PM0223
プログラミング·マニュアル

Cortex®-M0+ プログラミング·マニュアル（STM32L0、STM32G0、STM32WL および STM32WB シリーズ用）

概要

このプログラミング·マニュアルには、アプリケーションやシステムレベルのソフトウェアの開発者向けの情報が記載されています。本書では、STM32L0、STM32G0、STM32WL および STM32WB シリーズ MCU で使用される Cortex®-M0+ 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。

Cortex®-M0+ は、マイクロコントローラに統合するために設計された高性能の 32 ビット·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·オプション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

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®[a] 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。

italic	重要な注意事項を強調し、特別な用語を紹介し、内部の相互参照と引用を示します。
bold	メニュー名などのユーザ·インタフェース要素の強調に太字を使用します。信号名も太字で示します。適宜、記述リスト内の用語にも使用します。
`monospace`	コマンド、ファイル名、プログラム名、ソース·コードなどキーボードから入力可能なテキストを表します。
`monospace`	コマンドやオプション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 italic`	特定の値に置き換えられる引数は等幅フォントのテキストで表されます。
`monospace bold`	サンプル·コード以外で使用される言語キーワードを示します。
< および >	コードまたはコード·フラグメント内でアセンブラ構文の置き換えが可能な用語を囲みます。例： LDRSB<cond> <Rt>, [<Rn>, #<offset>]

a. Arm は、米国内およびその他の地域にある Arm Limited 社（またはその子会社）の登録商標です。

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。

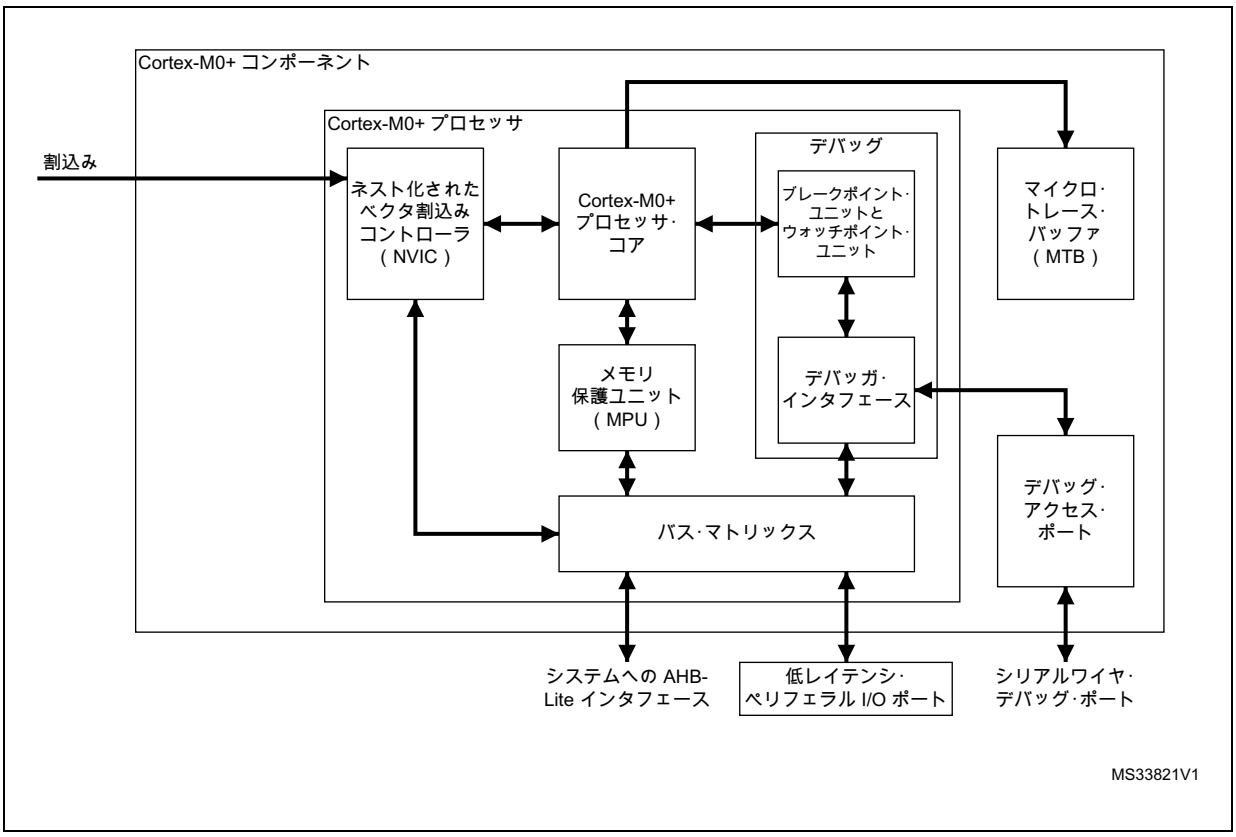
読出し／書込み（rw）	これらのビットは、ソフトウェアによる読出しと書込みができます。
読出し専用（r）	これらのビットは、ソフトウェアによる読出しのみが可能です。
書込み専用（w）	このビットは、ソフトウェアによる書込みのみが可能です。 ビットを読み出すと、リセット値が返されます。
読出し／クリア（rc_w）	このビットは、ソフトウェアによって読み出すことができ、任意の値を書き込むことによってクリアできます。
読出し/クリア（rc_w1）	このビットは、ソフトウェアによって読み出すことができ、“1”を書き込むことによってクリアできます。 ”0”を書き込んでも、ビットの値は変化しません。
読出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

1.3 Cortex-M0+ プロセッサおよびコア·ペリフェラルについて

Cortex-M0+ プロセッサは、幅広い組込みアプリケーション向けに設計されたエントリレベルの 32 ビット Arm® Cortex 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。

- 学習とプログラムが容易なシンプルなアーキテクチャ
- 超低電力、エネルギー効率に優れた動作
- 卓越したコード密度
- 高性能な決定性の割込み処理
- Cortex-M プロセッサ·ファミリとの上位互換
- プラットフォームに対する、オプションの内蔵メモリ保護ユニット（MPU）による堅牢なセキュリティ

図 1. Cortex-M0+ の実装



Cortex-M0+ プロセッサは、フォン·ノイマン型 2 段パイプライン·アーキテクチャを採用し、面積と電力を高度に最適化した 32 ビット·プロセッサ·コア上に構築されています。コンパクトで強力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エネルギー効率を実現し、シングルサイクル乗算器など、ハイエンドの処理ハードウェアを搭載しています。

Cortex-M0+ プロセッサは、16 ビット Thumb® 命令セットをベースにし、Thumb-2 テクノロジを採用した ARMv6-M アーキテクチャを実装しています。これにより、他の 8 ビットおよび 16 ビット·マイクロコントローラより高いコード密度を保ちつつ、最新の 32 ビット·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-M0+ プロセッサは、設定可能なネスト化されたベクタ割込みコントローラ（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 の特徴は次のとおりです。

- ノンマスカブル割込み（NMI）機能。
- ゼロジッタ割込みオプション。
- 4 段階の割込み優先度レベル。

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中止および再開する機能です。割込みハンドラはアセンブラ･ラッパー･コード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 の切替え時のオーバーヘッドを著しく短縮します。

NVIC は低電力設計の最適化のために、スリープ･モードを搭載していますが、これにはデバイス全体の迅速なパワーダウンを可能にするディープ･スリープ機能も含まれます。

1.3.1 システムレベル･インタフェース

Cortex-M0+ 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® テクノロジによるシングル･システムレベルのインタフェースを備えています。

Cortex-M0+ プロセッサは、きめ細かくメモリを制御するオプションの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使用し、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、車載システム用途など多くの組込み用途に不可欠な要件になっています。

1.3.2 設定可能な統合デバッグ機能

Cortex-M0+ プロセッサは、ハードウェアのブレークポイントとウォッチポイントのオプションを拡大した、完全なハードウェア･デバッグ･ソリューションを実装します。これにより、マイクロコントローラや他の小型パッケージ･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> を介して、プロセッサ、メモリ、およびペリフェラルに対するシステム可視性が高まります。

1.3.3 Cortex-M0+ プロセッサの特徴の概要

- Thumb-2 テクノロジを使用した Thumb 命令セット
- 32 ビット性能による高いコード密度
- ユーザ･モードと特権モードの実行
- ツールとバイナリの Cortex-M プロセッサ･ファミリとの上位互換
- 超低電力 SLEEP モードの搭載
- 効率的なコード実行によりプロセッサ･クロックの低速化と SLEEP 期間の増加が可能
- シングルサイクル 32 ビット･ハードウェア乗算器
- ゼロジッタ割込み処理
- 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）
- 低レイテンシで、高速なペリフェラル I/O ポート
- ベクタ･テーブル･オフセット･レジスタ
- 幅広いデバッグ機能

1.3.4 Cortex-M0+ のコア･ペリフェラル

コア･ペリフェラルは、以下のとおりです。

ネスト化されたベクタ割込みコントローラ（NVIC）

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。

I/O ポート

I/O ポートは、緊密に結合されたペリフェラルに対して、シングルサイクルのロードとストアを提供します。

2 Cortex-M0+ 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-M0+ のプログラマ·モデルについて説明します。個々のコア·レジスタの説明に加えて、ソフトウェアの実行、およびスタックに対するプロセッサのモードや、特権レベルの情報も提示します。

2.1.1 ソフトウェアの実行に対するプロセッサのモードと特権レベル

このプロセッサには、以下のモードがあります。

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

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

ソフトウェア実行には、次の特権レベルがあります。

非特権　ソフトウェアには以下の制限があります。

- MSR および MRS 命令を使用したシステム·レジスタへのアクセスが制限され、CPS 命令を使用した割込みのマスクはできません。
- システム·タイマ、NVIC、システム制御ブロックにはアクセスできません。
- メモリやペリフェラルへのアクセスが制限される場合があります。

非特権ソフトウェアは非特権レベルで実行されます。

特権　ソフトウェアは、すべての命令を使用できるとともにすべてのリソースへのアクセスが可能です。

特権ソフトウェアは特権レベルで実行されます。

スレッド·モードの場合、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（18 ページのCONTROL レジスタを参照）。ハンドラ·モードの場合、ソフトウェア実行には常に特権が与えられます。

スレッド·モードでは、特権ソフトウェアのみが、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。非特権ソフトウェアは、特権ソフトウェアに制御を移すためのスーパバイザ·コールを実行するために、SVC 命令を使用できます。

2.1.2 スタック

このプロセッサは完全降順スタックを使用します。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（15 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（18 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オプションの概要

プロセッサ·モード	実行対象	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. 18 ページのCONTROL レジスタを参照

2.1.3 コア·レジスタ

プロセッサのコア·レジスタを次に説明します。

図 2. プロセッサのコア·レジスタ

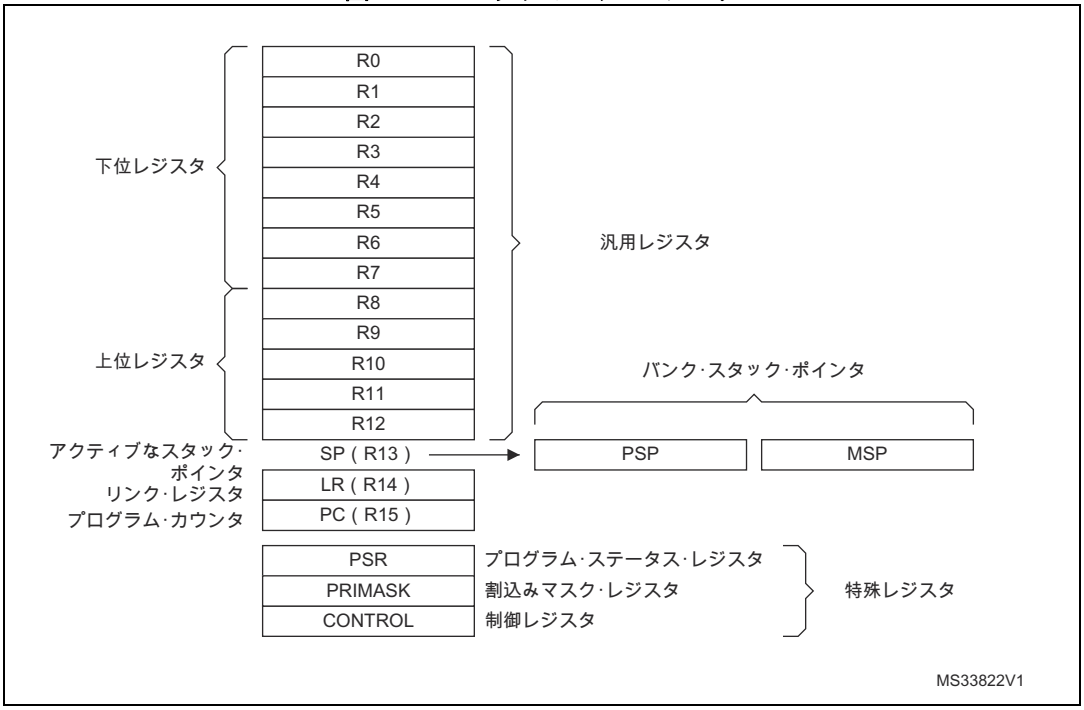


表 2. コア·レジスタ·セットの概要

名前	タイプ(1)	リセット値	説明
R0-R12	RW	不明	15 ページの汎用レジスタを参照してください。
MSP	RW	説明を参照	15 ページのスタック·ポインタを参照してください。
PSP	RW	不明	15 ページのスタック·ポインタ
LR	RW	不明	15 ページのリンク·レジスタ
PC	RW	説明を参照	15 ページのプログラム·カウンタ
PSR	RW	不明(2)	15 ページのプログラム·ステータス·レジスタ

表 2. コア·レジスタ·セットの概要 （続き）

APSR	RW	不明	16 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	RO	0x00000000	17 ページの割込みプログラム·ステータス·レジスタ
EPSR	RO	不明	17 ページの例外プログラム·ステータス·レジスタ
PRIMASK	RW	0x00000000	18 ページの優先度マスク·レジスタ
CONTROL	RW	0x00000000	18 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. ビット [24] は T ビットであり、リセット·ベクタのビット [0] からロードされます。

汎用レジスタ

R0 ~ R12 は、データ操作のための 32 ビット汎用レジスタです。

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、関数呼出し、および例外の復帰情報を格納します。リセットしたときの LR の値は不定です。

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。

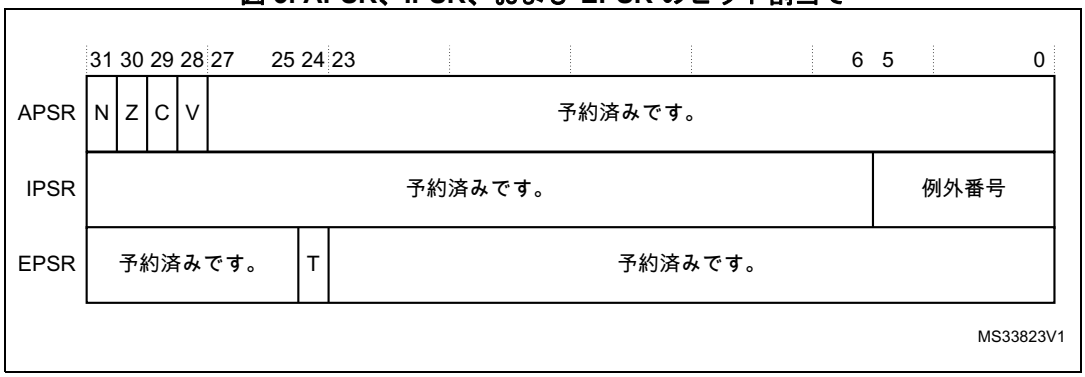
プログラム·ステータス·レジスタ

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、32 ビット PSR 内の互いに重ならないビットフィールドとして割り当てられます。PSR ビット割当てを次に示します。

図 3. APSR、IPSR、および EPSR のビット割当て



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

- MRS 命令で PSR を使用して、3 つすべてのレジスタを読み出します。
- `MSR` 命令で `APSR` を使用して、APSR に書き込みます。

PSR の組合せと属性は次のとおりです。

表 3. PSR レジスタの組合せ

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

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

プログラム･ステータス･レジスタへのアクセス方法の詳細については、75 ページのMRSおよび76 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、14 ページの表 2のレジスタ概要を参照してください。ビット割当てを次に示します。

表 4. APSR ビット割当て

ビット	名前	説明
[31]	N	ネガティブ･フラグ。
[30]	Z	ゼロ･フラグ。
[29]	C	キャリーまたはボロー･フラグ。
[28]	V	オーバーフロー･フラグ。
[27:0]	-	予約済み。

APSR のネガティブ、ゼロ、キャリーまたはボロー、およびオーバーフローの各フラグについての詳細は、44 ページの条件フラグを参照してください。

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）の例外の番号が格納されます。属性については、14 ページの表 2のレジスタ概要を参照してください。ビット割当てを次に示します。

表 5. IPSR のビット割当て

ビット	名前	機能
[31:6]	-	予約済みです。
[5:0]	例外番号	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 = HardFault 4-10 = 予約済み 11 = SVCall 12、13 = 予約済み 14 = PendSV 15 = SysTick | 予約済み 16 = IRQ0 . . 47 = IRQ31 48-63 = 予約済み 詳細は、27 ページの例外のタイプを参照してください。

例外プログラム・ステータス・レジスタ

EPSR は、Thumb 状態ビットを格納します。

EPSR の属性については、14 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 のビット割当て

ビット	名前	機能
[31:25]	-	予約済み。
[24]	T	Thumb 状態ビット
[23:0]	-	予約済み。

アプリケーション・ソフトウェアから MRS 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。MRS 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、フォールトの原因を特定することができます。31 ページの例外の開始と復帰を参照してください。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 HardFault またはロックアップが発生します。詳細については、34ページの2.4.1：ロックアップを参照してください。

中断可能で中断後からリスタート可能な命令

中断可能で中断後からリスタート可能な命令は、LDM と STM、PUSH、POP、および MULS です。これらの命令のいずれかの実行中に割込みが発生した場合、プロセッサは命令の実行を中止します。割込み処理の後、プロセッサは命令の実行を最初からリスタートします。

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またはアトミック性を必要とするコード・シーケンスに影響を与える可能性がある例外を無効にします。

例外を有効にする、または再度有効にするには、MSR および MRS 命令を使用するか、CPS 命令を使用して、PRIMASK の値を変更します。詳細については、75ページの3.7.6：MRS、76ページの3.7.7：MSR、および71ページの3.7.2：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、14 ページの表 2のレジスタ概要を参照してください。ビット割当てを次に示します。

表 7. PRIMASK レジスタのビット割当て

ビット	名前	機能
[31:1]	-	予約済み。
[0]	PM	優先順位の設定が可能な割込みマスク： 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。

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ます。属性については、14 ページの表 2のレジスタ概要を参照してください。ビット割当てを次に示します。

図 4. 制御ビット割当て

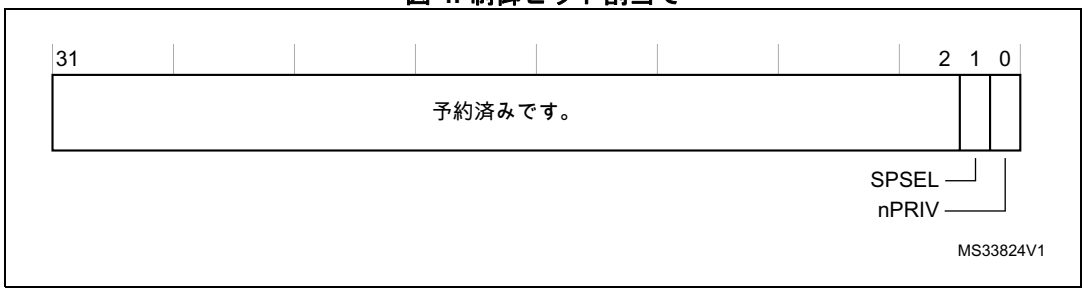


表 8. 制御レジスタのビット割当て

ビット	名前	機能
[31:2]	-	予約済み。
[1]	SPSEL	現在のスタックを定義します。 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。
[0]	nPRIV	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自動的に更新されます。

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、`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（75ページの3.7.6：MRSを参照）。

注： **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。74ページの3.7.5：ISBを参照してください。**

2.1.4 例外と割込み

Cortex-M0+ 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。割込みまたは例外は、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、32 ページの例外の開始および 33 ページの例外からの復帰を参照してください。

NVIC レジスタは割込み処理を制御します。詳細については、83ページの4.2：ネスト化されたベクタ割込みコントローラを参照してください。

2.1.5 データ型

プロセッサは、以下のとおりです。

- 以下のデータ型をサポートします。
  - 32 ビット·ワード
  - 16 ビット·ハーフワード
  - 8 ビット·バイト
- すべてのデータ·メモリ·アクセスを、リトルエンディアンまたはビッグエンディアンとして管理します。命令メモリとプライベート·ペリフェラル·バス（PPB）へのアクセスは、常にリトルエンディアンです。詳細については、22ページの2.2.1：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·インタフェース標準

Arm® では、Cortex-M0+ マイクロコントローラのプログラミング向けに、Cortex マイクロコントローラ·ソフトウェア·インタフェース標準（CMSIS）を制定しています。CMSIS は、デバイス·ドライバ·ライブラリ内に統合されています。CMSIS では、Cortex-M0+ 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

CMSIS には、Cortex-M0+ 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。また、TCP/IP スタックとフラッシュ·ファイル·システムからなるミドルウェア·コンポーネント向けの、オプションのインタフェースも含まれています。

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。

注： **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。**

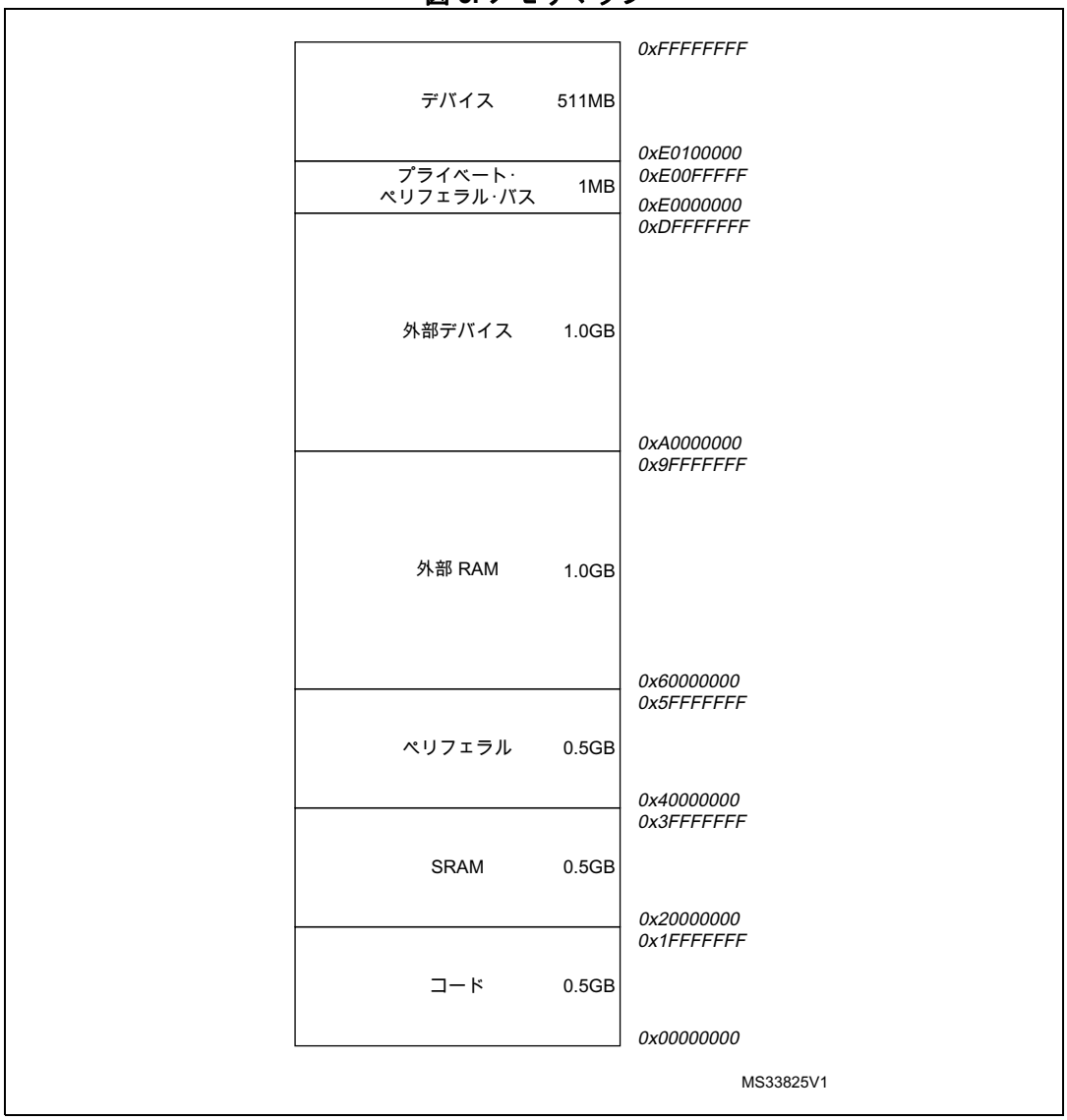
以降のセクションでは、CMSIS の詳細について説明します。

- 36ページの2.5.4：電源管理に関するプログラミングのヒント
- 40ページの3.2：組込み関数
- 83ページの4.2.1：CMSIS を使用した Cortex-M0+ NVIC レジスタへのアクセス
- 88 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·マップおよびメモリ·アクセスの動作について説明します。プロセッサには、最大 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。メモリ·マップは、以下のとおりです。

図 5. メモリマップ



プロセッサでは、コア·ペリフェラルのレジスタ向けに、プライベート·ペリフェラル·バス（PPB）のアドレス範囲の領域を確保しています（10ページの1.3：Cortex-M0+ プロセッサおよびコア·ペリフェラルについてを参照）。

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、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。

共有可能	共有可能なメモリ領域に対して、メモリ·システムは、複数のバス·マスタを持つシステム（たとえば、DMA コントローラを持つプロセッサ）内のバス·マスタ間のデータ同期を行います。 Strongly-ordered メモリは常に共有可能です。 複数のバス·マスタが共有不可能なメモリ領域にアクセスできる場合、ソフトウェアは、バス·マスタ間のデータの一貫性を確保する必要があります。 <この説明は、メモリが複数のプロセッサ間で共有されているシステムでデバイスが使用される可能性がある場合にのみ必要です。>
実行不可（XN）	プロセッサが命令へのアクセスを禁止することを意味します。メモリの XN 領域からフェッチされた命令を実行した場合、HardFault 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、順序の変更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（22ページの2.2.2 : メモリ·システムでのメモリ·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 9. 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	-	<	<	<

MS33826V1

1. - 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。

   < 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 10. メモリ·アクセスの動作(1)

アドレス範囲	メモリ領域	メモリ·タイプ	XN	説明
0x00000000 - 0x1FFFFFFF	コード	ノーマル	-	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	-	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。
0x40000000 - 0x5FFFFFFF	ペリフェラル	デバイス	XN	外部デバイス·メモリ。
0x60000000 - 0x9FFFFFFF	外部 RAM	ノーマル	-	データの実行可能領域。
0xA0000000 - 0xDFFFFFFF	外部デバイス	デバイス	XN	外部デバイス·メモリ。
0xE0000000 - 0xE00FFFFF	専用ペリフェラル·バス	Strongly- ordered	XN	この領域には、NVIC、システム·タイマ、システム制御ブロックが含まれます。 この領域ではワード·アクセスのみを使用できます。

1. 詳細については、22 ページのメモリの領域、タイプ、および属性を参照してください。

コード、SRAM、外部 RAMの領域にプログラムを保持できます。

MPU は、このセクションに示したデフォルトのメモリ·アクセス動作を上書きできます。詳細については、100ページの4.5 : メモリ保護ユニットを参照してください。

2.2.4 キャッシュと共有メモリに対する追加のメモリ·アクセス制約

システムにキャッシュまたは共有メモリが含まれている場合、表 11 に示すように、一部のメモリ領域には追加のアクセス制約があり、一部の領域は分割されています。

表 11. メモリ領域の共有可能性とキャッシュ·ポリシー

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

1. 詳細は、22ページの2.2.1：メモリの領域、タイプ、および属性を参照してください。
2. WT = ライトスルー、書込み割当てなしWBWA = ライトバック、書込み割当て

2.2.5 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。この原因は、以下のとおりです。

- プロセッサは、命令シーケンスの動作に影響を与えない限り、効率向上のために一部のメモリ·アクセスの順序を入れ換える場合がある。
- メモリ·マップ内のメモリまたはデバイスのウェイト·ステートが異なる場合がある。
- 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。

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

DMB　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。72 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。73 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。74 ページのISBを参照してください。

メモリ･バリア命令の使用例を次に示します。

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･マップの切替え　システムがメモリ･マップの切替えメカニズムを備えている場合、メモリ･マップの切替え後に 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。

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

VTOR プログラミング　プログラムが VTOR の値を更新する場合は、DMB 命令を使用して、新しいベクタ･テーブルが後続の例外に使用されるようにします。

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。

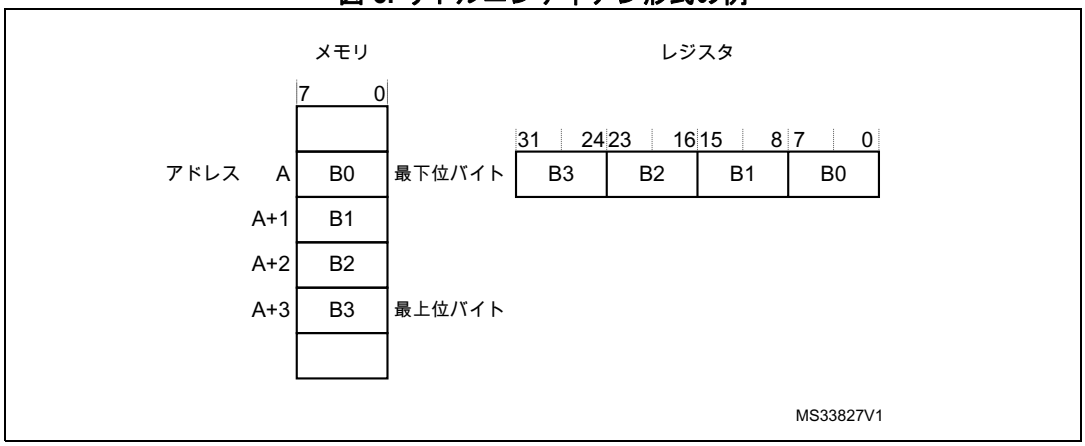
2.2.6 メモリのエンディアン形式

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は、データのワードがどのようにメモリに格納されるかを示したものです。

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（lsbyte）を一番小さい番号のバイトにストアし、最上位バイト（msbyte）を一番大きい番号のバイトにストアします。例：

図 6. リトルエンディアン形式の例



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。

HardFault HardFault は、通常または例外の処理中のエラーによって発生する例外です。HardFault の優先度は -1 に固定されています。これは、設定可能な優先度を持つどの例外よりも優先度が高いことを意味します。

SVCall 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 12. さまざまな例外タイプのプロパティ

例外番号(1)	IRQ 番号(1)	例外のタイプ	優先順位	ベクタ･アドレス(2)	アクティブ化
1	-	リセット	-3、最高	0x00000004	非同期
2	-14	NMI	-2	0x00000008	非同期
3	-13	HardFault	-1	0x0000000C	同期
4 ～ 10	-	予約済みです。	-	-	-
11	-5	SVCall	設定可能(3)	0x0000002C	同期
12 ～ 13	-	予約済みです。	-	-	-
14	-2	PendSV	設定可能(3)	0x00000038	非同期
15	-1	SysTick	設定可能(3)	0x0000003C	非同期
15	-	予約済みです。	-	-	-
16 以上	0 以上	割込み（IRQ）	設定可能(3)	0x00000040 以上 (4)	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。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（17 ページの割込みプログラム･ステータス･レジスタを参照）。
2. 詳細については、30ページの図 7.：ベクタ･テーブルを参照してください。
3. 86ページの4.2.6：割込み優先度のレジスタを参照
4. 4 ずつ増加します。

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

特権ソフトウェアは、28 ページの表 12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（85ページの4.2.3：割込みクリア･イネーブル･レジスタを参照）。

HardFault の詳細については、34ページの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

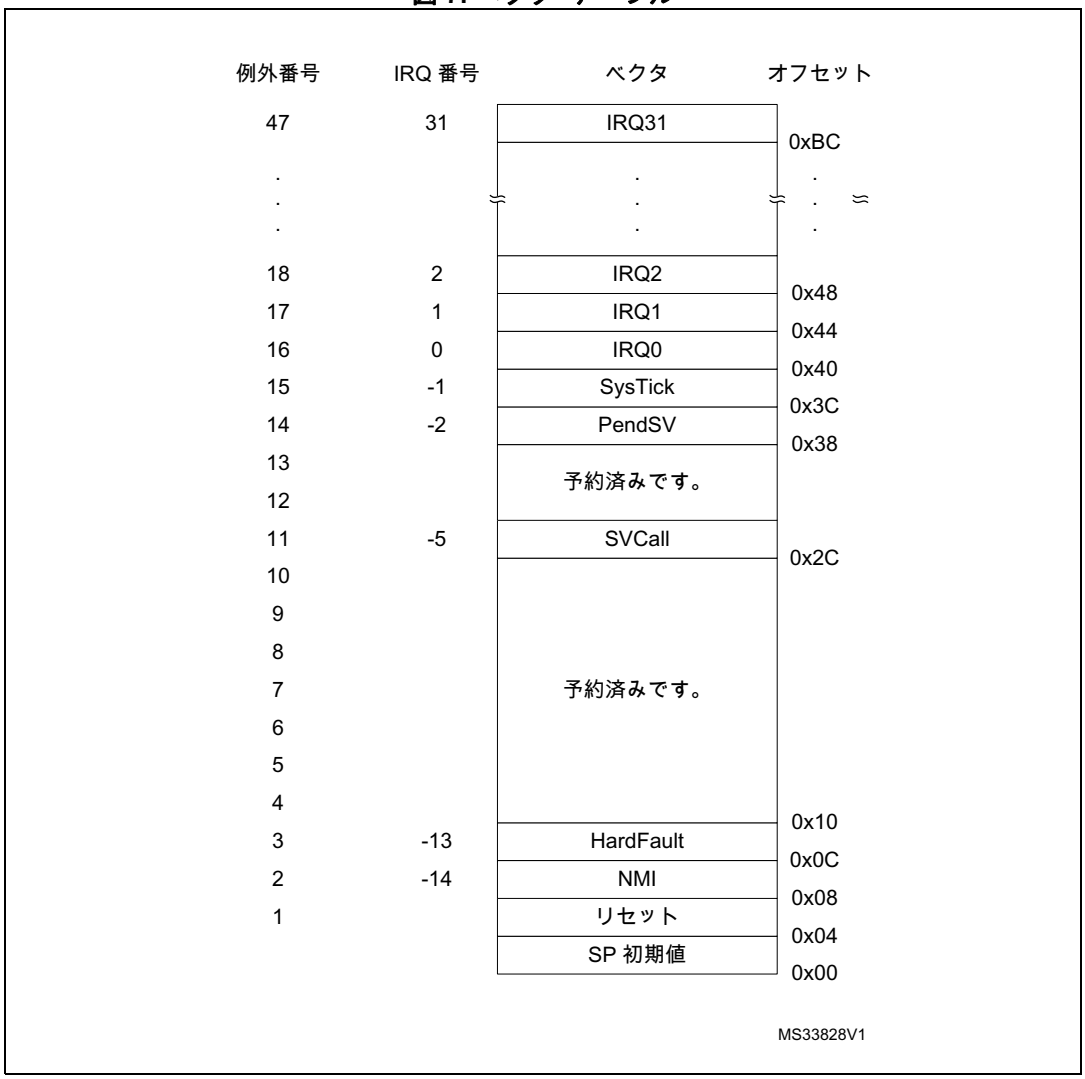
プロセッサは、以下を使用して例外を処理します。

割込みサービス･ルーチン（ISR）	IRQ0 ~ IRQ31 の割込みは、ISR が処理する例外です。
フォールト･ハンドラ	HardFault は、フォールト･ハンドラによって処理される唯一の例外です。
システム･ハンドラ	NMI、PendSV、SVCall、SysTick、および HardFault 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。30 ページの図 7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 コードに書き込まれたことを示す、1 である必要があります。

図 7. 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、ベクタ·テーブルのサイズと TBLOFF 設定の粒度に関してベクタ·テーブルの開始アドレスを別のメモリ位置に再配置するために VTOR に書き込むことができます（セクション 4.3.4：ベクタ·テーブル·オフセット·レジスタを参照）。

2.3.5 例外の優先度

28 ページの表 12に示すように、すべての例外には次のように優先度が関連付けられています。

- 優先度の値が小さいほど、優先順位が高いことを示します。
- リセット、HardFault、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。

- 95ページの4.3.8：システム･ハンドラ優先度レジスタ
- 86ページの4.2.6：割込み優先度のレジスタ

注： **設定可能な優先度の値の範囲は、0 ~ 192（64 刻み）です。固定された負の優先度値を持つリセット、HardFault、および NMI 例外は、常に他のどの例外よりも優先度が高くなります。**

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 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。

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、32 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、33 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。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。

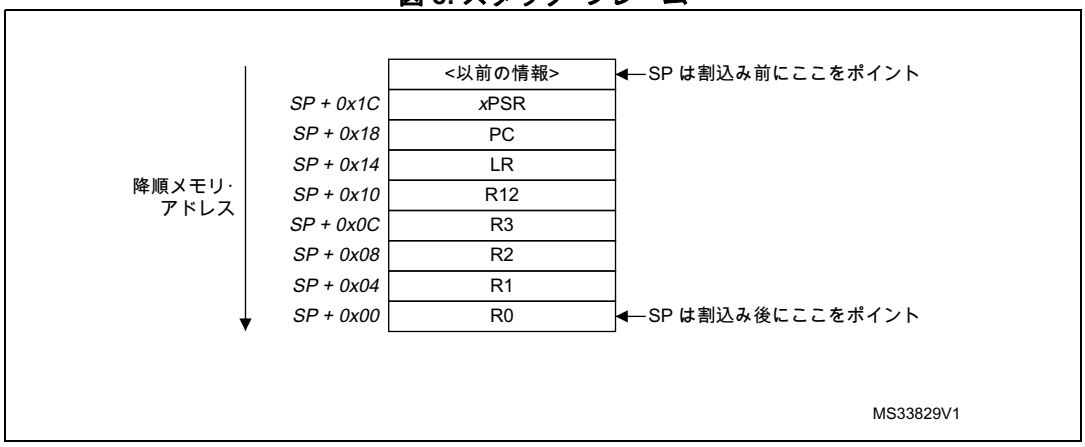
- プロセッサがスレッド･モードである。
- 新しい例外の優先度が処理されている例外より高い（この場合は新しい例外が処理されている例外を横取りする）。

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（18 ページの例外マスク･レジスタを参照）。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。スタック･フレームには、次の情報が含まれています。

図 8. スタック･フレーム



スタッキングの直後のスタック･ポインタは、スタック･フレームの最下位アドレスを示しています。スタック･フレームはダブルワード･アドレスにアライメントされます。

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、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 PC を EXC_RETURN 値に設定しようとすると発生します。

- A PC をロードする POP 命令。
- B 任意のレジスタを使用する PBX 命令。

プロセッサは、例外の開始時に EXC_RETURN 値を LR に保存します。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。EXC_RETURN 値のビット [31:4] は 0xFFFFFFF です。プロセッサがこのパターンに一致する値を PC にロードすると、その操作が通常の分岐操作ではなく、例外が完了したことを検出します。その結果、例外復帰のシーケンスが開始されます。EXC_RETURN 値のビット [3:0] は、33 ページの表 13に示すように、必要な復帰スタックとプロセッサ·モードを示します。

表 13. 例外からの復帰動作

EXC_RETURN	説明
0xFFFFFFF1	ハンドラ·モードに戻ります。 例外からの復帰では、メイン·スタックから状態を取得します。 実行では復帰後に MSP を使用します。
0xFFFFFFF9	スレッド·モードに戻ります。 例外からの復帰では、MSP から状態を取得します。 実行では復帰後に MSP を使用します。
0xFFFFFFFD	スレッド·モードに戻ります。 例外からの復帰では、PSP から状態を取得します。 実行では復帰後に PSP を使用します。
その他のすべての値	予約済み。

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（27ページの2.3：例外モデルを参照）。フォールトはすべて結果として、HardFault 例外を取得するか、NMI または HardFault ハンドラで発生した場合はロックアップを引き起こします。フォールトが発生するのは以下の場合です。

- SVCall 以上の優先度での `SVC` 命令の実行。
- デバッガがアタッチされていない状態での `BKPT` 命令の実行。
- ロードまたはストアでシステムが生成したバス・エラー。
- XN メモリ・アドレスからの命令の実行。
- システムがバス・フォールトを生成した場所からの命令の実行。
- ベクタ・フェッチでシステムが生成したバス・エラー。
- 未定義の命令の実行。
- T ビットが以前に 0 にクリアされた結果として Thumb 状態にない場合の命令の実行。
- アラインされていないアドレスへのロードまたはストアの試行。
- 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る MPU フォールト。

注： **固定優先度の HardFault ハンドラを横取りできるのは、リセットと NMI のみです。HardFault は、リセット、NMI、または別の HardFault を除くすべての例外を横取りできます。**

2.4.1 ロックアップ

NMI ハンドラまたは HardFault ハンドラの実行時にフォールトが発生した場合、あるいは MSP を使用して例外復帰時に PSR をアンスタックする際にバス・エラーが発生した場合、プロセッサはロックアップ状態になります。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。
- デバッガによって停止される。
- NMI が発生し、現在のロックアップが HardFault ハンドラ内にある。

注： **NMI ハンドラで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

Cortex-M0+ 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。
- ディープ SLEEP モードでは、超低電力モードに移行します。

SCR の SLEEPDEEP ビットにより、使用する SLEEP モードを選択します（94ページの4.3.6：システム制御ブロックを参照）。ディープ SLEEP モードに移行する際、PWR_CR レジスタの PDSS ビットが、停止モードまたはスタンバイ·モードへの移行を選択します。詳細については、リファレンス·マニュアルの「低電力モード」の章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。このため、ソフトウェアが、そのようなイベントの後にプロセッサを SLEEP モードに戻すことができる必要があります。たとえば、プロセッサを SLEEP モードに戻すアイドル·ループをプログラムに組み込むことができます。

割込みを待機

割込みを待機（WFI）命令は、ただちに SLEEP モードへの移行を発生させます。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、81ページの3.7.12：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·レジスタの値を条件として、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 にセットし、SLEEP モードに移行することなく、引き続き命令を実行します。

詳細については、80ページの3.7.11：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。これは通常、外部イベントのアサートのためか、またはシステム内の別のプロセッサにより SEV 命令が実行されているためです（78ページの3.7.9：SEVを参照）。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してスレッド·モードに戻ると、ただちに SLEEP モードに移行します。このメカニズムは、割込み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.PM ビットを 1 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.PM 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 の詳細については、18 ページの例外マスク・レジスタを参照してください。

WFE からのウェイクアップ

プロセッサは次の場合にウェイクアップします。

- 例外の開始を引き起こすのに十分な優先度の例外を検出した場合。
- 外部イベント信号を検出した場合（36ページの2.5.3：外部イベント入力を参照）。
- マルチプロセッサ・システムで、システム内の別のプロセッサが `SEV` 命令を実行した場合。

また、SCR の SEVONPEND ビットが 1 にセットされている場合、新しい保留中の割込みは、その割込みが無効、あるいは例外の開始を引き起こすのに十分な優先度を持っていなくても、イベントをトリガし、プロセッサをウェイクアップします。SCR の詳細については、94ページの4.3.6：システム制御ブロックを参照してください。

2.5.3 外部イベント入力

プロセッサは外部イベント入力信号を供給します。この信号は、ペリフェラルによって生成される可能性があります。この信号を使用しない場合は LOW にしてください。

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（35 ページのイベント待機を参照）。

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、WFE および SEV 命令を生成できません。これらの命令に対して、CMSIS では次の組込み関数を用意しています。

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

3 Cortex-M0+ 命令セット

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 14 にサポートされる命令を示します。

表 14 で、

- 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、任意のオペランドおよびニーモニック部分を囲みます。
- 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。

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 14. Cortex-M0+ 命令

ニーモニック	オペランド	概要	フラグ	セクション
ADCS	*{Rd}、Rn、Rm*	キャリー付き加算	N、Z、C、V	55ページの3.5.1
ADD{S}	*{Rd}、Rn、<Rm|#imm>*	加算	N、Z、C、V	55ページの3.5.1
ADR	*Rd、label*	レジスタに対する PC 相対アドレス	-	47ページの3.4.1
ANDS	*{Rd}、Rn、Rm*	ビット単位論理積	N、Z	57ページの3.5.2
ASRS	*{Rd}、Rm、<Rs|#imm>*	算術右シフト	N、Z、C	58ページの3.5.3
B{cc}	*label*	分岐 {条件付き}	-	67ページの3.6.1
BICS	*{Rd}、Rn、Rm*	ビット·クリア	N、Z	57ページの3.5.2
BKPT	*#imm*	ブレークポイント	-	70ページの3.7.1
BL	*label*	リンク付き分岐	-	67ページの3.6.1
BLX	*Rm*	リンク付き間接分岐	-	67ページの3.6.1
BX	*Rm*	間接分岐	-	67ページの3.6.1
CMN	*Rn、Rm*	否定比較	N、Z、C、V	60ページの3.5.4
CMP	*Rn、<Rm|#imm>*	比較	N、Z、C、V	60ページの3.5.4
CPSID	*i*	プロセッサ状態の変更、割込みを無効にする	-	71ページの3.7.2
CPSIE	*i*	プロセッサ状態の変更、割込みを有効にする	-	71ページの3.7.2
DMB	-	データ·メモリ·バリア	-	72ページの3.7.3
DSB	-	データ同期バリア	-	73ページの3.7.4
EORS	*{Rd}、Rn、Rm*	排他的論理和	N、Z	57ページの3.5.2
ISB	-	命令同期バリア	-	74ページの3.7.5
LDM	*Rn{!}、reglist*	多重レジスタ·ロード、ポスト·インクリメント	-	51ページの3.4.5
LDR	*Rt、label*	PC 相対アドレスからのレジスタ·ロード	-	48ページの3.4.2
LDR	*Rt、[Rn、<Rm|#imm>]*	レジスタ·ロード（ワード）	-	48ページの3.4.2

表 14. Cortex-M0+ 命令 （続き）

ニーモニック	オペランド	概要	フラグ	セクション
LDRB	*Rt、[Rn、<Rm|#imm>]*	レジスタ·ロード（バイト）	-	48ページの3.4.2
LDRH	*Rt、[Rn、<Rm|#imm>]*	レジスタ·ロード（ハーフワード）	-	48ページの3.4.2
LDRSB	*Rt、[Rn、<Rm|#imm>]*	レジスタ·ロード（符号付きバイト）	-	48ページの3.4.2
LDRSH	*Rt、[Rn、<Rm|#imm>]*	レジスタ·ロード（符号付きハーフワード）	-	48ページの3.4.2
LSLS	*{Rd}、Rn、<Rs|#imm>*	論理左シフト	N、Z、C	58ページの3.5.3
LSRS	*{Rd}、Rn、<Rs|#imm>*	論理右シフト	N、Z、C	58ページの3.5.3
MOV{S}	*Rd、Rm*	転送	N、Z	61ページの3.5.5
MRS	*Rd、spec_reg*	特殊レジスタの内容の汎用レジスタへの転送	-	75ページの3.7.6
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	76ページの3.7.7
MULS	*Rd、Rn、Rm*	乗算、32 ビットの結果	N、Z	62ページの3.5.6
MVNS	*Rd、Rm*	ビット単位否定	N、Z	61ページの3.5.5
NOP	–	何もしない	-	77ページの3.7.8
ORRS	*{Rd}、Rn、Rm*	論理和	N、Z	57ページの3.5.2
POP	*reglist*	レジスタをスタックからポップ	-	53ページの3.4.6
PUSH	*reglist*	レジスタをスタックにプッシュ	-	53ページの3.4.6
REV	*Rd、Rm*	ワードのバイト反転	-	63ページの3.5.7
REV16	*Rd、Rm*	パック·ハーフワードのバイト反転	-	63ページの3.5.7
REVSH	*Rd、Rm*	符号付きハーフワードのバイト反転	-	63ページの3.5.7
RORS	*{Rd}、Rn、Rs*	右ローテート	N、Z、C	58ページの3.5.3
RSBS	*{Rd}、Rn、#0*	反転減算	N、Z、C、V	55ページの3.5.1
SBCS	*{Rd}、Rn、Rm*	キャリー付き減算	N、Z、C、V	55ページの3.5.1
SEV	–	イベント送信	-	78ページの3.7.9
STM	*Rn!、reglist*	複数レジスタのストア、ポスト·インクリメント	-	51ページの3.4.5
STR	*Rt、[Rn、<Rm|#imm>]*	レジスタ·ストア（ワード）	-	48ページの3.4.2
STRB	*Rt、[Rn、<Rm|#imm>]*	レジスタ·ストア（バイト）	-	48ページの3.4.2
STRH	*Rt、[Rn、<Rm|#imm>]*	レジスタ·ストア（ハーフワード）	-	48ページの3.4.2
SUB{S}	*{Rd}、Rn、<Rm|#imm>*	減算	N、Z、C、V	55ページの3.5.1
SVC	*#imm*	スーパバイザ·コール	-	79ページの3.7.10
SXTB	*Rd、Rm*	符号拡張（バイト）	-	64ページの3.5.8

表 14. Cortex-M0+ 命令 （続き）

ニーモニック	オペランド	概要	フラグ	セクション
SXTH	*Rd*、*Rm*	符号拡張（ハーフワード）	-	64ページの3.5.8
TST	*Rn*、*Rm*	論理 AND ベースのテスト	N、Z	65ページの3.5.9
UXTB	*Rd*、*Rm*	ゼロ拡張（バイト）	-	64ページの3.5.8
UXTH	*Rd*、*Rm*	ゼロ拡張（ハーフワード）	-	64ページの3.5.8
WFE	-	イベント待機	-	80ページの3.7.11
WFI	-	割込み待機	-	81ページの3.7.12

3.2 組込み関数

ISO/IEC C コードは、一部の Cortex-M0+ 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関連する命令にアクセスする必要があることがあります。

CMSIS では、ISO/IEC C コードで直接アクセスできない命令を生成するため、以下の組込み関数を提供しています。

表 15. 一部の Cortex-M0+ 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)
ISB	void __ISB(void)
DSB	void __DSB(void)
DMB	void __DMB(void)
NOP	void __NOP(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)
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 16. 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)
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。

- オペランド
- PC または SP を使用した場合の制限事項
- シフト演算
- アドレスのアライメント
- PC 相対式
- 条件付き実行

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の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、これは通常他のオペランドの前に指定されます。

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）を使用できないか、またはそれらのどちらを使用できるかに関しての制約があります。詳細については、命令の説明を参照してください。

注：**PC を BX、BLX、または POP の命令で更新する場合、正常に実行するためには、任意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、このビットがデスティネーション命令セットを示しており、Cortex-M0+ 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。BL または BLX 命令でビット [0] の値を LR に書き込むと、自動的に値 1 が割り当てられます。**

3.3.3 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、ASR、LSR、LSL、および ROR 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。

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、演算結果の右側 *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] の値が左側 *n* ビットにコピーされます。42 ページの図 9を参照してください。

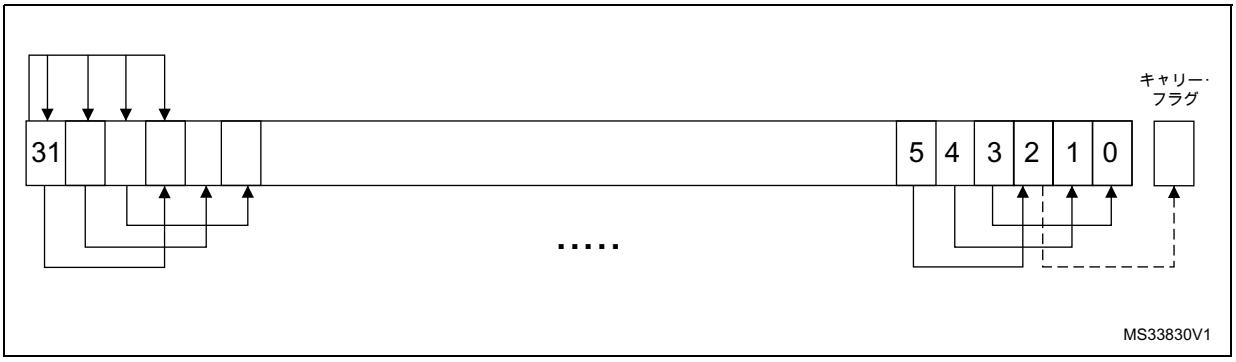
ASR 演算を使用すると、レジスタ Rm の符号付きの値を 2^n で除算し、その結果を負の無限大に丸めることができます。

命令が ASRS の場合、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：***n* が 32 以上の場合、結果のすべてのビットが 0 にクリアされます。**

***n* が 33 以上でキャリー·フラグが更新される場合、0 に更新されます。**

図 9. 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、演算結果の右側 *32-n* ビットが得られます。さらに、左 n ビットを 0 にセットします。42 ページの図 10を参照してください。

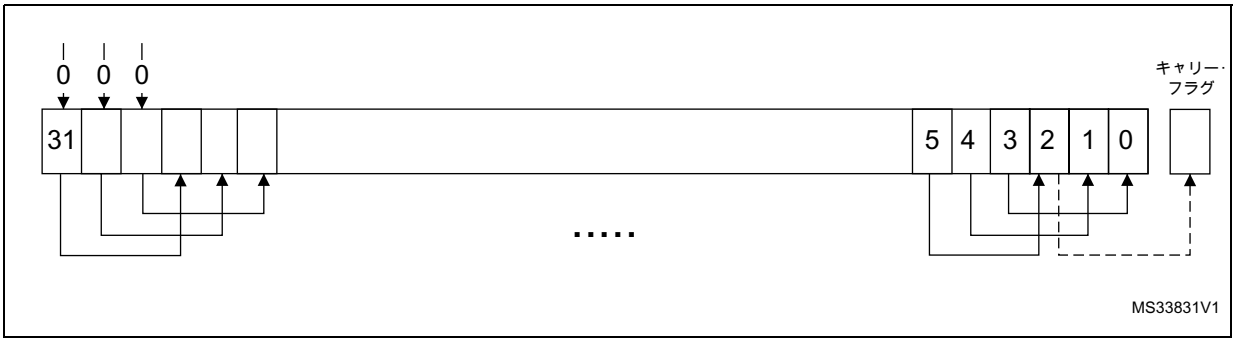
LSR 演算を使用すると、レジスタ *Rm* の値を 2^n で除算できます（値が符号なし整数と見なされる場合）。

命令が LSRS の場合、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： ***n*** **が 32 以上の場合、結果のすべてのビットが 0 にクリアされます。**

n **が 33 以上でキャリー・フラグが更新される場合、0 に更新されます。**

図 10. 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、演算結果の左側 *32-n* ビットが得られます。さらに、右 n ビットを 0 にセットします。43 ページの図 11を参照してください。

LSL 演算を使用すると、レジスタ *Rm* の値を 2^n 倍することができます（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）。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、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： ***n*** **が 32 以上の場合、結果のすべてのビットが 0 にクリアされます。**

n **が 33 以上でキャリー・フラグが更新される場合、0 に更新されます。**

図 11. LSL #3

ROR

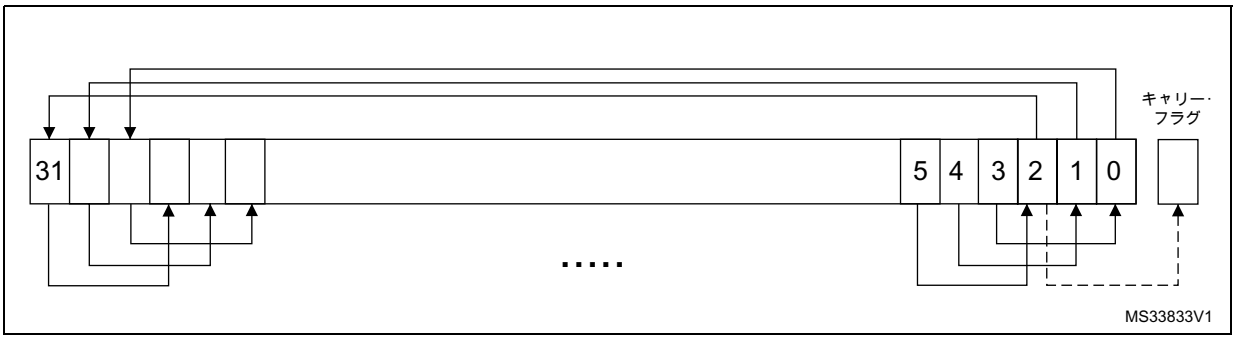
n ビットの右ローテートです。レジスタ *Rm* の左側 *32-n* ビットを右に n 桁転送し、演算結果の右側 *32-n* ビットが得られます。さらに、レジスタの右 n ビットを左 n ビットに転送します。43 ページの図 12を参照してください。

命令が RORS の場合、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： **n が 32 の場合、結果の値は *Rm* の値と同じになります。また、キャリー・フラグが更新される場合は、*Rm* のビット [31] に更新されます。**

32 を超えるシフト長 n と *ROR* を指定した場合、シフト長 *n*-32 と *ROR* を指定した場合と動作は同じになります。

図 12. ROR #3



3.3.4 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、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-M0+ プロセッサでのアンアラインド・アクセスはサポートされていません。アンアラインド・メモリ・アクセス操作を実行しようとすると、HardFault 例外が発生します。

3.3.5 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。

注： **ほとんどの命令では、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,#imm]` 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。

3.3.6 条件付き実行

ほとんどのデータ処理命令では、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します（16 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条件付き分岐命令を実行できます。

- 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。

Cortex-M0+ プロセッサでは、条件付き実行は、条件分岐を使用することによって可能です。

このセクションでは以下の内容について説明します。

- 44 ページの条件フラグ
- 45 ページの条件コードのサフィックス

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、15 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。
- シフトまたはローテート命令の結果として。

オーバーフローは、ビット [31] に格納されている結果の符号が、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。

条件コードのサフィックス

条件付き分岐は、構文の説明では、B{*cond*} と表記しています。条件コードが指定されている分岐命令は、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。それ以外の場合、分岐命令は無視されます。表 17 に、使用できる条件コードを示します。

表 17 では、条件コードのサフィックスと、N、Z、C、および V の各フラグとの関係も示します。

表 17. 条件コードのサフィックス

サフィックス	フラグ	意味
EQ	Z = 1	等しい、最後のフラグ設定結果はゼロでした。
NE	Z = 0	等しくない、最後のフラグ設定結果はゼロ以外でした。
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 または N != V	以下（符号付き）。
AL	すべて	無条件。サフィックスが指定されていない場合は、これがデフォルトです。

3.4 メモリ·アクセス命令

表 18 に、メモリ·アクセス命令を示します。

表 18. メモリ·アクセス命令

ニーモニック	概要	参照先
ADR	PC 相対アドレスの生成	47ページの3.4.1 : ADR
LDM	レジスタの多重ロード	51ページの3.4.5 : 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	48ページの3.4.2 : 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	49ページの3.4.3 : LDR と STR （レジスタ·オフセット）
LDR	PC 相対アドレスからのレジスタ·ロード	50ページの3.4.4 : LDR （PC 相対）
POP	レジスタをスタックからポップ	53ページの3.4.6 : PUSH と POP
PUSH	レジスタをスタックにプッシュ	53ページの3.4.6 : PUSH と POP
STM	レジスタの多重ストア	51ページの3.4.5 : 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	48ページの3.4.2 : 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	49ページの3.4.3 : LDR と STR （レジスタ·オフセット）

3.4.1 ADR

PC 相対アドレスを生成します。

構文

```
ADR Rd, label
```

ここで、

`Rd`	デスティネーション·レジスタです。
`label`	PC 相対式です。43ページの3.3.5：PC 相対式を参照してください。

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、アドレスを生成し、結果をデスティネーション·レジスタに書き込みます。

`ADR` は PC 相対アドレスであるため、位置独立コードの生成を容易にします。

`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。

制限事項

この命令の場合、`Rd` は R0 ~ R7 を指定する必要があります。アドレス指定されたデータ値は、ワード整列され、現在の PC の 1020 バイト以内にある必要があります。

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

```
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
ADR R3, [PC,#996]   ; R3 を PC + 996 の値に設定する。
```

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･オフセットを使ったロードとストア。

構文

```
LDR Rt, [<Rn | SP> {, #imm}]
LDR<B|H> Rt, [Rn {, #imm}]
STR Rt, [<Rn | SP>, {,#imm}]
STR<B|H> Rt, [Rn {,#imm}]
```

ここで、

Rt ロードまたはストアするレジスタです。

Rn メモリ･アドレスのベースとなるレジスタです。

imm *Rn* からのオフセットです。*imm* が省略されている場合、ゼロと想定されます。

動作

LDR、LDRB、LDRH の各命令は、*Rt* で指定されるレジスタに、メモリからワード、バイト、またはハーフワードのデータ値をロードします。ワード未満のサイズは、*Rt* で指定されるレジスタに書き込まれる前に 32 ビットにゼロ拡張されます。

STR、STRB、STRH の各命令は、*Rt* で指定されるシングル･レジスタ内のワード、最下位バイト、下位ハーフワードをメモリに格納します。ロードまたはストアのためのメモリ･アドレスは、*Rn* と SP のいずれかで指定されるレジスタの値とイミディエート値 *imm* の合計です。

制限事項

これらの命令の場合：

- *Rd* と *Rn* では R0 ~ R7 のみを指定する必要があります。
- *imm* の条件：
  - ベース･レジスタに SP を使用する LDR と STR については、0 ~ 1020 の範囲で、4 の倍数の整数。
  - ベース･レジスタに R0 ~ R7 を使用する LDR と STR については、0 ~ 124 の範囲で、4 の倍数の整数。
  - LDRH と STRH については、0 ~ 62 の範囲で、2 の倍数の整数。
  - LDRB と STRB については、0 ~ 31 の範囲
- 計算されたアドレスは、トランザクションのバイト数で割り切れなければなりません。43 ページの3.3.4：アドレスのアライメント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

```
  LDR R4, [R7 ; R7 のアドレスから R4 をロードする。
STR R2,[R0,#const-struc] ; const-struc は、
     ; 0 ~ 1020 の範囲内の定数に評価される式。
```

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

```
LDR Rt, [Rn, Rm]
LDR<B|H> Rt, [Rn, Rm]
LDR<SB|SH> Rt, [Rn, Rm]
STR Rt, [Rn, Rm]
STR<B|H> Rt, [Rn, Rm]
```

ここで、

Rt ロードまたはストアするレジスタです。

Rn メモリ･アドレスのベースとなるレジスタです。

Rm オフセットとして使用される値を含むレジスタです。

動作

LDR、LDRB、LDRH、LDRSB、LDRSH は *Rt* で指定されるレジスタに、ワード、ゼロ拡張バイト、ゼロ拡張ハーフワード、符号拡張バイト、符号拡張ハーフワードの値をメモリからロードします。

STR、STRB、STRH は、*Rt* で指定されるシングル･レジスタ内のワード、最下位バイト、下位ハーフワードをメモリに格納します。

ロードまたはストアのためのメモリ･アドレスは、*Rn* と *Rm* で指定される各レジスタの値の合計です。

制限事項

これらの命令の場合：

- *Rt*、*Rn*、*Rm* では R0 ~ R7 のみを指定する必要があります。
- 計算されたメモリ･アドレスは、ロードまたはストアのバイト数で割り切れなければなりません。43ページの3.3.4：アドレスのアライメント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

```
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
LDRSH  R1, [R2, R3]       ; （R2 + R3）で指定されたメモリ･アドレスから
                          ; ハーフワードをロードし、32 ビットに符号拡張して
                          ; R1 に書き込む。
```

3.4.4 LDR （PC 相対）

メモリからのレジスタ･ロード（リテラル）。

構文

```
LDR Rt, label
```

ここで、

Rt ロードするレジスタです。

label PC 相対式です。43ページの3.3.5：PC 相対式を参照してください。

動作

Rt で指定されるレジスタを、*label* で指定されるメモリ内のワードからロードします。

制限事項

これらの命令の場合、*label* は現在の PC の 1020 バイト以内にあり、ワード整列されている必要があります。

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。
LDR   R3, [PC, #100]    ; R3 に（PC + 100）のメモリ･ワードをロードする。
```

3.4.5 LDM と STM

複数レジスタのロードとストア。

構文

```
LDM Rn{!}, reglist
STM Rn!, reglist
```

ここで、

Rn	メモリ·アドレスのベースとなるレジスタです。
!	ライトバックサフィックス。
reglist	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（51 ページの例を参照）。

LDMIA と LDMFD は LDM の同義語です。LDMIA は、各アクセス後にインクリメントされるベース·レジスタを表します。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。

STMIA と STMEA は STM の同義語です。STMIA は、各アクセス後にインクリメントされるベース·レジスタを表します。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。

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。

アクセスに使用されるメモリ·アドレスは、*Rn* で指定されるレジスタの値から *Rn* + 4 * (n-1) で指定されるレジスタまでの値の範囲（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。ライトバックサフィックスが指定されている場合、*Rn* + 4 * n で指定されるレジスタの値が *Rn* で指定されるレジスタにライトバックされます。

制限事項

これらの命令の場合：

- *reglist* と *Rn* は R0 ~ R7 に制限されます。
- ライトバックサフィックスは必ず使用する必要がありますが、reglist にも *Rn* が格納される LDM が命令の場合を除きます。この場合、ライトバックサフィックスを使用できません。
- *Rn* で指定されるレジスタの値は、ワード整列されていなければなりません。詳細については、43ページの3.3.4：アドレスのアライメントを参照してください。
- STM については、*Rn* が *reglist* に含まれている場合、リストの先頭レジスタでなければなりません。

条件フラグ

これらの命令によるフラグの変更はありません。

例

```
LDM     R0,{R0,R3,R4}       ; LDMIA は LDM の同義語
STMIA   R1!,{R2-R4,R6}
```

誤用例

```
STM     R5!,{R4,R5,R6} ; 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
```

3.4.6 PUSH と POP

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

構文

```
PUSH reglist
POP reglist
```

ここで、

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

動作

`PUSH` はレジスタをスタックにストアし、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。

`POP` は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。

制限事項

これらの命令の場合：

- `reglist` は R0 ~ R7 のみを使用する必要があります。
- 例外は `PUSH` では LR、`POP` では PC になります

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

```
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
```

3.5 汎用データ処理命令

表 19 にデータ処理命令を示します。

表 19. データ処理命令

ニーモニック	概要	参照先
ADCS	キャリー付き加算	55ページの3.5.1 : ADC、ADD、RSB、SBC、および SUB
ADD{S}	加算	55ページの3.5.1 : ADC、ADD、RSB、SBC、および SUB
ANDS	論理積	57ページの3.5.2 : AND、ORR、EOR、および BIC
ASRS	算術右シフト	58ページの3.5.3 : ASR、LSL、LSR、および ROR
BICS	ビット·クリア	57ページの3.5.2 : AND、ORR、EOR、および BIC
CMN	否定比較	60ページの3.5.4 : CMP および CMN
CMP	比較	60ページの3.5.4 : CMP および CMN
EORS	排他的論理和	57ページの3.5.2 : AND、ORR、EOR、および BIC
LSLS	論理左シフト	58ページの3.5.3 : ASR、LSL、LSR、および ROR
LSRS	論理右シフト	58ページの3.5.3 : ASR、LSL、LSR、および ROR
MOV{S}	転送	61ページの3.5.5 : MOV および MVN
MULS	乗算	62ページの3.5.6 : MULS
MVNS	転送して否定	61ページの3.5.5 : MOV および MVN
ORRS	論理和	57ページの3.5.2 : AND、ORR、EOR、および BIC
REV	ワード内のバイト順序反転	63ページの3.5.7 : REV、REV16、および REVSH
REV16	各ハーフワード内のバイト順序反転	63ページの3.5.7 : REV、REV16、および REVSH
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	63ページの3.5.7 : REV、REV16、および REVSH
RORS	右ローテート	58ページの3.5.3 : ASR、LSL、LSR、および ROR
RSBS	反転減算	55ページの3.5.1 : ADC、ADD、RSB、SBC、および SUB
SBCS	キャリー付き減算	55ページの3.5.1 : ADC、ADD、RSB、SBC、および SUB
SUBS	減算	55ページの3.5.1 : ADC、ADD、RSB、SBC、および SUB
SXTB	符号拡張（バイト）	64ページの3.5.8 : SXT および UXT
SXTH	符号拡張（ハーフワード）	64ページの3.5.8 : SXT および UXT
UXTB	ゼロ拡張（バイト）	64ページの3.5.8 : SXT および UXT
UXTH	ゼロ拡張（ハーフワード）	64ページの3.5.8 : SXT および UXT
TST	テスト	65ページの3.5.9 : TST

3.5.1 ADC、ADD、RSB、SBC、および SUB

キャリー付き加算、加算、反転減算、キャリー付き減算、および減算。

構文

```
ADCS   {Rd,} Rn, Rm
ADD{S} {Rd,} Rn, <Rm|#imm>
RSBS   {Rd,} Rn, Rm, #0
SBCS   {Rd,} Rn, Rm
SUB{S} {Rd,} Rn, <Rm|#imm>
```

ここで、

S	ADD または SUB 命令によるフラグ更新を起動させます。
Rd	結果レジスタを指定します。
reglist	最初のソース·レジスタを指定します。
Imm	定数イミディエート値を指定します。

オプションの Rd レジスタ指示子が省略された場合、Rn と同じ値を取ると想定されます。たとえば、ADDS R1,R2 は ADDS R1,R1,R2 と同等に扱われます。

動作

ADCS 命令は、Rn の値を Rm の値に付加し、キャリー·フラグがセットされている場合はさらに 1 が付加されます。また Rd で指定されたレジスタに結果を配置し、N、Z、C、V の各フラグを更新します。

ADD 命令は、Rn の値を Rm の値または imm で指定されるイミディエート値に付加し、結果を Rd で指定されるレジスタに配置します。

ADDS 命令は、ADD と同じ演算を実行し、N、Z、C、V の各フラグも更新します。

RSBS 命令は、Rn の値をゼロから減算し、算術的な負の値を生成し、結果を Rd で指定されるレジスタに配置して、N、Z、C、V の各フラグを更新します。

SBCS 命令は、*Rm* の値を Rn の値から減算し、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。この命令は、結果を Rd で指定されるレジスタに配置し、N、Z、C、V の各フラグを更新します。

SUB 命令は、*Rm* の値または imm で指定されるイミディエート値を減算します。この後、Rd で指定されるレジスタに結果を配置します。

SUBS 命令は、SUB と同じ演算を実行し、N、Z、C、V の各フラグも更新します。

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（56 ページの例を参照）。

47ページの3.4.1：ADRも参照してください。

制限事項

表 20 に、レジスタ指示子の正規の組合せと、各命令で使用できるイミディエート値を一覧にしています。

表 20. ADC、ADD、RSB、SBC および SUB オペランドの制限事項

命令	Rd	Rn	Rm	imm	制限事項
ADCS	R0-R7	R0-R7	R0-R7	-	Rd と Rn で同じレジスタを指定する必要があります。
ADD	R0-R15	R0-R15	R0-PC	-	Rd と Rn で同じレジスタを指定する必要があります。 Rn と Rm には PC を指定できません。
	R0-R7	SP または PC	-	0-1020	イミディエート値は 4 の倍数の整数にする必要があります。
	SP	SP	-	0-508	イミディエート値は 4 の倍数の整数にする必要があります。
ADDS	R0-R7	R0-R7	-	0-7	-
	R0-R7	R0-R7	-	0-255	Rd と Rn で同じレジスタを指定する必要があります。
	R0-R7	R0-R7	R0-R7	-	-
RSBS	R0-R7	R0-R7	-	-	-
SBCS	R0-R7	R0-R7	R0-R7	-	Rd と Rn で同じレジスタを指定する必要があります。
SUB	SP	SP	-	0-508	イミディエート値は 4 の倍数の整数にする必要があります。
SUBS	R0-R7	R0-R7	-	0-7	-
	R0-R7	R0-R7	-	0-255	Rd と Rn で同じレジスタを指定する必要があります。
	R0-R7	R0-R7	R0-R7	-	-

例

例 1 は、R0 と R1 に含まれる 64 ビット整数を、R2 と R3 に含まれる別の 64 ビット整数に加算し、その結果を R0 と R1 に格納する 2 つの命令を示しています。

例 1 64 ビット加算

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

マルチワードの値には、連続するレジスタを使用する必要はありません。例 2 は、R1、R2 および R3 に含まれる 96 ビット整数を、R4、R5 および R6 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 R4、R5 および R6 に格納します。

例 2 96 ビット減算

```
UBS     R4, R4, R1    ; 最下位ワードを減算
SBCS    R5, R5, R2    ; 中位ワードをキャリー付きで減算
SBCS    R6, R6, R3    ; 最上位ワードをキャリー付きで減算
```

例 3 に、単一レジスタの 1 の補数演算を実行する場合に使用される、RSBS 命令を示します。

例 3 算術否定

```
RSBS    R7, R7, #0    ; R7 をゼロから減算
```

3.5.2 AND、ORR、EOR、および BIC

論理積、論理和、排他的論理和、およびビット･クリア。

構文

```
ANDS {Rd,} Rn, Rm
ORRS {Rd,} Rn, Rm
EORS {Rd,} Rn, Rm
BICS {Rd,} Rn, Rm
```

ここで、

Rd	デスティネーション･レジスタです。
Rn	第 1 オペランドを保持するレジスタです。デスティネーション･レジスタと同じです。
Rm	セカンド･レジスタ

動作

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

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

演算結果に基づいて条件コード･フラグが更新されます（48 ページの条件フラグを参照）。

制限事項

これらの命令の場合、*Rd*、*Rn*、*Rm* では R0 ~ R7 のみを指定する必要があります。

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。

C フラグまたは V フラグに影響することはありません。

例

```
ANDS    R2, R2, R1
ORRS    R2, R2, R5
ANDS    R5, R5, R8
EORS    R7, R7, R6
BICS    R0, R0, R1
```

3.5.3 ASR、LSL、LSR、および ROR

算術右シフト、論理左シフト、論理右シフト、および右ローテート。

構文

```
ASRS {Rd,} Rm, Rs
ASRS {Rd,} Rm, #imm
LSLS {Rd,} Rm, Rs
LSLS {Rd,} Rm, #imm
LSRS {Rd,} Rm, Rs
LSRS {Rd,} Rm, #imm
RORS {Rd,} Rm, Rs
```

ここで、

Rd　デスティネーション・レジスタです。*Rd* が省略された場合、*Rm* と同じ値を取ると仮定されます。

Rm　シフトされる値を保持するレジスタです。

Rs　*Rm* の値に適用されるシフト長を保持するレジスタです。

Imm　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

注： **MOVS *Rd*, *Rm* は、LSLS Rd, *Rm*, #0 のシュードニムです。**

動作

ASR、LSL、LSR、ROR はレジスタ *Rm* のビットに対して、イミディエート *imm* で指定される桁数、または *Rs* で指定されるレジスタの最下位バイトの値の算術左シフト、論理左シフト、論理右シフト、右ローテートを実行します。

各種命令から生成される結果の詳細については、41ページの3.3.3：シフト演算を参照してください。

制限事項

これらの命令の場合、*Rd*、*Rm*、*Rs* では R0 ~ R7 のみを指定する必要があります。非イミディエート命令の場合、*Rd* と *Rm* で同じレジスタを指定する必要があります。

条件フラグ

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（41ページの3.3.3：シフト演算を参照）。V フラグは変更されません。

例

```
ASRS    R7, R5, #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
LSRS    R4, R5, #6  ; 6 ビット論理右シフト
RORS    R4, R4, R6  ; R6 の下位バイトの値だけ右ローテートします。
```

3.5.4 CMP および CMN

比較および否定比較。

構文

```
CMN Rn, Rm
CMP Rn, #imm
CMP Rn, Rm
```

ここで、

Rn 第 1 オペランドを保持するレジスタです。

Rm 比較対象のレジスタです。

Imm 比較対象のイミディエート値です。

動作

これらの命令は、レジスタの値を別のレジスタの値またはイミディエート値と比較します。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、*Rm* で指定されるレジスタの値かイミディエート *imm* を *Rn* の値から減算し、フラグを更新します。これは、結果が破棄されることを除けば、SUBS 命令と同じです。

CMN 命令は、*Rm* の値を *Rn* の値に加算し、フラグを更新します。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

次の制限があります。

- CMN 命令 *Rn* と *Rm* では R0 ~ R7 のみを指定する必要があります。
- CMP 命令：
  - *Rn* と *Rm* では R0 ~ R14 を指定できます。
  - イミディエートの範囲は 0 ~ 255 にする必要があります。

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, R2
```

3.5.5 MOV および MVN

転送および転送して否定。

構文

```
MOV{S} Rd, Rm
MOVS Rd, #imm
MVNS Rd, Rm
```

ここで、

S 任意に指定できるサフィックスです。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（44ページの3.3.6：条件付き実行を参照）。

Rd デスティネーション·レジスタです。

Rm レジスタです。

Imm 0 ~ 255 の範囲の値です。

動作

MOV 命令は、*Rm* の値を *Rd* にコピーします。

MOVS 命令は、MOV 命令と同じ演算を実行しますが、N と Z のフラグも更新します

MVSN 命令は、*Rm* 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。

制限事項

これらの命令の場合、*Rd* と *Rm* では R0 ~ R7 のみを指定する必要があります。

Rd が MOV 命令の PC の場合：

- 結果のビット [0] は破棄されます。
- 結果のビット [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。T ビットは変更されません。

注： **MOV を分岐命令として使用することは可能ですが、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強く推奨されています。**

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
- C フラグまたは V フラグに影響することはありません。

例

```
MOVS   R0, #0x000B     ; 0x000B の値を R0 に書き込む。フラグが更新される。
MOVS   R1, #0x0        ; ゼロの値を R1 に書き込む。フラグが更新される。
MOV    R10, R12        ; R12 の値を R10 に書き込む。フラグは更新されない。
MOVS   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, R0          ; R0 の逆数を R2 に書き込み、フラグを更新する
```

3.5.6 MULS

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。

構文

MULS Rd, Rn, Rm

ここで、

Rd デスティネーション·レジスタです。

Rn、*Rm* 乗算される値を保持するレジスタです。

動作

MUL 命令は、Rn と Rm で指定されるレジスタの値を乗算し、結果の下位 32 ビットを Rd に格納します。演算結果に基づいて条件コード·フラグが更新されます（44ページの3.3.6 : 条件付き実行を参照）

この命令の結果は、オペランドの符号の有無には依存しません。

制限事項

この命令の場合：

- Rd、Rn、Rm では R0 ~ R7 のみを指定する必要があります。
- Rd は Rm と同じであることが必要です。

条件フラグ

この命令の場合：

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

例

```
MULS    R0, R2, R0      ; 乗算してフラグを更新、R0 = R0 x R2
```

3.5.7 REV、REV16、および REVSH

バイトの反転。

構文

```
REV Rd, Rn
REV16 Rd, Rn
REVSH Rd, Rn
```

ここで、

Rd デスティネーション·レジスタです。

Rn ソース·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。

RER

`REV` 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、または 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·データに変換します。

`REV16` 2 つのパックされた 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、または 2 つのパックされた 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·データに変換します。

`REVSH` 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、または 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·データに変換します。

制限事項

これらの命令の場合, `Rd` と `Rn` では R0 ~ R7 のみを指定する必要があります。

条件フラグ

これらの命令によるフラグの変更はありません。

例

```
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
```

3.5.8 SXT および UXT

符号付き拡張およびゼロ拡張。

構文

```
SXTB Rd, Rm
SXTH Rd, Rm
UXTB Rd, Rm
UXTH Rd, Rm
```

ここで、

Rd デスティネーション・レジスタです。

Rm 拡張される値を保持するレジスタです。

動作

- これらの命令は、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

これらの命令の場合, *Rd* と *Rm* では R0 ~ R7 のみを指定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への影響はありません。

例

```
SXTH  R4, R6          ; R6 の値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, R1          ; 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
```

3.5.9 TST

ビットのテスト。

構文

```
TST Rn, Rm
```

ここで、

Rn 第 1 オペランドを保持するレジスタです。

Rm テストの対象となるレジスタ。

動作

この命令は、レジスタ内の値を別のレジスタ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 *Rm* 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 にクリアしたレジスタを使用して、TST 命令を実行します。

制限事項

これらの命令の場合, *Rn* と *Rm* では R0 ~ R7 のみを指定する必要があります。

条件フラグ

この命令の場合：

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

例

```
TST   R0, R1 ; R0 の値と R1 の値のビット単位論理積演算を実行
             ; 条件コード·フラグは更新されるが、結果は破棄される
```

3.6 分岐命令と制御命令

表 21 は分岐命令と制御命令を示しています。

表 21. 分岐命令と制御命令

ニーモニック	概要	参照先
B{cc}	分岐 {条件付き}	67ページの3.6.1 : B、BL、BX、および BLX
BL	リンク付き分岐	67ページの3.6.1 : B、BL、BX、および BLX
BLX	リンク付き間接分岐	67ページの3.6.1 : B、BL、BX、および BLX
BX	間接分岐	67ページの3.6.1 : B、BL、BX、および BLX

3.6.1 B、BL、BX、および BLX

分岐命令。

構文

```
B{cond} label
BL label
BX Rm
BLX Rm
```

ここで、

Cond 任意の条件コードです（44ページの3.3.6：条件付き実行を参照）。

label PC 相対式です。43ページの3.3.5：PC 相対式を参照してください。

Rm 分岐先アドレスを示すレジスタです。

動作

これらすべての命令は、*label* で示されたアドレスへの分岐または *Rm* で指定されるレジスタに含ま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、HardFault 例外を発生させます。

BL と BLX 命令はまた、LR のビット [0] を 1 にセットします。これにより、後続の POP {PC} または BX 命令がリターン分岐を成功させるために適した値になります。

表 22 は、各種分岐命令の範囲を示しています

表 22. 分岐範囲

命令	分岐範囲
B *label*	-2 KB ~ +2 KB
B*cond* *label*	-256 バイト ~ +254 バイト
BL *label*	-16 MB ~ +16 MB
BX *Rm*	レジスタ内の任意の値
BLX *Rm*	レジスタ内の任意の値

制限事項

これらの命令の場合：

- BX または BLX 命令では SP または 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。ビット [0] は、EPSR T ビットの更新に使用され、ターゲット･アドレスから破棄されます。

注： **Bcond は、Cortex-M0+ プロセッサの唯一の条件付き命令です。**

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
BL      funC  ; 関数 funC にリンク付きで分岐し（呼出）、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
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）
BEQ     labelD ; 最後のフラグ設定命令で Z フラグをセットした場合は、
              ; 条件付きで labelD に分岐。それ以外の場合は、分岐しない。
```

3.7 その他の命令

表 23 に、Cortex-M0+ のその他の命令を示します。

表 23. 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	70ページの3.7.1：BKPT
CPSID	プロセッサ状態の変更、割込みを無効にする	71ページの3.7.2：CPS
CPSIE	プロセッサ状態の変更、割込みを有効にする	71ページの3.7.2：CPS
DMB	データ·メモリ·バリア	72ページの3.7.3：DMB
DSB	データ同期バリア	73ページの3.7.4：DSB
ISB	命令同期バリア	74ページの3.7.5：ISB
MRS	特殊レジスタからレジスタへの転送	75ページの3.7.6：MRS
MSR	レジスタから特殊レジスタへの転送	76ページの3.7.7：MSR
NOP	何もしない	76ページの3.7.7：MSR
SEV	イベント送信	78ページの3.7.9：SEV
SVC	スーパバイザ·コール	79ページの3.7.10：SVC
WFE	イベント待機	80ページの3.7.11：WFE
WFI	割込み待機	81ページの3.7.12：WFI

3.7.1 BKPT

ブレークポイント。

構文

```
BKPT #imm
```

ここで、

Imm 0 ~ 255 の範囲の整数です。

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。

またプロセッサは、BKPT 命令の実行時にデバッガがアタッチされない場合に、HardFault を生成するか、ロックアップ状態に移行します。詳細については、34ページの2.4.1 : ロックアップを参照してください。

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

```
BKPT #0      ; イミディエート値が 0x0 に設定されているブレークポイント
```

3.7.2 CPS

プロセッサ状態を変更します。

構文

```
CPSID i
CPSIE i
```

動作

CPS は PRIMASK の特殊レジスタの値を変更します。CPSID は、PRIMASK をセットすることにより、割込みを無効にします。CPSIE は、PRIMASK をクリアすることにより、割込みを有効にします。これらのレジスタの詳細については、18 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

現在の実行モードに特権がない場合、この命令は NOP として動作し、PRIMASK の現在の状態を変更しません。

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 ; NMI を除きすべての割込みを無効にする（PRIMASK.PM をセット）
CPSIE i ; 割込みを有効化（PRIMASK.PM をクリア）
```

3.7.3 DMB

データ・メモリ・バリア。

構文

```
DMB
```

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・アクセスより前に観測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には影響しません。

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

```
DMB  ; データ・メモリ・バリア
```

3.7.4 DSB

データ同期バリア。

構文

```
DSB
```

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。

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

```
DSB ; データ同期バリア
```

3.7.5 ISB

命令同期バリア。

構文

```
ISB
```

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。

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

```
ISB  ; 命令同期バリア
```

3.7.6 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 Rd, spec_reg
```

ここで、

Rd 汎用のデスティネーション·レジスタです。

spec_reg 特殊な目的のレジスタAPSR、IPSR、EPSR、IEPSR、IAPSR、EAPSR、PSR、MSP、PSP、PRIMASK、または CONTROL のいずれかです。

動作

MSR は特殊な目的のレジスタの内容を汎用レジスタに格納します。MSR 命令を MSR 命令と結合して、PSR の特定のフラグの変更に適した read-modify-write シーケンスを生成できます。

76ページの3.7.7 : MSRを参照してください。

制限事項

この命令の場合、*Rd* を SP または PC にできません。

現在の実行モードに特権がない場合、APSR 以外のすべてのレジスタの値はゼロとして読み取られます。

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

```
MRS  R0, PRIMASK ; PRIMASK 値を読み出し、R0 に書き込む
```

3.7.7 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 spec_reg, Rn
```

ここで、

Rn 汎用ソース・レジスタです。

spec_reg 特殊な目的のデスティネーション・レジスタ APSR、IPSR、EPSR、IEPSR、IAPSR、EAPSR、PSR、MSP、PSP、PRIMASK、または CONTROL のいずれかです。

動作

MSR は特殊レジスタのいずれかを、Rn で指定されるレジスタの値で更新します。

75ページの3.7.6：MRSを参照してください。

制限事項

この命令の場合、Rn に SP または PC は使用できません。

現在の実行モードに特権がない場合、APSR 以外のレジスタを変更しようとする試みはすべて無視されます。

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
```

3.7.8 NOP

何もしない。

構文

```
NOP
```

動作

NOP は、演算を実行せず、時間消費が保証されません。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。

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

```
NOP  ; 何もしない
```

3.7.9 SEV

イベントを送信します。

構文

```
SEV
```

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ます。また、ローカル･イベント･レジスタをセットします（35ページの2.5：電源管理を参照）。

80ページの3.7.11：WFEも参照してください。

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

```
SEV ; イベント送信
```

3.7.10 SVC

スーパバイザ·コール。

構文

```
SVC #imm
```

ここで、

Imm 0 ~ 255 の範囲の整数です。

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。

制限事項

現在の実行優先順位レベルが SVCall ハンドラの優先順位レベル以上であるときに、SVC 命令を実行すると、フォルトが発生します。

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

```
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）
```

3.7.11 WFE

イベント待機。

構文

```
WFE
```

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、すぐに完了します。

詳細については、35ページの2.5：電源管理を参照してください。

注： **WFE は省電力のみを目的としています。ソフトウェアは書き込み時に、WFE が NOP として動作すると想定します。**

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

```
WFE  ; イベント待機
```

3.7.12 WFI

割込み待機。

構文

```
WFI
```

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します。

- 例外。
- 割込みが保留中になり、PRIMASK.PM がクリアされた場合にプリエンプトする。
- 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）。

注： **WFI は省電力のみを目的としています。ソフトウェアは書き込み時に、WFI が NOP 操作として動作すると想定します。**

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

```
WFI ; 割込み待機
```

4 Cortex-M0+ のコア･ペリフェラル

4.1 Cortex-M0+ のコア･ペリフェラルについて

プライベート･ペリフェラル･バス（PPB）のアドレス･マップを次に示します。

表 24. コア･ペリフェラルのレジスタ領域

アドレス	コア･ペリフェラル	説明
0xE000E008-0xE000E00F	システム制御ブロック	89 ページの表 29
0xE000E010-0xE000E01F	予約済みです。	-
0xE000E010-0xE000E01F	システム･タイマ	97 ページの表 32
0xE000E100-0xE000E4EF	ネスト化されたベクタ割込みコントローラ	83 ページの表 25
0xE000ED00-0xE000ED3F	システム制御ブロック	89 ページの表 29
0xE000ED90-0xE000EDB8	メモリ保護ユニット(1)	101 ページの表 34
0xE000EF00-0xE000EF03	ネスト化されたベクタ割込みコントローラ	83 ページの表 25

1. ソフトウェアは、0xE000ED90 の MPU タイプ･レジスタを読み出して、メモリ保護ユニット（MPU）が存在するかどうかをテストできます。

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 ネスト化されたベクタ割込みコントローラ

このセクションでは、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。

- 32 個の割込み。
- 割込みごとにプログラム可能な、0 ~ 192（64 刻み）の優先度レベル。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 25. NVIC レジスタの概要

アドレス	名前	タイプ	リセット値	説明
0xE000E100	NVIC_ISER	RW	0x00000000	84 ページの割込みセット·イネーブル·レジスタ
0xE000E180	NVIC_ICER	RW	0x00000000	85 ページの割込みクリア·イネーブル·レジスタ
0xE000E200	NVIC_ISPR	RW	0x00000000	85 ページの割込みセット·ペンディング·レジスタ
0xE000E280	NVIC_ICPR	RW	0x00000000	86 ページの割込みクリア·ペンディング·レジスタ
0xE000E400 ~ 0xE000E4EF	NVIC_IPR0-7	RW	0x00000000	86 ページの割込み優先度のレジスタ

4.2.1 CMSIS を使用した Cortex-M0+ 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 26. CMSIS の NVIC アクセス関数

CMSIS 関数	説明
void NVIC_EnableIRQ(IRQn_Type IRQn) (1)	割込みまたは例外を有効にします。
void NVIC_DisableIRQ(IRQn_Type IRQn) (1)	割込みまたは例外を無効にします。
void NVIC_SetPendingIRQ(IRQn_Type IRQn) (1)	割込みまたは例外の保留ステータスを 1 にセットします。
void NVIC_ClearPendingIRQ(IRQn_Type IRQn) (1)	割込みまたは例外の保留ステータスを 0 にクリアします。
uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn) (1)	割込みまたは例外の保留ステータスを読み出します。この関数は、保留ステータスが 1 にセットされている場合は 0 以外の値を返します。
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority) (1)	設定可能な優先度レベルを持つ割込みまたは例外の優先度を 1 にセットします。
uint32_t NVIC_GetPriority(IRQn_Type IRQn) (1)	設定可能な優先度レベルを持つ割込みまたは例外の優先度を読み出します。この関数は、現在の優先度レベルを返します。

1. 入力パラメータ IRQn は IRQ 番号です（詳細については28 ページの表 12を参照 ）。

4.2.2 割込みセット･イネーブル･レジスタ

NVIC_ISER は割込みを有効にして、どの割込みが有効なのかを示します。レジスタの属性については、83 ページの表 25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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETPENA[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
SETPENA[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。

4.2.3 割込みクリア･イネーブル･レジスタ

NVIC_ICER は割込みを無効にして、どの割込みが有効なのかを示します。レジスタの属性については、83 ページの表 25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。

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。

4.2.4 割込みセット･ペンディング･レジスタ

NVIC_ISPR は、割込みを強制的に保留状態にして、どの割込みが保留中なのかを示します。レジスタの属性については、83 ページの表 25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。

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	rs7	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。

注： **NVIC_ISPR ビットへの 1 の書込みは以下に対応します。**

- 保留中の割込みは何の効果もありません。
- 無効な割込みはその割込みの状態を保留中に設定します。

4.2.5 割込みクリア･ペンディング･レジスタ

NVIC_ICPR は、割込みの保留状態を解除して、どの割込みが保留中なのかを示します。レジスタの属性については、83 ページの表 25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。

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：保留状態と割込みを削除します。

読出し：

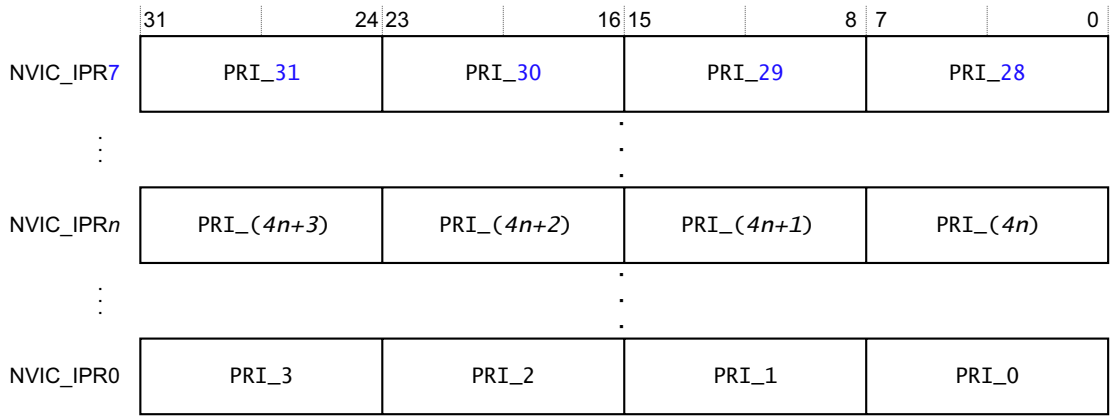
0：割込みは保留中ではありません。

1：割込みは保留中です。

注： **NVIC_ICPR ビットへの 1 の書込みは、対応する割込みのアクティブ状態に影響を及ぼしません。**

4.2.6 割込み優先度のレジスタ

NVIC_IPR0-NVIC_IPR7 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、ワードアクセスのみ可能です。属性については、83 ページの表 25のレジスタ概要を参照してください。各レジスタは、以下に示すように、4 つの優先度フィールドを保持します。



MS33834V1

表 27. NVIC_IPRx 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 ~ 192 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 [7:6] のみ実装し、ビット [5:0] は読み出す値は 0 となり、書込みは無視されます。これは、優先度レジスタに 255 を書き込むと、値 192 がレジスタに保存されることを意味します。
[23:16]	優先度、バイト·オフセット 2	
[15:8]	優先度、バイト·オフセット 1	
[7:0]	優先度、バイト·オフセット 0	

ソフトウェアから見た割込み優先順位を示す割込み優先順位配列へのアクセスの詳細については、83 ページの4.2.1 : CMSIS を使用した Cortex-M0+ NVIC レジスタへのアクセスを参照してください。

割込み M の NVIC_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 NVIC_IPR 番号 N は、N = 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 M MOD 4 です。ここで、
  - バイト·オフセット 0 はレジスタ·ビット [7:0] を参照します。
  - バイト·オフセット 1 はレジスタ·ビット [15:8] を参照します。
  - バイト·オフセット 2 はレジスタ·ビット [23:16] を参照します。
  - バイト·オフセット 3 はレジスタ·ビット [31:24] を参照します。

4.2.7 レベル割込みとパルス割込み

Cortex-M0+ 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（87 ページのハードウェアとソフトウェアによる割込み制御を参照）。レベル検出割込みの場合、プロセッサが ISR から復帰する前に信号がネゲートされない場合、割込みは再び保留中になり、プロセッサはその ISR を再実行する必要があります。これは、割込み処理が不要になるまで、ペリフェラルは割込み信号のアサート状態を保持できることを意味します。

ハードウェアとソフトウェアによる割込み制御

Cortex-M0+ プロセッサは、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、割込み信号がアクティブであるのを検出し、対応する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（85 ページの4.2.4 : 割込みセット·ペンディング·レジスタを参照）。

保留中の割込みは、次のいずれかが発生するまで保留状態に維持されます。

- プロセッサが割込みの ISR に入る。これにより、割込みの状態が保留中からアクティブに変化します。その後、
  - レベル検出割込みの場合は、プロセッサが ISR から復帰する際に NVIC が割込み信号をサンプリングします。信号がアサートされている場合は、割込み状態が保留中に変化し、これによってプロセッサがただちに ISR を再開する可能性があります。アサートされていない場合は、割込み状態が非アクティブに変化します。
  - パルス割込みの場合、NVIC は割込み信号の監視を続行し、パルスが発生すると、割込み状態が保留中およびアクティブに変化します。この場合、プロセッサが ISR から復帰すると、割込み状態は保留中に変化し、これによってプロセッサがただちに ISR を再開する可能性があります。プロセッサが ISR の処理中に割込み信号にパルスが発生しない場合、プロセッサが ISR から復帰すると、割込みの状態は非アクティブに変化します。
- ソフトウェアが、割込みクリア·ペンディング·レジスタの対応するビットに書き込む。

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。

4.2.8 NVIC 使用のヒントとコツ

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。

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、92ページの4.3.4：ベクタ·テーブル·オフセット·レジスタを参照してください。

NVIC のプログラミングのヒント

ソフトウェアは、`CPSIEi` と `CPSIDi` 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 28. CMSIS NVIC 制御関数

CMSIS 割込み制御関数	説明
`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（1）を返します。
`void NVIC_SetPendingIRQ (IRQn_t IRQn)`	IRQn を保留中に設定します。

表 28. CMSIS NVIC 制御関数 （続き）

CMSIS 割込み制御関数	説明
`void NVIC_ClearPendingIRQ (IRQn_t IRQn)`	IRQn の保留ステータスをクリアします。
`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（詳細については28 ページの表 12を参照 ）。これらの関数の詳細については、CMSIS ドキュメントを参照してください。

4.3 システム制御ブロック

システム制御ブロック（SCB）は、システム実装情報を提供し、システムを制御します。これには、システム例外の設定、制御、およびレポートが含まれます。SCB レジスタの内容は以下のとおりです。

表 29. SCB レジスタの概要

アドレス	名前	タイプ	リセット値	説明
0xE000ED00	CPUID	RO	0x410CC601	90ページの4.3.2 : CPUID レジスタ
0xE000ED04	ICSR	RW(1)	0x00000000	90 ページの4.3.3 : 割込み制御およびステート･レジスタ（ICSR）
0xE000ED08	VTOR	RW	0x00000000	92ページの4.3.4 : ベクタ･テーブル･オフセット･レジスタ
0xE000ED0C	AIRCR	RW(1)	0xFA050000	93ページの4.3.5: アプリケーション割込みおよびリセット制御レジスタ
0xE000ED10	SCR	RW	0x00000000	94ページの4.3.6 : システム制御ブロック
0xE000ED14	CCR	RO	0x00000204	95ページの4.3.7 : 設定および制御レジスタ
0xE000ED1C	SHPR2	RW	0x00000000	96 ページのシステム･ハンドラ優先度レジスタ 2
0xE000ED20	SHPR3	RW	0x00000000	96 ページのシステム･ハンドラ優先度レジスタ 3

1. 詳細については、レジスタの説明を参照してください。

4.3.1 Cortex-M0+ SCB レジスタの CMSIS マッピング

ソフトウェアの効率を向上させるために、CMSIS では SCB レジスタの表示を簡素化しています。CMSIS では、配列 `SHP[1]` がレジスタ SHPR2 ~ SHPR3 に対応しています。

4.3.2 CPUID レジスタ

CPUID レジスタには、プロセッサの部品番号、バージョン、および実装情報が格納されます。属性については、89 ページの表 29のレジスタ概要を参照してください。ビット割当てを次に示します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IMPLEMENTER								VARIANT				アーキテクチャ			
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
PART No												REVISION			
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**：rnp*m* リビジョン･ステータスのメジャー･リビジョン番号 *n*：

`0x0`：リビジョン 0

ビット 19:16 **アーキテクチャ**：プロセッサのアーキテクチャを定義する定数：

`0xC`：ARMv6-M アーキテクチャ

ビット 15:4 部品番号：プロセッサの部品番号

`0xC60`：= Cortex-M0+

ビット 3:0 **リビジョン**：rnp*m* リビジョン･ステータスのマイナー･リビジョン番号 *m*：

`0x1`：パッチ1

4.3.3 割込み制御およびステート･レジスタ（ICSR）

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：
  - 保留中の例外の中で優先度が最も高い例外の番号。

ICSR の属性については、89 ページの表 29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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NMIPENDSET	予約済みです。		PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	予約済みです。		ISRPENDING	予約済みです。			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	予約済みです。		VECTACTIVE[8:0]								
r	r	r	r	r			rw	rw	rw	rw	rw	rw	rw	rw	rw

表 30. ICSR ビット割当て

ビット	名前	タイプ	機能
[31]	NMIPENDSET	rw	NMI セット･ペンディング･ビット。 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、通常はこのビットへの 1 の書き込みが検出されるとすぐにプロセッサが NMI 例外ハンドラを開始します。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。つまり、プロセッサがこのハンドラを実行中に NMI が再アサートされた場合、NMI 実行ハンドラで 1 が読めます。
[30:29]	-	-	予約済み。
[28]	PENDSVSET	rw	PendSV セット･ペンディング･ビット。 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。
[27]	PENDSVCLR	w	PendSV クリア･ペンディング･ビット。 書込み： 0 = 影響なし。 1 = PendSV 例外の保留状態を解除します。
[26]	PENDSTSET	rw	SysTick 例外セット･ペンディング･ビット。 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。
[25]	PENDSTCLR	w	SysTick 例外クリア･ペンディング･ビット。 書込み： 0 = 影響なし。 1 = SysTick 例外の保留状態を解除します。 このビットは WO です。レジスタを読み出したときの値は予測不能です。
[24:18]	-	-	予約済み。

表 30. ICSR ビット割当て （続き）

ビット	名前	タイプ	機能
[17:12]	VECTPENDING	r	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。 ゼロ以外 = 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、および優先度レジスタの対応するビットを識別します（17 ページの表 5を参照）。
[11:0]	-	-	予約済み。

ICSR に書き込む際、次の操作をすると、効果は予測不能になります。

- PENDSVSET ビットに 1 を書き込み、PENDSVCLR ビットに 1 を書き込む
- PENDSTSET ビットに 1 を書き込み、PENDSTCLR ビットに 1 を書き込む

4.3.4 ベクタ･テーブル･オフセット･レジスタ

VTOR は、ベクタ･テーブル･ベース･アドレスのメモリ･アドレス `0x00000000` からのオフセットを示しています。属性については、レジスタ概要を参照してください。

ビット割当てを次に示します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TBLOFF[31:16]															
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
TBLOFF[15:7]									予約済みです。						
rw	rw	rw	rw	rw	rw	rw	rw	rw							

ビット 31:7 TBLOFF ベクタ･テーブル･ベース･オフセット･フィールド。
メモリ･マップの下部からのテーブル･ベースのオフセットのビット [31:7] が格納されています。

ビット 6:0 予約済みです。

4.3.5 アプリケーション割込みおよびリセット制御レジスタ

AIRCR は、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 0x05FA を書き込む必要があります。そうしないと、プロセッサによって書込みが無視されます。

ビット割当てを次に示します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VECTKEYSTAT															
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	予約済みです。												SYSRESETREQ	VECTCLRACTIVE	予約済みです。
r													w	w	

ビット 31:16 VECTKEY レジスタ･キー

レジスタ･キー：

不明として読み出されます

書込み時は、VECTKEY に 0x05FA 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 14:3 予約済みです。

ビット 2 SYSRESETREQ システム･リセット要求：

0：影響なし。

1：システム･レベルのリセットを要求します。

このビットは 0 として読み出されます。

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 予約済みです。

4.3.6 システム制御ブロック

SCR は、低電力状態の開始と終了の機能を制御します。属性については、89 ページの表 29のレジスタ概要を参照してください。ビット割当てを次に示します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済みです。															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済みです。											SEVONPEND	Res.	SLEEPDEEP	SLEEPONEXIT	Res.
											rw		rw	rw	

ビット 31:5 予約済みです。

ビット 4 **SEVEONPEND**：保留時イベント送信ビット。

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。

イベントまたは割込みが保留中になると、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。

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：SLEEP

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 を示します。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：ISR からスレッド･モードに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。

4.3.7 設定および制御レジスタ

CCR は読出し専用レジスタであり、Cortex-M0+ プロセッサの動作のいくつかの側面を示しています。CCR の属性については、89 ページの表 29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。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済みです。															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済みです。						STK ALIGN	BFHF NMIGN	予約済みです。			DIV_0_T RP	UN ALIGN_ TRP	Res.	USER SET MPEND	NON BASE THRD ENA
						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

常に 1 として読み出され、例外の開始時の 8 バイトのスタック·アライメントを示します。

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。例外から復帰するときに、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。

ビット 8:4 **予約済み**、クリア状態を保つ必要があります。

ビット 3 **UNALIGN_ TRP**

常に 1 として読み出され、すべてのアンアライン ド·アクセスによって HardFault が発生することを示します。

ビット 2:0 予約済み、クリア状態を保つ必要があります。

4.3.8 システム·ハンドラ優先度レジスタ

SHPR2 ~ SHPR3 の各レジスタは、優先度を設定可能なシステム例外ハンドラの優先度レベルを 0 ~ 192 の範囲で設定します。

SHPR2 ~ SHPR3 は、ワード単位でアクセスできます。属性については、レジスタ概要を参照してください。

CMSIS を使用してシステム例外優先度レベルにアクセスするには、次の CMSIS 関数を使用します。

- `uint32_t NVIC_GetPriority(IRQn_Type IRQn)`
- `void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)`

入力パラメータ `IRQn` は IRQ 番号です（詳細については28 ページの表 12を参照 ）。

システム·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 31. 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
SVCall	PRI_11	96 ページのシステム·ハンドラ優先度レジスタ 2
PendSV	PRI_14	96 ページのシステム·ハンドラ優先度レジスタ 3
SysTick	PRI_15	

各 PRI_N フィールドは 8 ビット幅ですが、プロセッサは各フィールドのビット [7:6] のみ実装し、ビット [5:0] は 0 として読み出され、ここへの書込みは無視されます。

システム･ハンドラ優先度レジスタ 2

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
予約済みです。								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 **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

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
PRI_15								PRI_14							
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
予約済みです。															

ビット 31:24 **PRI_15**：システム･ハンドラ 15、SysTick 例外の優先度(1)

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。

1. これは、SysTick タイマが実装されていない場合、予約済みです。

4.3.9 SCB 使用のヒントとコツ

ソフトウェアが、整列された 32 ビット･ワード･サイズ･トランザクションを使用して、すべての SCB のレジスタにアクセスするようにします。

4.4 SysTick タイマ（STK）

このタイマを有効にすると、再ロード値から 0 にカウント·ダウンし、次のクロック·サイクルで SYST_RVR の値を再ロード（ラップ）した後、後続のクロック·サイクルでデクリメントします。SYST_RVR に 0 の値を書き込むと、次のラップでカウンタが無効になります。カウンタが 0 に遷移すると、COUNTFLAG ステータス·ビットが 1 にセットされます。SYST_CSR を読み出すと、COUNTFLAG ビットが 0 にクリアされます。SYST_CVR に書き込むと、レジスタと COUNTFLAG ステータス·ビットが 0 にクリアされます。この書き込みでは、SysTick 例外ロジックはトリガされません。レジスタを読み出すと、アクセス時にその値が返されます。

注： **プロセッサがデバッグで停止している間は、カウンタはデクリメントしません。**

システム·タイマ·レジスタの内容は以下のとおりです。

表 32. システム·タイマのレジスタの概要

アドレス	名前	タイプ	必要特権	リセット値	説明
0xE000E010	STK_CSR	RW	特権	0x00000000	97ページの4.4.1 : SysTick 制御およびステータス·レジスタ（STK_CSR）
0xE000E014	STK_RVR	RW	特権	不明	98ページの4.4.2 : SysTick 再ロード値レジスタ（STK_RVR）
0xE000E018	STK_CVR	RW	特権	不明	98ページの4.4.3 : SysTick 現在値レジスタ（STK_CVR）
0xE000E01C	STK_CALIB	RO	特権	0xC0000000(1)	99ページの4.4.4 : SysTick 較正値レジスタ（STK_CALIB）

1. SysTick 較正値。

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

SYST_CSR は、SysTick 機能を有効にします。属性については、97 ページの表 32のレジスタ概要を参照してください。ビット割当てを次に示します。

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済みです。															
													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** このレジスタの最後の読出しの後にタイマが 0 になった場合、1 を返します。

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** SysTick タイマ·クロック·ソースを選択します。

0 = 外部リファレンス·クロック。

1 = プロセッサ·クロック。

ビット 1 **TICKINT** SysTick 例外要求を有効にします。
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。

ビット 0 **ENABLE** カウンタを有効にします。
0 = カウンタは無効です。
1 = カウンタは有効です。

4.4.2 SysTick 再ロード値レジスタ（STK_RVR）

STK_RVR は、SYST_CVR にロードする開始値を指定します。属性については、97 ページの表 32のレジスタ概要を参照してください。ビット割当てを次に示します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済みです。								RELOAD							
								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															
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** カウンタが有効であり、それが 0 に到達したときに STK_CVR にロードする値（98 ページの RELOAD 値の計算を参照）。

RELOAD 値の計算

RELOAD 値には、`0x00000001 ~ 0x00FFFFFF` の範囲の任意の値を指定できます。0 の値をプログラムすることはできますが、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。

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。

4.4.3 SysTick 現在値レジスタ（STK_CVR）

STK_CVR には、SysTick カウンタの現在値が格納されます。属性については、97 ページの表 32のレジスタ概要を参照してください。ビット割当てを次に示します。

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

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** 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 SYST_CSR.COUNTFLAG ビットが 0 にクリアされます。

4.4.4 SysTick 較正値レジスタ（STK_CALIB）

STK_CALIB レジスタは、SysTick 較正プロパティを示します。属性については、97 ページの表 32のレジスタ概要を参照してください。ビット割当てを次に示します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NO REF	SKEW	予約済みです。						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 :** 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 :** 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] :**
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。

4.4.5 SysTick 使用のヒントとコツ

割込みコントローラ･クロックは、SysTick カウンタを更新します。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･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がリセット時には未定義の場合の、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 メモリ保護ユニット

このセクションでは、メモリ保護ユニット（MPU）について説明します。

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できます。以下をサポートしています。

- 領域ごとに独立した属性設定
- 領域のオーバーラップ
- システムへのメモリ属性のエクスポート

メモリ属性は、領域へのメモリ･アクセスの動作に影響を及ぼします。Cortex-M0+ 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-M0+ 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 HardFault 例外を生成します。

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（22ページの2.2.1 : メモリの領域、タイプ、および属性を参照）。

100 ページの表 33 に、使用可能な MPU 領域属性を示します。これには、ほとんどのマイクロコントローラの実装に関係のない、共有可能性とキャッシュ動作の属性が含まれます。このような実装をプログラミングするためのガイドラインについては、109 ページのマイクロコントローラの MPU 設定を参照してください。

表 33. 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。

表 33. メモリ属性の概要 （続き）

メモリタイプ	共有可能性	その他の属性	説明
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。

MPU レジスタを使用して、MPU 領域とその属性を定義します。101 ページの表 34に MPU レジスタを示します。

表 34. MPU レジスタの概要

アドレス	名前	タイプ	リセット値	説明
0xE000ED90	MPU_TYPE	RO	0x00000000 または 0x00000800 (1)	102ページの4.5.1：MPU タイプ·レジスタ
0xE000ED94	MPU_CTRL	RW	0x00000000	103ページの4.5.2：MPU 制御レジスタ
0xE000ED98	MPU_RNR	RW	不明	104ページの4.5.3：MPU 領域番号レジスタ
0xE000ED9C	MPU_RBAR	RW	不明	105ページの4.5.4：MPU 領域ベース·アドレス·レジスタ
0xE000EDA0	MPU_RASR	RW	不明	106ページの4.5.5：MPU 領域属性およびサイズ·レジスタ

1. ソフトウェアは、MPU タイプ·レジスタを読み出して、メモリ保護ユニット（MPU）が存在するかどうかをテストできます。MPU タイプ·レジスタを参照してください。

4.5.1 MPU タイプ･レジスタ

MPU_TYPE レジスタは、MPU が存在しているかどうか、存在している場合はサポートする領域の数を示します。属性については、101 ページの表 34のレジスタ概要を参照してください。ビット割当てを次に示します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済みです。								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]								予約済みです。							SEPARATE
r	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]**：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]**：サポートされている MPU データ領域の数を示します。
0x00 = 0 個の領域（デバイスに MPU が含まれていない場合）。
0x08 = 8 個の領域（デバイスに MPU が含まれている場合）。

ビット 7:1 **予約済み。**

ビット 0 SEPARATE：命令とデータのメモリ･マップとして統合マップまたは個別マップのどちらをサポートしているかを示します。
0 = 統合

4.5.2 MPU 制御レジスタ

MPU_CTRL レジスタは、次の目的に使用します。

- MPU を有効にします。
- デフォルトのメモリ·マップのバックグラウンド領域を有効にします。
- HardFault またはノンマスカブル割込み（NMI）ハンドラ内での MPU の使用を有効にします。

MPU_CTRL の属性については、101 ページの表 34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。

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約済みです。													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**：HardFault および NMI 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、HardFault および NMI の各ハンドラの実行中、MPU は無効です。

1 = HardFault および NMI の各ハンドラの実行中に MPU が有効になります。

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 は無効です。

1：MPU は有効です。

ENABLE と PRIVDEFENA の両方が 1 にセットされている場合：

- 特権アクセスの場合、デフォルトのメモリ·マップは21ページの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、MemManage 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（23 ページの表 10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、HardFault または NMI 例外を処理中の場合のみです。それらの 2 つの優先度で動作中の場合、HFNMIENA ビットを 1 にセットすると、MPU が有効になります。

4.5.3 MPU 領域番号レジスタ

MPU_RNR は、MPU_RBAR レジスタと MPU_RASR レジスタが参照するメモリ領域を選択します。属性については、101 ページの表 34のレジスタ概要を参照してください。ビット割当てを次に示します。

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

ビット 31:8 予約済み、クリア状態を保つ必要があります。

ビット 7:0 **REGION** MPU_RBAR レジスタと MPU_RASR レジスタが参照する MPU 領域を示します。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 に書き込むことで領域番号を変更できます（105 ページのMPU 領域ベース·アドレス·レジスタを参照）。この書込みにより、REGION フィールドの値が更新されます。

4.5.4 MPU 領域ベース·アドレス·レジスタ

MPU_RBAR は、MPU_RNR によって選択されている MPU 領域のベース·アドレスを定義し、このレジスタへの書込みにより MPU_RNR の値を更新できます。属性については、101 ページの表 34のレジスタ概要を参照してください。

VALID ビットを 1 にセットして MPU_RBAR に書き込むことによって、現在の領域番号を変更し、MPU_RNR を更新します。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
ADDR[31:N]...															
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
....ADDR[31:N]											VALID	REGION[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	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。
詳細については、ADDR フィールドを参照してください

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。

領域サイズが 32 バイトの場合、ADDR フィールドはビット [31:5] であり、予約済みフィールドはありません。

ADDR フィールド

ADDR フィールドは、MPU_RBAR のビット [31:N] です。MPU_RASR の SIZE フィールドで指定される領域サイズによって、N の値は次のように定義されます。

$N = Log_2$（領域サイズ（バイト単位））

MPU_RASR で領域サイズが 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する必要があります。たとえば、64KB の領域は、64KB の倍数（`0x00010000`、`0x00020000` など）で整列されている必要があります。

4.5.5 MPU 領域属性およびサイズ·レジスタ

MPU_RASR は、MPU_RNR で指定されている MPU 領域の領域サイズとメモリ属性を定義し、その領域とサブ領域を有効にします。属性については、100 ページの表 33のレジスタ概要を参照してください。

ビット割当てを次に示します。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約済みです。			XN	予約済みです。	AP[2:0]			予約済みです。					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]								予約済みです。		SIZE					EN ABLE
rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

ビット 31:29 **予約済みです。**

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]**：アクセス許可フィールド（表 37：AP エンコードを参照）

ビット 23:19 **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 18 **S：共有可能ビット**、（107 ページの表 36を参照）

ビット 17 **C：キャッシュ可能**ビット（107 ページの表 37を参照）

ビット 16 **B：バッファ可能**ビット（107 ページの表 36を参照）

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次を示します。
0 = 対応するサブ領域が有効。
1 = 対応するサブ領域が無効。
詳細については、108 ページのサブ領域を参照してください。

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。
MPU 領域のサイズを指定します。最小許容値は 7 （b00111）です。詳細については、106 ページの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(1)。

1. すべての領域の領域イネーブル·ビットが 0 にリセットされます。これにより、有効にしたい領域のみをプログラムできます。

アクセス許可については、107 ページのMPU アクセス許可属性を参照してください。

SIZE フィールドの値

SIZE フィールドは、MPU_RNR で指定されている MPU メモリ領域のサイズを次のように定義します。

（領域サイズ（バイト単位））= $2^{(SIZE+1)}$

最小許容領域サイズは 256 バイト（SIZE 値 7 に相当）です。表 35 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 35. SIZE フィールドの値の例

SIZE 値	領域サイズ	N 値(1)	注
b00111 (7)	256B	8	最小許容サイズ
b01001 (9)	1KB	10	-
b10011 (19)	1MB	20	-
b11101 (29)	1GB	30	-
b11111 (31)	4GB	32	最大許容サイズ

1. MPU_RBAR 内（105 ページのMPU 領域ベース·アドレス·レジスタを参照）。

4.5.6 MPU アクセス許可属性

このセクションでは、MPU アクセス許可属性について説明します。MPU_RASR の 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 36 に、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 36. C、B、S のエンコード

C	B	S	メモリ·タイプ	共有可能性	その他の属性
0	0	-(1)	Strongly-ordered	共有可能	-
	1	-(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	

1. MPU はこのビットの値を無視します。

表 37 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 37. AP エンコード

AP[2:0]	特権 許可	非特権 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します。
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

4.5.7 MPU の不一致

アクセスが MPU 許可に違反する場合、プロセッサは、HardFault を生成します。

4.5.8 MPU 領域の更新

MPU 領域の属性を更新するには、MPU_RNR、MPU_RBAR、および MPU_RASR の各レジスタを更新します。

MPU 領域の更新

1 つの領域を設定する簡単なコード：

```
; R1 = 領域番号
; R2 = サイズ／有効化
; R3 = 属性
; R4 = アドレス
LDR R0,=MPU_RNR          ; 0xE000ED98、MPU 領域番号レジスタ
STR R1, [R0, #0x0]       ; 領域番号
STR R4, [R0, #0x4]       ; 領域ベース·アドレス
STRH R2, [R0, #0x8]      ; 領域のサイズと有効化
STRH R3, [R0, #0xA]      ; 領域属性
```

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、命令同期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。

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。

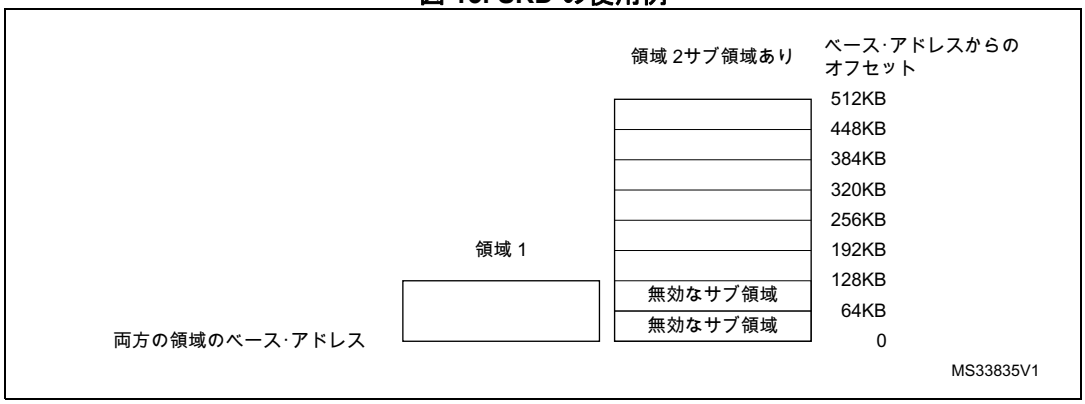
サブ領域

領域は、8 つの等サイズのサブ領域に分割されます。MPU_RASR の SRD フィールドで対応するビットをセットして、サブ領域を無効にします（106 ページのMPU 領域属性およびサイズ·レジスタを参照）。SRD の最下位ビットが最初のサブ領域を制御し、最上位ビットが最後のサブ領域を制御します。あるサブ領域を無効にすると、その無効な範囲とオーバーラップするもう 1 つの領域が代わりに一致をとります。無効なサブ領域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。

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 13. SRD の使用例



4.5.9 MPU 設計のヒントとコツ

予測不能な動作を避けるため、割込みハンドラがアクセスする可能性がある領域の属性を更新する前に、割込みを無効にします。

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 セットアップに影響を及ぼさないようにします。

マイクロコントローラの MPU 設定

通常、マイクロコントローラ･システムにはプロセッサが 1 つしかなく、キャッシュはありません。このようなシステムでは、MPU を次のようにプログラムします。

表 38. マイクロコントローラのメモリ領域属性

メモリ領域	C	B	S	メモリ･タイプと属性
Flashメモリ	1	0	0	ノーマル･メモリ、共有不可、ライトスルー
内部 SRAM	1	0	1	ノーマル･メモリ、共有可能、ライトスルー
外部 SRAM	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	0	1	1	デバイス･メモリ、共有可能

ほとんどのマイクロコントローラ実装では、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。マルチプロセッサ設計や個別の DMA エンジンを備えた設計などの特殊なシステムでは、共有可能性属性が重要になる場合があります。このような場合は、メモリ･デバイスの製造元の推奨事項を参照してください。

4.6 I/O ポート

Cortex-M0+ は、ペリフェラルへの高速で、低レイテンシなアクセスのために専用の I/O ポートを実装しています。I/O ポートはメモリ・マップされており、46 ページのメモリ・アクセス命令で指定されたすべてのロードおよびストア命令をサポートします。I/O ポートはコード実行をサポートしていません。

汎用 I/O は、I/O ポートを介してアクセスされます。

I/O ポートは MPU で保護できます。

5 改版履歴

表 39. 文書改版履歴

日付	版	変更内容
2014 年 4 月 15 日	1	初版発行
2017 年 6 月 16 日	2	セクション 2.3.4 : ベクタ·テーブルを更新。
2018 年 1 月 19 日	3	セクション 3.5.1 : ADC、ADD、RSB、SBC、および SUBを更新。
2018 年 10 月 25 日	4	STM32G0 シリーズを追加。
2019 年 10 月 10 日	5	STM32WL および STM32WB シリーズを追加。

表 40. 日本語版文書改版履歴

日付	版	変更内容
2022 年 3 月 15 日	1	日本語版初版発行

重要なお知らせ（よくお読み下さい）

STMicroelectronics NV およびその子会社（以下、ST）は、ST製品及び本書の内容をいつでも予告なく変更、修正、改善、改定及び改良する権利を留保します。購入される方は、発注前にST製品に関する最新の関連情報を必ず入手してください。ST 製品は、注文請書発行時点で有効なST の販売条件に従って販売されます。

ST製品の選択並びに使用については購入される方が全ての責任を負うものとします。購入される方の製品上の操作や設計に関してST は一切の責任を負いません。

明示又は黙示を問わず、STは本書においていかなる知的財産権の実施権も許諾致しません。

本書で説明されている情報とは異なる条件でST 製品が再販された場合、その製品についてSTが与えたいかなる保証も無効となります。

ST およびST ロゴはSTMicroelectronics の商標です。STの登録商標についてはSTウェブサイトをご覧ください。
www.st.com/trademarks
その他の製品またはサービスの名称は、それぞれの所有者に帰属します。

本書の情報は本書の以前のバージョンで提供された全ての情報に優先し、これに代わるものです。

この資料は、STMicroelectronics NV 並びにその子会社(以下ST)が英文で記述した資料（以下、「正規英語版資料」）を、皆様のご理解の一助として頂くためにSTマイクロエレクトロニクス㈱が英文から和文へ翻訳して作成したものです。この資料は現行の正規英語版資料の近時の更新に対応していない場合があります。この資料は、あくまでも正規英語版資料をご理解頂くための補助的参考資料のみにご利用下さい。この資料で説明される製品のご検討及びご採用にあたりましては、必ず最新の正規英語版資料を事前にご確認下さい。ST及びSTマイクロエレクトロニクス㈱は、現行の正規英語版資料の更新により製品に関する最新の情報を提供しているにも関わらず、当該英語版資料に対応した更新がなされていないこの資料の情報に基づいて発生した問題や障害などにつきましては如何なる責任も負いません。