRI_5：システム·ハンドラ 5、バス·フォールトの優先度
ビット 7:0	PRI_4：システム·ハンドラ 4、メモリ管理フォールトの優先度

システム·ハンドラ優先度レジスタ 2 （SHPR2）

アドレス·オフセット：0x1C

リセット値：0x0000 0000

必要な特権：特権

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRI_11[7:4]				PRI_11[3:0]				予約済み							
rw	rw	rw	rw	r	r	r	r								

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み															

ビット 31:24 **PRI_11**：システム·ハンドラ 11、SVCall の優先度

ビット 23:0 予約済み、クリア状態を保つ必要があります。

システム·ハンドラ優先度レジスタ 3 （SHPR3）

アドレス：0xE000 ED20

リセット値：0x0000 0000

必要な特権：特権

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRI_15[7:4]				PRI_15[3:0]				PRI_14[7:4]				PRI_14[3:0]			
rw	rw	rw	rw	r	r	r	r	rw	rw	rw	rw	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み															

ビット 31:24 **PRI_15**：システム·ハンドラ 15、SysTick 例外の優先度

ビット 23:16 **PRI_14**：システム·ハンドラ 14、PendSV の優先度

ビット 15:0 予約済み、クリア状態を保つ必要があります。

4.4.9 システム·ハンドラ制御およびステート・レジスタ（SHCSR）

アドレス·オフセット：0x24

リセット値：0x0000 0000

必要な特権：特権

SHCSR は、システム·ハンドラを有効にし、以下を示します。

- バス·フォールト、メモリ管理フォールト、および SVC の各例外の保留ステータス
- システム·ハンドラのアクティブ·ステータス

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、リード・モデファイ・ライトを使用する必要があります。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み													USG FAULT ENA	BUS FAULT ENA	MEM FAULT ENA
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SV CALL PENDED	BUS FAULT PENDED	MEM FAULT PENDED	USG FAULT PENDED	SYS TICK ACT	PENDSV ACT	予約済み	MONITOR ACT	SV CALL ACT	予約済み			USG FAULT ACT	予約済み	BUS FAULT ACT	MEM FAULT ACT
rw	rw	rw	rw	rw	rw		rw	rw				rw		rw	rw

ビット 31:19 予約済み、クリア状態を保つ必要があります。

ビット 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)

ビット 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)

ビット 16 **MEMFAULTENA**：メモリ管理フォールト·イネーブル·ビット、有効にするには 1 をセット(1)

ビット 15 **SVCALLPENDED**：SVC コール保留ビット、例外が保留中の場合に 1 として読出し(2)

ビット 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)

ビット 13 **MEMFAULTPENDED**：メモリ管理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)

ビット 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)

ビット 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)

ビット 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

ビット 9 予約済み、クリア状態を保つ必要があります。

ビット 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

ビット 7	**SVCALLACT**：SVC 呼出しアクティブ·ビット、SVC 呼出しがアクティブの場合に 1 として読出し
ビット 6:4	予約済み、クリア状態を保つ必要があります。
ビット 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
ビット 2	予約済み、クリア状態を保つ必要があります。
ビット 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
ビット 0	**MEMFAULTACT**：メモリ管理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

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。

4.4.10 設定可能フォールト･ステータス･レジスタ（CFSR：UFSR+BFSR+MMFSR）

アドレス･オフセット：0x28

リセット値：0x0000 0000

必要な特権：特権

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。

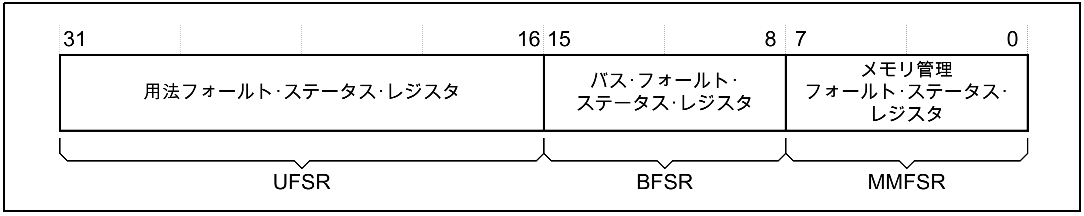
- 237 ページの用法フォールト･ステータス･レジスタ（UFSR）
- 238 ページのバス･フォールト･ステータス･レジスタ（BFSR）
- 239 ページのメモリ管理フォールト･ステータス･レジスタ（MMFSR）

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、次の方法でアクセスできます。

- 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。

CFSR は、メモリ管理フォールト、バス･フォールト、または用法フォールトの原因を示します。

図 20. CFSR のサブレジストリ



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み						DIVBYZERO	UNALIGNED	予約済み				NOCP	INVPC	INVSTATE	UNDEFINSTR
						rc_w1	rc_w1					rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BFARVALID	予約済み	LSPERR	STKERR	UNSTKERR	IMPRECISERR	PRECISERR	IBUSERR	MMARVALID	予約済み	MLSPERR	MSTKERR	MUNSTKERR	予約済み	DACCVIOL	IACCVIOL
rw		rw	rw	rw	rw	rw	rw	rw		rw	rw	rw		rw	rw

ビット 31:16 **UFSR**：237 ページの用法フォールト･ステータス･レジスタ（UFSR）を参照してください。

ビット 15:8 **BFSR**：238 ページのバス･フォールト･ステータス･レジスタ（BFSR）を参照してください。

ビット 7:0 **MMFSR**：239 ページのメモリ管理フォールト･ステータス･レジスタ（MMFSR）を参照してください。

4.4.11 用法フォールト･ステータス･レジスタ（UFSR）

ビット 31:26	予約済み、クリア状態を保つ必要があります。
ビット 25	**DIVBYZERO**：0 による除算の用法フォールト。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。 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（230 ページの設定および制御レジスタ（CCR）を参照）。 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。 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。
ビット 24	**UNALIGNED**：アンアラインド･アクセスによる用法フォールト。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（230 ページの設定および制御レジスタ（CCR）を参照）。 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。 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。 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。
ビット 23:20	予約済み、クリア状態を保つ必要があります。
ビット 19	**NOCP**：コプロセッサがないことによる用法フォールト。プロセッサは、コプロセッサ命令をサポートしません。 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。 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。
ビット 18	**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。 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。 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。 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。
ビット 17	**INVSTATE**：無効な状態による用法フォールト。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。 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：無効な状態による用法フォールトはありません。 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。
ビット 16	**UNDEFINSTR**：未定義命令による用法フォールト。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。 未定義命令は、プロセッサがデコードできない命令です。 0：未定義命令による用法フォールトはありません。 1：プロセッサが未定義命令を実行しようとしました。

4.4.12 バス·フォールト·ステータス·レジスタ（BFSR）

ビット 15 **BFARVALID** : バス·フォールト·アドレス·レジスタ（BFAR）有効フラグ。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。

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の問題が回避されます。

0 : BFAR の値は有効なフォールト·アドレスではありません。

1 : BFAR に有効なフォールト·アドレスが保持されています。

ビット 14 予約済み、クリア状態を保つ必要があります。

ビット 13 **LSPERR** : 浮動小数点のレイジーな状態保持の際のバス·フォールト。

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。

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。

ビット 12 **STKERR** : 例外開始時のスタッキングでのバス·フォールト。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。

0 : スタッキング·フォールトはありません。

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。

ビット 11 **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。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。

0 : アンスタッキング·フォールトはありません。

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。

ビット 10 **IMPRECISERR** : 不正確なデータ·バス·エラー。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。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。

0 : 不正確なデータ·バス·エラーはありません。

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。

ビット 9 **PRECISERR** : 正確なデータ·バス·エラー。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 : 正確なデータ·バス·エラーはありません。

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。

ビット 8 **IBUSERR** : 命令バス·エラー。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。

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。

0 : 命令バス·エラーはありません。

1 : 命令バス·エラー。

4.4.13 メモリ管理フォールト･ステータス･レジスタ（MMFSR）

ビット 7 **MMARVALID**：メモリ管理フォールト･アドレス･レジスタ（MMAR）有効フラグ。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･ハンドラに戻った場合の問題が回避されます。

0：MMAR の値は有効なフォールト･アドレスではありません。
1：MMAR に有効なフォールト･アドレスが保持されています。

ビット 6 予約済み、クリア状態を保つ必要があります。

ビット 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。
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。

ビット 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。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。

0：スタッキング･フォールトはありません。
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。

ビット 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。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。

0：アンスタッキング･フォールトはありません。
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。

ビット 2 予約済み、クリア状態を保つ必要があります。

ビット 1 **DACCVIOL**：データ･アクセス違反フラグ。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：データ･アクセス違反フォールトはありません。
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。

ビット 1 **IACCVIOL**：命令アクセス違反フラグ。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。

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。

0：命令アクセス違反フォールトはありません。
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。

4.4.14 ハード·フォールト·ステータス·レジスタ（HFSR）

アドレス·オフセット：0x2C

リセット値：0x0000 0000

必要な特権：特権

HFSR は、ハード·フォールト·ハンドラのアクティブなイベントに関する情報を提供します。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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DEBUG_VT	FORCED	予約済み													
rc_w1	rc_w1														

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み														VECTTBL	予約済み
														rc_w1	

ビット 31 DEBUG_VT：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 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。

ビット 30 FORCED：強制ハード·フォールト。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。

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。

0：強制ハード·フォールトはありません。
1：強制ハード·フォールト。

ビット 29:2 予約済み、クリア状態を保つ必要があります。

ビット 1 VECTTBL：ベクタ·テーブル·ハード·フォールト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。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。

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：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。
1：ベクタ·テーブル読出し時のバス·フォールト。

ビット 0 予約済み、クリア状態を保つ必要があります。

4.4.15 メモリ管理フォールト･アドレス･レジスタ（MMFAR）

アドレス･オフセット：0x34

リセット値：定義されていません。

必要な特権：特権

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MMFAR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MMFAR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ビット 31:0 **MMFAR**：メモリ管理フォールト･アドレス

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 レジスタのフラグは、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。236 ページの設定可能フォールト･ステータス･レジスタ（CFSR：UFSR+BFSR+MMFSR）を参照してください。

4.4.16 バス･フォールト･アドレス･レジスタ（BFAR）

アドレス･オフセット：0x38

リセット値：定義されていません。

必要な特権：特権

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BFAR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BFAR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ビット 31:0 **BFAR**：バス･フォールト･アドレス

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 レジスタのフラグは、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。236 ページの設定可能フォールト･ステータス･レジスタ（CFSR：UFSR+BFSR+MMFSR）を参照してください。

4.4.17 補助フォールト·ステータス·レジスタ（AFSR）

アドレス·オフセット：0x3C

リセット値：定義されていません。

必要な特権：特権

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IMPDEF[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMPDEF[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ビット 31:0 **IMPDEF**：実装定義。AFSR には追加のシステム·フォールト情報が格納されます。これらのビットは、AUXFAULT 入力信号にマッピングされます。

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。

AFSR の各ビットは、プロセッサの AUXFAULT 入力に直接マッピングされ、1 サイクルの HIGH 信号が入力されると、対応する AFSR ビットが 1 にセットされます。これは、そのビットに 1 を書き込むことによってそのビットが 0 にクリアされるまで 1 のままです。

AFSR ビットが 1 としてラッチされている場合、例外は発生しません。例外が必要な場合は、割込みを使用します。

4.4.18 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。

4.4.19 SCB レジスタ･マップ

次の表に、システム制御ブロックのレジスタ･マップとリセット値を示します。SCB レジスタ･ブロックのベース･アドレスは、表 52 に示すレジスタの場合、0xE000 ED00 です。

表 52. SCB レジスタ･マップとリセット値

オフセット	レジスタ	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	CPUID	実装者								バリアント				定数				部品番号												リビジョン			
	リセット値	0	1	0	0	0	0	0	1	0	0	0	1	1	1	1	1	1	1	0	0	0	0	1	0	0	0	1	1	0	0	0	1
0x04	ICSR	NMIPENDSET	予約済み		PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	予約済み		ISRPENDING	VECTPENDING[9:0]										RETOBASE	予約済み		VECTACTIVE[8:0]								
	リセット値	0			0	0	0	0			0	0	0	0	0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0	0
0x08	VTOR	予約済み		TABLEOFF[29:9]																					予約済み								
	リセット値			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0									
0x0C	AIRCR	VECTKEY[15:0]																ENDIANESS	予約済み				PRIGROUP[2:0]			予約済み					SYSRESETREQ	VECTCLRACTIVE	VECTRESET
	リセット値	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	1	0					0	0	0						0	0	0
0x10	SCR	予約済み																											SEVONPEND	予約済み	SLEEPDEEP	SLEEPONEXIT	予約済み
	リセット値																												0		0	0	
0x14	CCR	予約済み																						STKALIGN	BFHFNIGN	予約済み			DIV_0_TRP	UNALIGN_TRP	予約済み	USERSETMPEND	NONBASETHRDENA
	リセット値																							1	0				0	0		0	0
0x18	SHPR1	予約済み								PRI6								PRI5								PRI4							
	リセット値									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	SHPR2	PRI11								予約済み																							
	リセット値	0	0	0	0	0	0	0	0																								
0x20	SHPR3	PRI15								PRI14								予約済み															
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																

表 52. SCB レジスタ･マップとリセット値 （続き）

オフセット	レジスタ	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x24	SHCRS	予約済み													USG FAULT ENA	BUS FAULT ENA	MEM FAULT ENA	SV CALL PENDED	BUS FAULT PENDED	MEM FAULT PENDED	USG FAULT PENDED	SYS TICK ACT	PENDSV ACT	予約済み	MONITOR ACT	SV CALL ACT	予約済み			USG FAULT ACT	予約済み	BUS FAULT ACT	MEM FAULT ACT
	リセット値														0	0	0	0	0	0	0	0	0		0	0				0		0	0
0x28	CFSR	UFSR																BFSR								MMFSR							
	リセット値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C	HFSR	DEBUG_VT	FORCED	予約済み																												VECTTBL	予約済み
	リセット値	0	0																													0	
0x34	MMAR	MMAR[31:0]																															
	リセット値	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x38	BFAR	BFAR[31:0]																															
	リセット値	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x3C	AFSR	IMPDEF[31:0]																															
	リセット値	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

4.5 SysTick タイマ（STK）

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 STK_LOAD レジスタの値を再ロード（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。

表 53. システム･タイマのレジスタの概要

アドレス	名前	タイプ	必要な特権	リセット値	説明
0xE000E010	STK_CTRL	RW	特権	0x00000000	246 ページのSysTick 制御およびステータス･レジスタ（STK_CTRL）
0xE000E014	STK_LOAD	RW	特権	予測不能	247 ページのSysTick 再ロード値レジスタ（STK_LOAD）
0xE000E018	STK_VAL	RW	特権	予測不能	248 ページのSysTick 現在値レジスタ（STK_VAL）
0xE000E01C	STK_CALIB	RO	特権	0xC0000000	249 ページのSysTick 較正値レジスタ（STK_CALIB）

4.5.1 SysTick 制御およびステータス·レジスタ（STK_CTRL）

アドレス·オフセット：0x00

リセット値：0x0000 0000

必要な特権：特権

SysTick CTRL レジスタは、SysTick の機能を有効にします。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み															COUNTFLAG
															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み													CLKSOURCE	TICK INT	EN ABLE
													rw	rw	rw

ビット 31:17 予約済み、クリア状態を保つ必要があります。

ビット 16 COUNTFLAG：

最後の読み出しの後にタイマが 0 になった場合、1 を返します。

ビット 15:3 予約済み、クリア状態を保つ必要があります。

ビット 2 **CLKSOURCE**：クロック·ソースの選択

クロック·ソースを選択します。

0：AHB/8

1：プロセッサ·クロック（AHB）

ビット 1 **TICKINT**：SysTick 例外要求の有効化

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。

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。

注： 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。

ビット 0 **ENABLE**：カウンタ·イネーブル

カウンタを有効にします。ENABLE が 1 にセットされている場合、カウンタは LOAD 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。

0：カウンタは無効です。

1：カウンタは有効です。

4.5.2 SysTick 再ロード値レジスタ（STK_LOAD）

アドレス･オフセット：0x04

リセット値：0x0000 0000

必要な特権：特権

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み								RELOAD[23:16]							
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RELOAD[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ビット 31:24 予約済み、クリア状態を保つ必要があります。

ビット 23:0 **RELOAD**：RELOAD 値

LOAD レジスタは、カウンタが有効であり、それが 0 に到達したときに STK_VAL レジスタにロードする開始値を指定します。

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。

- 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。たとえば、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。
- N プロセッサ･クロック･サイクルの後に 1 回 SysTick 割込みを入れるには、RELOAD 値を N にします。たとえば、100 クロック･パルスの後に SysTick 割込みが必要な場合、RELOAD を 99 に設定します。

4.5.3 SysTick 現在値レジスタ（STK_VAL）

アドレス·オフセット：0x08

リセット値：0x0000 0000

必要な特権：特権

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み								CURRENT[23:16]							
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CURRENT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ビット 31:24 予約済み、クリア状態を保つ必要があります。

ビット 23:0 **CURRENT:** カウンタの現在値

VAL レジスタには、SysTick カウンタの現在値が格納されます。

読み出すと、SysTick カウンタの現在値が返されます。

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 STK_CTRL レジスタの COUNTFLAG ビットが 0 にクリアされます。

4.5.4 SysTick 較正値レジスタ（STK_CALIB）

アドレス･オフセット：0x0C

リセット値：0x0000000

必要な特権：特権

CALIB レジスタは、SysTick 較正プロパティを示します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NO REF	SKEW	予約済み						TENMS[23:16]							
r	r							r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TENMS[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

ビット 31	**NOREF**：NOREF フラグ。0 として読み出されます。独立した参照クロックが提供されていることを示します。このクロックの周波数は HCLK/8 です。
ビット 30	**SKEW**：SKEW フラグ。TENMS 値が正確かどうかを示します。1 として読み出されます。TENMS が不明なため、1ms の不正確なタイミングの較正値は不明です。これは、SysTick のソフトウェア･リアルタイム･クロックとしての適合性に影響を及ぼす可能性があります。
ビット 29:24	予約済み、クリア状態を保つ必要があります。
ビット 23:0	**TENMS[23:0]**：較正値。SysTick カウンタが外部クロックとして HCLK の最大値の 1/8 を使用している場合の較正値を示します。この値は製品に依存します。製品のリファレンス･マニュアルのSysTick 較正値のセクションを参照してください。HCLK が最大周波数でプログラムされている場合、SysTick 周期は 1ms 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。

4.5.5 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。

4.5.6 SysTick レジスタ･マップ

次の表に、SysTick レジスタ･マップとリセット値を示します。SysTick レジスタ･ブロックのベース･アドレスは、0xE000 E010 です。

表 54. SysTick レジスタ･マップとリセット値

オフセット	レジスタ	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	STK_CTRL	予約済み															COUNTFLAG	予約済み													CLKSOURCE	TICK INT	EN ABLE
	リセット値																0														1	0	0
0x04	STK_LOAD	予約済み								RELOAD[23:0]																							
	リセット値									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	STK_VAL	予約済み								CURRENT[23:0]																							
	リセット値									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	STK_CALIB	予約済み								TENMS[23:0]																							
	リセット値									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

4.6 浮動小数点ユニット（FPU）

Cortex-M4F FPU は、FPv4-SP 浮動小数点拡張機能を実装しています。

FPU は、単精度の加算、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 standard 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。

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 55 に、Cortex-M4F システム制御ブロック（SCB）の浮動小数点システム･レジスタを示します。FP 拡張の追加レジスタのベース･アドレスは、0xE000 ED00 です。

表 55. Cortex-M4F 浮動小数点システム･レジスタ

アドレス	名前	タイプ	リセット	説明
0xE000ED88	CPACR	RW	0x00000000	252 ページのセクション 4.6.1: コプロセッサ･アクセス制御レジスタ（CPACR）
0xE000EF34	FPCCR	RW	0xC0000000	252 ページのセクション 4.6.2: 浮動小数点コンテキスト制御レジスタ（FPCCR）
0xE000EF38	FPCAR	RW	-	254 ページのセクション 4.6.3: 浮動小数点コンテキスト･アドレス･レジスタ（FPCAR）
0xE000EF3C	FPDSCR	RW	0x00000000	256 ページのセクション 4.6.5: 浮動小数点デフォルト･ステータス制御レジスタ（FPDSCR）
-	FPSCR	RW	-	254 ページのセクション 4.6.4: 浮動小数点ステータス制御レジスタ（FPSCR）

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。

注： **IEEE 規格と浮動小数点演算（IEEE 754）の詳細については、アプリケーション･ノート AN4044（当社 Web サイト www.st.com から入手可能）を参照してください。**

4.6.1 コプロセッサ·アクセス制御レジスタ（CPACR）

アドレス·オフセット（SCB から）: 0x88

リセット値：0x0000000

必要な特権：特権

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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み								CP11		CP10		予約済み			
								rw		rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み															

ビット 31:24 予約済み。0 として読み出され、書込みは無視します。

ビット 23:20 **CPn**：n 値 10 および 11にビット [2n+1:2n] が対応。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。

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。

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。

0b10：予約済み。アクセスの結果は予測不能です。

0b11：フル·アクセス。

ビット 19:0 予約済み。0 として読み出され、書込みは無視します。

4.6.2 浮動小数点コンテキスト制御レジスタ（FPCCR）

アドレス·オフセット：0x04

リセット値：0xC000000

必要な特権：特権

FPCCR レジスタは、FPU 制御データをセットまたは返します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ASPEN	LSPEN	予約済み													
rw	rw														

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み							MONRDY	予約済み	BFRDY	MMRDY	HFRDY	THREAD	予約済み	USER	LSPACT
							rw		rw	rw	rw	rw		rw	rw

ビット 31 **ASPEN**：浮動小数点命令の実行において CONTROL<2> 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。

0：浮動小数点命令の実行において CONTROL<2> 設定を無効にします。

1：浮動小数点命令の実行において CONTROL<2> 設定を有効にします。

ビット 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。

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。

ビット 29:9 予約済み。

ビット 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。

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。

ビット 7 予約済み。

ビット 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。

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。

ビット 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。

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。

ビット 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。

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。

ビット 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。

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。

ビット 2 予約済み。

ビット 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。

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。

ビット 1 **LSPACT** :

0 : レイジーな状態保持はアクティブではありません。

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･フレームは割当て済みですが、そこへの状態の保存は延期されています。

4.6.3 浮動小数点コンテキスト·アドレス·レジスタ（FPCAR）

アドレス·オフセット：0x08

リセット値：0x0000000

必要な特権：特権

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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADDRESS[31:16]															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDRESS[15:3]													予約済み		
rw															

ビット 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。

ビット 2:0 予約済み。0 として読み出され、書込みは無視します。

4.6.4 浮動小数点ステータス制御レジスタ（FPSCR）

アドレス·オフセット：未割当て

リセット値：0x0000000

必要な特権：特権

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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
N	Z	C	V	予約済み	AHP	DN	FZ	RMode		予約済み					
rw	rw	rw	rw		rw	rw	rw	rw	rw						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み								IDC	予約済み		IXC	UFC	OFC	DZC	IOC
								rw			rw	rw	rw	rw	rw

ビット 31 **N**：ネガティブ条件コード·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。結果の詳細については、表 56 を参照してください。

0：演算結果は正、0、より大きい、または等しいでした。

1：演算結果は負またはより小さいでした。

ビット 30 **Z**：ゼロ条件コード·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。結果の詳細については、表 56 を参照してください。

0：演算結果はゼロではありませんでした。

1：演算結果はゼロでした。

ビット 29 **C**：キャリー条件コード·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。結果の詳細については、表 56 を参照してください。

0：加算によってキャリー·ビットが発生しなかったか、減算によってボロー·ビットが発生しました。

1：加算によってキャリー·ビットが発生したか、減算によってボロー·ビットが発生しませんでした。

ビット 28 **V**：オーバーフロー条件コード·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。結果の詳細については、表 56 を参照してください。

0：演算でオーバーフローは発生しませんでした。

1：演算でオーバーフローが発生しました。

ビット 27 予約済み。

ビット 26 **AHP**：オルタネィティブ半精度制御ビット。

0：IEEE 半精度形式が選択されています。

1：オルタネィティブ半精度形式が選択されています。

ビット 25 **DN**：デフォルトの NaN モード制御ビット。

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。

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。

ビット 24 **FZ**：Flush-to-zero モード制御ビット。

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。

1：Flush-to-zero モードは有効です。

ビット 23:22 **RMode**：丸めモード制御フィールド。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。

0b00：近似値への丸め（RN）モード

0b01：プラス無限大への丸め（RP）モード

0b10：マイナス無限大への丸め（RM）モード

0b11：0 への丸め（RZ）モード

ビット 21:8 予約済み。

ビット 7 **IDC**：入力非正規化累積例外ビット。浮動小数点例外の累積例外ビットです。

1：最後に 0 が書き込まれた後に、対応する例外が発生したことを示します。

ビット 6:5 予約済み

ビット 4 **IXC**：不正確累積例外ビット。浮動小数点例外の累積例外ビットです。

1：最後に 0 が書き込まれた後に、対応する例外が発生したことを示します。

ビット 3 **UFC**：アンダーフロー累積例外ビット。浮動小数点例外の累積例外ビットです。

1：最後に 0 が書き込まれた後に、対応する例外が発生したことを示します。

ビット 2 **OFC**：オーバーフロー累積例外ビット。浮動小数点例外の累積例外ビットです。

1：最後に 0 が書き込まれた後に、対応する例外が発生したことを示します。

ビット 1 **DZC**：0 による除算累積例外ビット。浮動小数点例外の累積例外ビットです。1：最後に 0 が書き込まれた後に、対応する例外が発生したことを示します。

ビット 0 **IOC**：無効演算累積例外ビット。浮動小数点例外の累積例外ビットです。1：最後に 0 が書き込まれた後に、対応する例外が発生したことを示します。

表 56. 浮動小数点の比較の条件フラグに対する影響

比較結果	N	Z	C	V
等しい	0	1	1	0
より小さい	1	0	0	0
より大きい	0	0	1	0
順序付けなし	0	0	1	1

4.6.5 浮動小数点デフォルト·ステータス制御レジスタ（FPDSCR）

アドレス·オフセット：0x0C

リセット値：0x0000000

必要な特権：特権

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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済み					AHP	DN	FZ	RMode		予約済み					
					rw	rw	rw	rw	rw						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済み															

ビット 31:27 予約済み、クリア状態を保つ必要があります。

ビット 26 **AHP**：FPSCR.AHP のデフォルト値

ビット 25 **DN**：FPSCR.DN のデフォルト値

ビット 24 **FZ**：FPSCR.FZ のデフォルト値

ビット 23:22 **RMode**：FPSCR.RMode のデフォルト値

ビット 21:0 予約済み、クリア状態を保つ必要があります。

4.6.6 FPU の有効化

FPU は、リセットにより無効になります。何らかの浮動小数点命令を使用する前に、FPU を有効にする必要があります。

特権モードとユーザ·モードの両方で FPU を有効にするコード·シーケンスの例を次に示します。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

```
                  ; CPACR is located at address 0xE000ED88
LDR.W   R0, =0xE000ED88
                  ; Read CPACR
LDR     R1, [R0]
                  ; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR     R1, R1, #(0xF << 20)
                  ; Write back the modified value to the CPACR
STR     R1, [R0]; wait for store to complete
DSB
; reset pipeline now the FPU is enabled
ISB
```

4.6.7 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。

システム設定コントローラ（SYSCFG）ではマスク·ビットも提供されており、各 FPU フラグ割込み生成を個別に有効化／無効化できます。

注： **STM32F4xx デバイスには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、このデバイスの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）をダミー読み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。

  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
}
```

5 改版履歴

表 57. 文書改版履歴

日付	版	変更点
2012 年 2 月 20 日	1	初版発行
2012 年 7 月 9 日	2	セクション 4.6.2: 浮動小数点コンテキスト制御レジスタ（FPCCR）のリセット値を変更。 Table 1: Applicable products を追加
2012 年 9 月 4 日	3	STM32F3xxx Cortex-M4 プロセッサに関する情報を追加。 Table 1: Applicable products に部品番号を追加。 はじめにに関連参照文献を追加。 セクション 1.3.3: Cortex-M4 プロセッサの特徴と利点の概要の「IEEE754準拠の単精度FPU」の箇条書きを変更。 セクション 2.5.3: 外部イベント入力／拡張割込みおよびイベント入力に拡張割込み／イベント･コントローラの情報を追加。 セクション 4.3: ネスト化されたベクタ割込みコントローラ（NVIC）の最初の「割込み」の箇条書きを変更。 セクション 4.4.7: 設定および制御レジスタ（CCR）の古いリセット値の情報および表 51: システム･フォールト･ハンドラ優先度フィールドの 0x14 オフセットを削除。 セクション 4.6: 浮動小数点ユニット（FPU）に IEEE 754 に関する注を追加。
2014 年 5 月 12 日	4	参照文献を更新。 セクション 4.4.1: 補助制御レジスタ（ACTLR）を更新。 セクション 4.5.1: SysTick 制御およびステータス･レジスタ（STK_CTRL）を更新。
2016 年 4 月 18 日	5	更新： – はじめに – 参照文献 – セクション 2.5.3: 外部イベント入力／拡張割込みおよびイベント入力 – セクション 4.6.7: FPU 例外割込みの有効化とクリア – 表 50: 優先度のグループ化 削除： – Table 1: Applicable products
2017 年 10 月 2 日	6	ドキュメントの範囲に STM32L4+ シリーズを追加（影響を受けるのはドキュメントのタイトルと表紙のみ）。 表 48: NVIC レジスタ･マップとリセット値を更新。

表 58. 日本語版文書文書改版履歴

日付	版	変更点
2018 年 12 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）

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。