第 2 巻：RISC-V 特権アーキテクチャ V1.10

はじめに

この文章は RISC-V の命令マニュアルを @shibatchii が RISC-V アーキテクチャ勉強のためメモしながら訳しているものです。
原文は https://riscv.org/specifications/ にある riscv-privileged-v1.10.pdf です。

https://github.com/shibatchii/RISC-V
に置いてあります。

英語は得意でないので誤訳等あるかもしれません。ご指摘歓迎です。
Twitter: @shibatchii

Google 翻訳、Bing 翻訳、Webilo 翻訳、Exclite 翻訳 を併用しながら翻訳し、勉強しています。

まずは意味が分からないところもあるかもしれませんが、ざっくり訳して 2 周位回ればまともになるかなと。
体裁とかは後で整えようと思います。

文章は以下の様に色分けしてます。

黒文字：翻訳した文書。
赤文字：@shibatchii コメント。わからないところとか、こう解釈したとか。
青文字：RISC-V にあまり関係なし。訳した日付とか、集中力が切れた時に書くヨタ話とか。

2018/09/28 @shibatchii

RISC-V 命令セットマニュアル第 2 巻：特権付きアーキテクチャ
特権アーキテクチャバージョン 1.10
ドキュメントバージョン 1.10

警告！ このドラフト仕様は、RISC-V 財団によって標準として承認される前に変更される可能性があります。
編集者はこの仕様の今後の変更が順方向互換になるよう意図していますが、
この仕様書の実装が将来の標準に準拠しない可能性があります。

編集者：アンドリュー ウォーターマン 1、クレステ アサノビッチ 1,2
1SiFive Inc.、
2 カリフォルニア大学バークレー校 EECS 学科 CS 課
andrew@sifive.com、krste@berkeley.edu
2017 年 5 月 7 日

第２巻：RISC-V 特権アーキテクチャ V1.10

アルファベット順の仕様全バージョン貢献者（訂正があれば編集者に連絡してください）：
クレステ・アサノビック、リマス・アヴィジエニス、ジェイコブ・バッハマイヤー、アレン・バウム、パオロ・ボンツィーニ、ルスラン・ブーキン、クリストファー・セリオ、デビッド・キスナー、アンソニー・コールター、パーマー・ダッベルト、モンテ・ダリムプリ、デニス・ファーガソン、マイク・フライシンガー、ジョン・ハウザー、デヴィッド・ホーナー、オロフ・ヨハンソン、リー・ユンサプ、アンドリュー・ルトミルスキー、ジョナサン・ノイシェファー、リシュユール・ニヒル、ステファン・オレア、アルバート・ウー、ジョン・オースターハウト、デビッド・パターソン、コリン・シュミット、ウェズリー・ターブストラ、マット・トーマス、トミー・ソーン、レイ・ヴァンデウォーカー、メガン・ワックス、アンドリュー・ウォーターマン、レイノード・ザンディク

次のように引用してください：「RISC-V 命令セットマニュアル、第２巻：特権アーキテクチャ、バージョン 1.10」、編集者アンドリュー・ウォーターマンとクレステ・アサノビック、RISC-V 財団、2017 年５月。

配布条件はこれですね。https://creativecommons.org/licenses/by/4.0/deed.ja
制限が少ない大変良いですね。

--2018/10/12

第 2 巻：RISC-V 特権アーキテクチャ V1.10

序文

これは、RISC-V 特権アーキテクチャ提案のバージョン 1.10 です。
バージョン 1.9.1 からの変更点：

- このドキュメントの以前のバージョンは、元の作者が作成したクリエイティブコモンズ帰属 4.0 国際ライセンスの下でリリースされました。このドキュメントの今後のバージョンは、同じライセンスでリリースされる予定です。
- シャドー CSR アドレスに関する明示的な規則は、CSR スペースを再利用するために削除されました。
  シャドー CSRs は、必要に応じて追加することができます。
- mvendorid レジスタには、財団が提供するコードとは対照的に、コアプロバイダの JEDEC コードが含まれるようになりました。
  これにより、財団からの冗長性とオフロード作業が回避されます。 ←JEDEC コードってなんじゃろ？
- 割込み可能なスタック規律は単純化されています。
- 監督者モードとユーザモードで使用されるベース ISA を変更するためのオプションのメカニズムが mstatus CSR に追加されました。misa の以前に Base と呼ばれていたフィールドは、整合性のために MXL に改名されました。
- mstatus の追加の拡張状態のステータスフィールドを要約するために XS を使用することを明確にしました。
- mtvec および stvec CSRs には、オプションのベクタ割り込みのサポートが追加されています。
- mip CSR の SEIP ビットと UEIP ビットは、外部割り込みのソフトウェア注入をサポートするために再定義されています。
- mbadaddr レジスタはより一般的な mtval レジスタに含まれています。これにより、命令エミュレーションの速度を上げるために不正な命令フォルトで不良命令ビットを取り込むことができます。
- 仮想メモリ構成を sptbr（now satp）に移行する過程の一環として、マシンモードのベースと境界の変換と保護のスキームが仕様から削除されました。
  ベースとバインドされたスキームのモチベーションのいくつかは PMP レジスタでカバーされていますが、mstatus で利用可能なまま残っていれば、後で役立ちます。
- M モードのみ、または M モードと U モードの両方で U モードトラップをサポートしていないシステムでは、medeleg レジスタと mideleg レジスタは存在しませんでしたが、以前はゼロを返していました。
- 仮想メモリページ違反は、物理メモリアクセス例外とは別の mcause 値を持つようになりました。
  ページ違反例外は、PMA および PMP チェックによって生成された例外を委任することなく、
  S モードに委任できるようになりました。
- オプションの物理メモリ保護（PMP）方式が提案されています。
- 監督者の仮想メモリ構成は、mstatus レジスタから sptbr レジスタに移動されました。

したがって、sptbr レジスタは satp（監督者アドレスの転送と保護）に名前が変更され、広がった役割が反映されます。

↑レジスタ名は整理しとかないとどれがどれかわからなくなりそう。

- 改善された SFENCE.VMA 命令のために、SFENCE.VM 命令が削除されました。
- mstatus ビット MXR は sstatus を介して S モードにさらされています。
- sstatus の PUM ビットの極性は、MXR を含むコードシーケンスを短縮するために反転されています。
  ビットの名前が SUM に変更されました。
- ページテーブルエントリのハードウェア管理 Accessed ビットと Dirty ビットはオプションです。簡単な実装では、ソフトウェアをトラップして設定することができます。
- カウンタイネーブル方式が変更されたため、S モードはカウンタの U モードへの可用性を制御できます。
- S モードでの再帰的仮想化サポートに焦点を当てているため、H モードは削除されています。
- エンコーディング空間は予約されており、後で再利用することができます。
- S モード仮想メモリ管理操作をトラップして仮想化パフォーマンスを向上させるメカニズムが追加されました。
- 監督者バイナリインターフェイス（SBI）の章が削除され、別の仕様として維持することができます。

第 2 巻：RISC-V 特権アーキテクチャ V1.10

バージョン 1.9.1 の序文

これは、RISC-V 特権アーキテクチャ提案のバージョン 1.9.1 です。
バージョン 1.9 からの変更点：

- 解説セクションへの多数の追加と改良。
- 設定文字列の提案を変更して、デバイスツリーストリングやフラット化されたデバイスツリーなどのさまざまなフォーマットをサポートする検索プロセスを使用します。
- misa は、ベースおよびサポートされている ISA 拡張の変更をサポートするように、オプションで書き込み可能になりました。
  misa の CSR アドレスが変更されました。
- デバッグモードとデバッグ CSRs の説明が追加されました。
- ハードウェアパフォーマンス監視スキームを追加しました。
  既存のハードウェアカウンタの処理を簡素化し、カウンタの特権バージョンと対応するデルタレジスタを削除しました。
- ユーザーレベルの割り込みがある場合の SPIE の説明が修正されました。

↑misa ってなんだっけ。後で確認

-- 1 ページ空き、次ページへ

内容

↑冪等性 (とうべきせい) 「大雑把に言って、ある操作を１回行っても複数回行っても結果が同じであることをいう概念である。」って。なんか難しい言葉やね。初めて聞いた。プログラムでいうといつも同じ答えが出るってこと。あたりまえじゃんと思うけど、割り込み処理が入って意図せずメモリ内容買い換えられちゃったりすると崩れちゃう。

-- 2018/10/13

第 2 巻：RISC-V 特権アーキテクチャ V1.10

第 1 章

前書き

これは、RISC-V の特権付きアーキテクチャ記述文書の草案です。
フィードバックを歓迎します。
変更は最終リリース前に発生します。←行われます。位の意味か

このドキュメントでは、RISC-V 特権アーキテクチャについて説明します。特権命令やオペレーティングシステムの実行、外部デバイスの接続に必要な追加機能など、ユーザーレベルの ISA を超えた RISC-V システムのあらゆる側面をカバーしています。

私たちの意思決定に関する解説は、この段落のように書式化されており、読者が仕様書そのものに興味があればスキップすることができます。
↑riscv-spec-v2.2 にはこれ書いてなかったような。字下げしてあるところは経緯なので、飛ばしても OK だよって。あ、いや、すまん、書いてあった。

ここでは、このドキュメントで説明している特権レベルの設計全体を、ユーザーレベルの ISA を変更することなく、おそらく ABI を変更することなく、まったく異なる特権レベルの設計に置き換えることができます。
特に、この特権仕様は、既存の一般的なオペレーティングシステムを実行するように設計されており、従来のレベルベースの保護モデルを具体化しています。
代替の特権的な仕様は、他のより柔軟な保護ドメインモデルを具体化することができます。

1.1 RISC-V ハードウェアプラットフォームの用語

RISC-V ハードウェアプラットフォームには、他の非 RISC-V 互換コア、固定機能アクセラレータ、さまざまな物理メモリ構造、I / O デバイス、およびコンポーネントが通信できるようにする相互接続構造とともに、1 つまたは複数の RISC-V 互換プロセッシングコアを含めることができます。

コンポーネントが独立した命令フェッチユニットを含む場合、コンポーネントはコアと呼ばれます。
RISC-V 互換のコアは、マルチスレッド化により、複数の RISC-V 互換ハードウェアスレッドまたはハートをサポートしている可能性があります。←可能性があります。より サポートする場合があります。ぐらい？

RISC-V コアには、追加の特殊な命令セット拡張機能や追加されたコプロセッサが追加されている場合があります。
コプロセッサという用語は、RISC-V コアに接続されたユニットを指し、ほとんどが RISC-V 命令ストリームによってシーケンスされますが、追加のアーキテクチャ状態および命令セット拡張、および場合によってはプライマリ RISC-V 命令ストリームに対するいくつかの限定された自律性を含みます。

我々は、非プログラマブルな固定機能ユニットまたは自律的に動作することができますが、特定のタスクに特化されたコアのいずれかを指すように用語アクセラレータを使用しています。
RISC-V システムでは、多くのプログラム可能なアクセラレータが、専用の命令セット拡張および/またはカスタマイズされたコプロセッサを備えた RISC-V ベースのコアになると期待しています。
RISC-V アクセラレータの重要なクラスは I / O アクセラレータであり、メインアプリケーションコアから I / O 処理タスクの荷をおろします。←I/O 処理タスクを切り離す 位の意味か

RISC-V ハードウェアプラットフォームのシステムレベルの構成には、シングルコアマイクロコントローラから数千ノードの共有メモリの manycore サーバーノードのクラスタまでが含まれます。
小さなシステム・オン・チップでも、開発努力(作業)をモジュール化したり、サブシステム間の安全な分離を提供するために、マルチコンピュータおよび/またはマルチプロセッサの階層として構成することができます。←構造化される可能性があります。

このドキュメントでは、ユニプロセッサまたは共有メモリマルチプロセッサ内で動作する各ハードウェア（ハードウェアスレッド）が認識できる特権アーキテクチャに焦点を当てています。

-- 2018/10/14

1.2 RISC-V 特権ソフトウェアスタックの用語

このセクションでは、RISC-V のさまざまな可能な特権ソフトウェアスタックのコンポーネントを説明するために使用する用語について説明します。

図 1.1 は、RISC-V アーキテクチャでサポート可能なソフトウェアスタックのいくつかを示しています。
左側には、アプリケーション実行環境（AEE）上で実行される単一のアプリケーションのみをサポートするシンプルなシステムが示されています。
アプリケーションは、特定のアプリケーションバイナリインタフェース（ABI）で動作するようにコード化されています。
ABI には、サポートされているユーザーレベルの ISA と、AEE と対話する一連の ABI 呼び出しセットが含まれています。
AEE は、AEE の詳細を柔軟に適用できるように、アプリケーションから AEE の詳細を隠しています。←遮蔽している
同じ ABI を複数の異なるホスト OSs にネイティブに実装することも、異なるネイティブ ISA を持つマシン上で実行されるユーザー・モード・エミュレーション環境でサポートすることもできます。

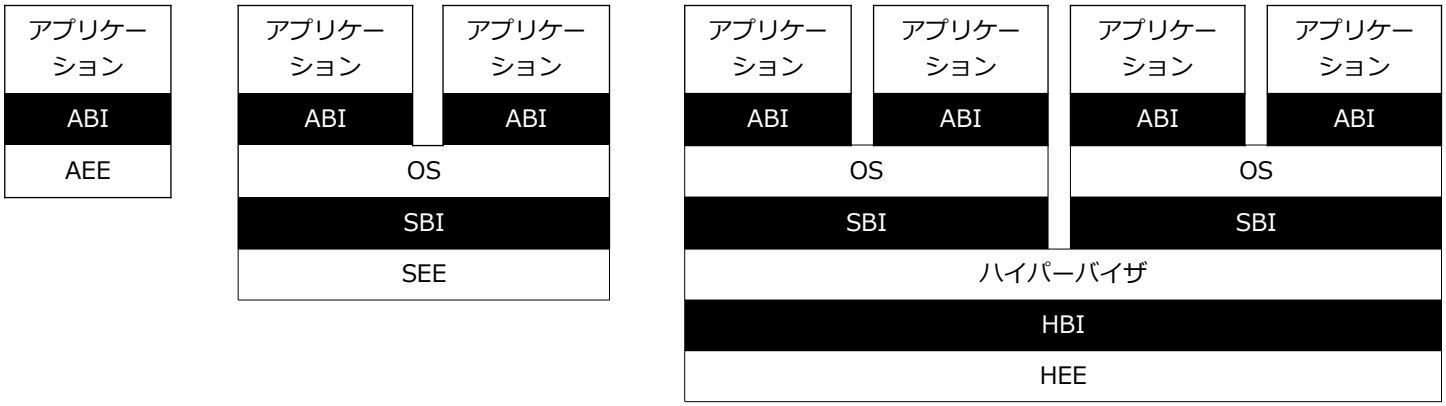


図 1.1：さまざまな形式の特権実行をサポートする異なる実装スタック

私たちのグラフィカルな表記法は、白いテキストの黒いボックスを使用して抽象的なインターフェイスを表し、インターフェイスを実装するコンポーネントの具体的なインスタンスからそれらを分けます。

中央の構成は、複数のアプリケーションのマルチプログラム実行をサポートできる従来のオペレーティングシステム（OS）を示しています。
各アプリケーションは、ABI を介して AEE を提供する OS と通信します。
アプリケーションが ABI 経由で AEE とインタフェースするのと同様に、RISC-V オペレーティングシステムは、監督者バイナリインタフェース（SBI）を介して監督者実行環境（SEE）とインタフェースします。
SBI は、ユーザレベルおよび監督者レベルの ISA と、一連の SBI ファンクションコールを備えています。←組み合わせて構成されています。

すべての SEE 実装で単一の SBI を使用することで、単一の OS バイナリイメージを任意の SEE で実行することができます。
SEE は、ローエンドのハードウェアプラットフォームで単純なブートローダと BIOS スタイルの IO システムにすることができ、またはハイエンドサーバにハイパーバイザが提供する仮想マシンを使用して、またはアーキテクチャシミュレーション環境でホストオペレーティングシステム上の薄い転送層を使用することができます。←シントランスレーションレイヤー

ほとんどの監督者レベルの ISA 定義では、SBI と実行環境および/またはハードウェアプラットフォームが分離されておらず、新しいハードウェアプラットフォームの仮想化および構築が複雑になります。

右端の構成は、複数のマルチプログラミングされた OSs が単一のハイパーバイザによってサポートされる仮想マシンモニタ構成を示しています。
各 OS は SBI を介して SEE を提供するハイパーバイザと通信します。
ハイパーバイザは、ハイパーバイザバイナリインターフェイス（HBI）を使用してハイパーバイザ実行環境（HEE）と通信し、ハードウェアプラットフォームの詳細からハイパーバイザを分離します。

ABI、SBI、および HBI はまだ進行中ですが、SBI が S モード OS によって再帰的に提供されるタイプ 2 ハイパーバイザーのサポートに優先順位を付けるようになっています。

RISC-V ISA のハードウェア実装では、通常、さまざまな実行環境（AEE、SEE、または HEE）をサポートするために、特権 ISA 以外の追加機能が必要になります。

1.3 特権レベル

いつでも、RISC-V ハードウェアスレッド（ハート）は、1 つまたは複数の CSR（コントロールおよびステータスレジスタ）のモードとしてエンコードされた権限レベルで実行されています。
3 つの RISC-V 特権レベルが現在表 1.1 に示すように定義されています。

レベル	エンコーディング	名前	略語
0	00	ユーザー/アプリケーション	U
1	01	スーパーバイザ(監督者)	S
2	10	予約	
3	11	マシン	M

表 1.1：RISC-V 特権レベル。

特権レベルは、ソフトウェアスタックのさまざまなコンポーネント間で保護を提供するために使用され、現在の特権モードで許可されていない操作を実行しようとすると、例外が発生します。
これらの例外は、通常、トラップを基になる実行環境に持ち込みます。←トラップを発生させます。

マシンレベルは最高の特権を持ち、RISC-V ハードウェアプラットフォームの唯一の必須の特権レベルです。
マシンモード（M モード）で実行されるコードは、マシン実装に対する低レベルのアクセス権を持つため、通常は本質的に信頼されています。
M モードは、RISC-V 上の安全な実行環境を管理するために使用できます。
ユーザモード（U モード）および監督者モード（S モード）は、それぞれ従来のアプリケーションおよびオペレーティングシステムの使用を意図しています。←対象としています。

タイプ 1 ハイパーバイザをサポートするように設計された以前のハイパーバイザモード（H モード）は削除され、そして、
第 5 章で説明したタイプ 1 とタイプ 2 の両方のハイパーバイザーに適した拡張 S モードを使用してハイパーバイザーのサポートに焦点を当てているため、予約されているエンコードスペースが必要です。

H のエンコーディング空間は、将来の使用のために予約されており、様々な状態記録者のビット位置の後方互換性のない変更を回避するために予約されています。
ビットポジションは、異なる Type-1 ハイパーバイザのサポート、または可能であれば付加的な安全な実行モードのために将来再利用される可能性があります。

各特権レベルには、オプションの拡張子とバリアント(変形、異形)を持つ特権 ISA 拡張のコアセットがあります。
たとえば、マシンモードでは、アドレス変換とメモリ保護のためのいくつかのオプションの標準バリアントがサポートされています。
また、第 5 章で説明したように、監督者モードを拡張してタイプ 2 ハイパーバイザの実行をサポートすることもできます。

表 1.2 に示すように、インプリメンテーションでは、実装コストを削減するために、1 つから 3 つの特権モードを削減することができます。

この説明では、コードが書き込まれる特権レベルを、それが実行される特権モードから分離しようとしますが、2 つは頻繁に結びついています。
たとえば、監督者レベルのオペレーティングシステムは、3 つの特権モードを持つシステム上で監督者モードで実行できますが、2 つ以上の特権モードを持つシステム上の従来の仮想マシンモニタでは、ユーザモードで実行することもできます。
いずれの場合も、同じ監督者レベルのオペレーティングシステムバイナリコードを使用し、監督者レベルの SBI にコード化し、監督者レベルの特権命令および CSR s を使用できるようにすることを期待しています。
ゲスト OS をユーザモードで実行する場合、すべての監督者レベルのアクションは、より高い特権レベルで実行されている SEE によってトラップとエミュレートされます。

レベル数	サポートされるモード	使用目的
1	M	シンプルな組み込みシステム
2	M, U	安全な組み込みシステム
3	M, S, U	Unix ライクなオペレーティングシステムを実行するシステム

表 1.2：サポートされている特権モードの組み合わせ

すべてのハードウェア実装は、M モードを提供する必要があります。これは、マシン全体に自由にアクセスできる唯一のモードであるためです。
最も単純な RISC-V 実装では、M モードのみが提供されますが、不正または悪意のあるアプリケーションコードに対する保護は提供されません。

オプションの PMP 設備のロック機能は、M モードのみが実装されていても、ある程度の保護を提供することができます。

多くの RISC-V 実装は、アプリケーションコードからシステムの残りの部分を保護するために、少なくともユーザーモード（U モード）もサポートします。
監督者・モード（S モード）を追加すると、監督者・レベルのオペレーティング・システムと SEE の間の分離が可能になります。

ハートは通常、あるトラップ（例えば監督者コールまたはタイマー割込み）が通常より特権モードで動作するトラップハンドラへの切り替えを強制するまで、U モードでアプリケーションコードを実行します。
ハートはトラップハンドラを実行し、最終的に U モードで元のトラップされた命令の実行後に実行を再開します。
特権レベルを上げるトラップは垂直トラップと呼ばれ、同じ特権レベルに留まるトラップは水平トラップと呼ばれます。
RISC-V 特権アーキテクチャは、異なる特権層へのトラップの柔軟なルーティングを提供します。

水平トラップは、特権レベルの低いモードで水平トラップハンドラに制御を戻す垂直トラップとして実装できます。

-- ABI,AEE,SBI,SEE とかの略語はどこかで表にしよう。

1.4 デバッグモード

実装はまた、オフチップデバッグおよび/または製造テストをサポートするためのデバッグモードを含むことができます。
デバッグモード（D モード）は、M モードよりもさらに多くのアクセス権を持つ追加の特権モードと考えることができます。
個別のデバッグ仕様の提案では、デバッグモードでの RISC-V ハートの動作について説明します。
デバッグモードでは、D モードでしかアクセスできないいくつかの CSR アドレスが予約されます。また、プラットフォーム上の物理メモリ領域の一部を予約することもできます。

-- 2018/10/21

-- 1 ページ空き、次ページへ

第 2 章

制御およびステータスレジスタ（CSR）

システムの主要なオペコードは、RISC-V ISA 内のすべての特権命令をエンコードするために使用されます。
これらは 2 つの主要なクラスに分けることができます：
これらは、アトミックにリード・モディファイ・ライト・コントロールおよびステータス・レジスタ（CSRs）、およびその他のすべての特権命令です。
このマニュアルの第 1 巻で説明されているユーザーレベルの状態に加えて、実装には追加の CSRs が含まれ、ユーザレベルのマニュアルに記載された CSR 命令を使用して権限レベルのいくつかのサブセットによってアクセス可能です。
この章では、CSR アドレス空間をマッピングします。
以下の章では、特権レベルによる各 CSRs の機能、および一般に特定の特権レベルと密接に関連するその他の特権命令について説明します。
CSR と命令は 1 つの特権レベルに関連付けられていますが、それらはまた、すべての上位の特権レベルでアクセス可能であることに注意してください。

2.1 CSR アドレスマッピング規則

標準の RISC-V ISA は、最大 4,096 の CSR に対して 12 ビットの符号化空間（csr [11：0]）を設定します。
表 2.1 に示すように、CSR アドレスの上位 4 ビット（csr [11：8]）を使用して、CSR の読み取りおよび書き込みアクセシビリティを特権レベルに従ってエンコードします。
上位 2 ビット（csr [11:10]）は、レジスタが読み/書き（00,01,10）か読み取り専用（11）かを示します。
次の 2 ビット（csr [9：8]）は、CSR にアクセスできる最低の特権レベルをエンコードします。

CSR アドレス規約では、CSR アドレスの上位ビットを使用して、規定のアクセス特権をエンコードします。
これにより、ハードウェアのエラーチェックが簡素化され、より大きな CSR スペースが提供されますが、CSR のアドレス空間へのマッピングが制限されます。

実装により、特権レベルでは、許可されていない CSR アクセスを低特権レベルでトラップして、これらのアクセスを傍受することができます。
この変更は、特権の低いソフトウェアにとっては透過的でなければなりません。

存在しない CSR にアクセスしようとすると、不正な命令例外が発生します。
適切な権限レベルを持たない CSR にアクセスしたり、読み取り専用レジスタを書き込んだりすると、不正な命令例外も発生します。
読み出し/書き込みレジスタには、読み出し専用ビットが含まれている場合があります。この場合、読み出し専用ビットへの書き込みは無視されます。

CSR アドレス			16 進	使用とアクセシビリティ
[11:10]	[9:8]	[7:6]		
ユーザー CSRs				
00	00	XX	0x000-0x0FF	標準 読み／書き
01	00	XX	0x400-0x4FF	標準 読み／書き
10	00	XX	0x800-0x8FF	非標準 読み／書き
11	00	00-10	0xC00-0xCBF	標準 読み取り専用
11	00	11	0xCC0-0xCFF	非標準 読み取り専用
監督者 CSRs				
00	01	XX	0x100-0x1FF	標準 読み／書き
01	01	00-10	0x500-0x5BF	標準 読み／書き
01	01	11	0x5C0-0x5FF	非標準 読み／書き
10	01	00-10	0x900-0x9BF	標準 読み／書き
10	01	11	0x9C0-0x9FF	非標準 読み／書き
11	01	00-10	0xD00-0xDBF	標準 読み取り専用
11	01	11	0xDC0-0xDFF	非標準 読み取り専用
予約 CSRs				
XX	10	XX	予約	
マシン CSRs				
00	11	XX	0x300-0x3FF	標準 読み／書き
01	11	00-10	0x700-0x79F	標準 読み／書き
01	11	10	0x7A0-0x7AF	標準 読み／書き デバッグ CSRs
01	11	10	0x7B0-0x7BF	デバッグモードのみ CSRs
01	11	11	0x7C0-0x7FF	非標準 読み／書き
10	11	00-10	0xB00-0xBBF	標準 読み／書き
10	11	11	0xBC0-0xBFF	非標準 読み／書き
11	11	00-10	0xF00-0xFBF	標準 読み取り専用
11	11	11	0xFC0-0xFFF	非標準 読み取り専用

表 2.1：RISC-V CSR アドレス範囲の割り当て。

表 2.1 はまた、標準と非標準の使用の間に CSR アドレスを割り当てる規則を示しています。
非標準的な使用のために予約された CSR アドレスは、将来の標準拡張によって再定義されません。

私たちは、シャドー CSRs のための CSR スペースの明示的な割り当てを廃止し、他の CSRs を割り当てた方がより柔軟になるようにしました。
シャドー CSRs は、適切な R / W 空間に追加することができます。
カウンタは、現在の仕様では唯一のシャドーされた CSR です。

シャドー CSRs は読み取り／書き込み・アドレスを提供し、より高い特権レベルでより低い特権レベルで読取り専用のレジスタを変更することができます。
1 つの特権レベルで既に読取り/書込みシャドウ・アドレスが割り当てられている場合は、より高い特権レベルであれば、同じレジスタへの読み取り/書き込みアクセスに同じ CSR アドレスを使用できます。

効果的な仮想化では、仮想化された環境内で可能な限り多くの命令をネイティブに実行する必要があり、特権アクセスは仮想マシンモニタ[1]にトラップします。
より低い特権レベルで読み取り専用の CSRs は、より高い特権レベルで読み書き可能にされている場合、別の CSR アドレスにシャドーイングされます。
これにより、許可された低特権アクセスのトラップを回避しながら、不正アクセスのトラップを引き起こします。←発生させます。

マシンモード標準の読み書き可能な CSR 0x7A0-0x7BF は、デバッグシステムで使用するために予約されています。
実装は、これらのレジスタへのマシンモードアクセスで不正な命令例外を発生させるべきです。←必要があります。

-- 2018/10/27

2.2 CSR リスト

表 2.2-2.5 に、現在 CSR アドレスが割り当てられている CSRs の一覧を示します。
タイマー、カウンタ、および浮動小数点 CSRs は、標準的なユーザレベルの CSR であり、N 拡張によって追加されたユーザトラップレジスタも追加されています。
他のレジスタは、次の章で説明するように、特権コードによって使用されます。
すべての実装ですべてのレジスタが必要というわけではないことに注意してください。

番号	特権	名前	説明
			ユーザートラップ設定
0x000	URW	ustatus	ユーザー状態レジスタ。
0x004	URW	uie	ユーザー割り込み有効レジスタ。
0x005	URW	utvec	ユーザトラップハンドラのベースアドレス。
			ユーザートラップ処理
0x040	URW	uscratch	ユーザトラップハンドラのためのスクラッチレジスタ。
0x041	URW	uepc	ユーザー例外プログラムカウンタ。
0x042	URW	ucause	ユーザトラップの原因。
0x043	URW	utval	ユーザーのアドレスまたは命令が不正。
0x044	URW	uip	ユーザー割り込み保留中。
			ユーザ浮動小数点 CSRs
0x001	URW	fflags	浮動小数点発生例外。
0x002	URW	frm	浮動小数点動的丸めモード。
0x003	URW	fcsr	浮動小数点制御および状態レジスタ(frm + fflags)。
			ユーザカウンタ/タイマ
0xC00	URO	cycle	RDCYCLE 命令のサイクルカウンタ。
0xC01	URO	time	RDTIME 命令のタイマ。
0xC02	URO	insret	RDINSTRET 命令の命令リタイヤカウンタ。
0xC03	URO	hpmcounter3	パフォーマンス監視カウンタ
0xC04	URO	hpmcounter4	パフォーマンス監視カウンタ
		：	
0xC1F	URO	hpmcounter31	パフォーマンス監視カウンタ
0xC80	URO	cycleh	cycle の上位 32 ビット、RV32I のみ
0xC81	URO	timeh	time の上位 32 ビット、RV32I のみ。
0xC82	URO	instreth	instret の上位 32 ビット、RV32I のみ。
0xC83	URO	hpmcounter3h	hpmcounter3 の上位 32 ビット、RV32I のみ。
0xC84	URO	hpmcounter4h	hpmcounter4 の上位 32 ビット、RV32I のみ。
		：	
0xC9F	URO	hpmcounter31h	hpmcounter31 の上位 32 ビット、RV32I のみ。

表 2.2：現在割り当てられている RISC-V ユーザーレベルの CSR アドレス。

番号	特権	名前	説明
			監督者トラップ設定
0x100	SRW	sstatus	監督者状態レジスタ。
0x102	SRW	sedeleg	監督者例外委任登録。
0x103	SRW	sideleg	監督者割り込み委譲レジスタ。
0x104	SRW	sie	監督者割り込み有効レジスタ。
0x105	SRW	stvec	監督者トラップハンドラのベースアドレス。
0x106	SRW	scountern	監督者カウンタ有効。
			監督者トラップの処理
0x140	SRW	sscratch	監督者トラップハンドラのスクラッチレジスタ。
0x141	SRW	sepc	監督者例外プログラムカウンタ。
0x142	SRW	scause	監督者トラップの原因。
0x143	SRW	stval	監督者の不良アドレスまたは命令。
0x144	SRW	sip	監督者割り込み保留中。
			監督者保護と翻訳
0x180	SRW	satp	監督者のアドレス変換と保護

表 2.3：現在割り当てられている RISC-V 監督者レベルの CSR アドレス。

-- scratch register ってなんだろ。scratch って削るとか、引っ掻くっていういみだよね。

番号	特権	名前	説明
			マシン情報レジスタ
0xF11	MRO	mvendorid	ベンダー ID。
0xF12	MRO	marchid	アーキテクチャ ID。
0xF13	MRO	mimpid	実装 ID。
0xF14	MRO	mhartid	ハードウェアスレッド ID。
			マシントラップ設定
0x300	MRW	mstatus	マシン状態レジスタ。
0x301	MRW	misa	ISA と拡張機能
0x302	MRW	medeleg	マシン例外委譲レジスタ。
0x303	MRW	mideleg	マシン割り込み委譲レジスタ。
0x304	MRW	mie	マシン割込み許可レジスタ。
0x305	MRW	mtvec	マシントラップハンドラのベースアドレス。
0x306	MRW	mcounteren	マシンカウンター有効。
			マシントラップの処理
0x340	MRW	mscratch	マシントラップハンドラのスクラッチレジスタ。
0x341	MRW	mepc	マシン例外プログラムカウンタ。
0x342	MRW	mcause	マシントラップ原因。
0x343	MRW	mtval	マシンのアドレスまたは命令が不正。
0x344	MRW	mip	マシン割り込み保留中。
			マシンの保護と翻訳
0x3A0	MRW	pmpcfg0	物理メモリ保護構成。
0x3A1	MRW	pmpcfg1	物理メモリ保護構成、RV32 のみ。
0x3A2	MRW	pmpcfg2	物理メモリ保護構成。
0x3A3	MRW	pmpcfg3	物理メモリ保護構成、RV32 のみ。
0x3B0	MRW	pmpaddr0	物理メモリ保護アドレスレジスタ。
0x3B1	MRW	pmpaddr1	物理メモリ保護アドレスレジスタ。
		：	：
0x3BF	MRW	pmpaddr15	物理メモリ保護アドレスレジスタ。

表 2.4：現在割り当てられている RISC-V マシンレベルの CSR アドレス。

番号	特権	名前	説明
		マシンカウンタ/タイマ	
0xB00	MRW	mcycle	マシンサイクルカウンタ。
0xB02	MRW	minstret	マシン命令 – 退役カウンタ。
0xB03	MRW	mhpmcounter3	マシン性能監視カウンタ。
0xB04	MRW	mhpmcounter4	マシン性能監視カウンタ。
		:	
0xB1F	MRW	mhpmcounter31	マシン性能監視カウンタ。
0xB80	MRW	mcycleh	mcycle の上位 32 ビット、RV32I のみ。
0xB82	MRW	minstreth	minstret の上位 32 ビット、RV32I のみ。
0xB83	MRW	mhpmcounter3h	mhpmcounter3 の上位 32 ビット、RV32I のみ。
0xB84	MRW	mhpmcounter4h	mhpmcounter4 の上位 32 ビット、RV32I のみ。
		:	
0xB9F	MRW	mhpmcounter31h	mhpmcounter31 の上位 32 ビット、RV32I のみ。
		マシンカウンタ設定	
0x323	MRW	mhpmevent3	マシン性能監視イベントセレクタ
0x324	MRW	mhpmevent4	マシン性能監視イベントセレクタ
0x33F	MRW	mhpmevent31	マシン性能監視イベントセレクタ
		デバッグ/追跡レジスタ（デバッグモードと共有）	
0x7A0	MRW	tselect	デバッグ/追跡トリガレジスタの選択。
0x7A1	MRW	tdata1	最初のデバッグ/追跡トリガデータレジスタ。
0x7A2	MRW	tdata2	第 2 のデバッグ/追跡トリガデータレジスタ。
0x7A3	MRW	tdata3	第 3 のデバッグ/ト追跡トリガデータレジスタ。
		デバッグモードレジスタ	
0x7B0	DRW	dcsr	デバッグ制御および状態レジスタ。
0x7B1	DRW	dpc	デバッグ PC。
0x7B2	DRW	dscratch	デバッグスクラッチレジスタ。

表 2.5：現在割り当てられている RISC-V マシンレベルの CSR アドレス。

2.3 CSR フィールドの仕様

以下の定義および略語は、CSR 内のフィールドの動作を指定する際に使用されます。

予約済み書き込み無視、読み込み無視値（WIRI） ← リードした値は無視してねってこと

読み取り専用と読み取り/書き込みレジスタの中には、将来の使用のために予約された読み取り専用フィールドがあるものもあります。
これらの予約済みの読み取り専用フィールドは、読み取り時に無視する必要があります。
これらのフィールドへの書き込みは、CSR 全体が読み取り専用でない限り、無効です。この場合、書き込みによって不正な命令例外が発生する可能性があります。
これらのフィールドは、レジスタ記述で WIRI とラベル付けされています。

予約済み書き込みは値を保持し、無視値を読み込みます（WPRI）

一部の読み取り/書き込みフィールド全体は、将来の使用のために予約されています。← some はいくつか、whole は全体 約合ってるるかな
ソフトウェアはこれらのフィールドから読み取った値を無視し、同じレジスタの他のフィールドに値を書き込むときにこれらのフィールドに保持されている値を保持する必要があります。←予約されてるので読んだ値は無視してね。書いた時に書き換わっちゃダメよってことかな。
これらのフィールドは、レジスタ記述で WPRI と表示されます。

ソフトウェアモデルを簡素化するために、CSR 内の以前に予約されたフィールドの将来互換性のある将来の定義は、非原子的な読み取り/修正/書き込みシーケンスが CSR の他のフィールドを更新するために使用される可能性に対処しなければならなりません。←対処する必要があります。
あるいは(また)、元の CSR 定義では、サブフィールドをアトミックにしか更新できない(アトミックにのみ更新できるように)ことを指定しなければならず、中間値が合法(有効)でない場合に問題となる 2 命令のクリアビット/セットビットシーケンスが一般的に必要となる可能性があります。← うーん、いまいちわからん

書き込み/読み取り 専用(のみ)の有効値（WLRL）

いくつかの読み取り/書き込み CSR フィールドは、可能なビットエンコーディングのサブセットのみの動作を指定し、その他のビットエンコーディングは予約されています。
ソフトウェアは、そのようなフィールドに正当な値以外を書き込むべきではなく、最後の書き込みが正当な値でないか、別の操作（例えばリセット）からレジスタが書き込まれていない限り、 レジスタを正しい値に設定します。
これらのフィールドは、レジスタの説明で WLRL というラベルが付けられています。

ハードウェアの実装では、サポートされている値を区別するのに十分な状態ビットを実装するだけで済みますが(実装する必要がありますが)、読み込み時にサポートされている値の完全な指定されたビットエンコーディングを常に返す必要があります。

命令がサポートされていない値を CSR フィールドに書き込もうとすると、実装は許可されますが、不正命令例外を発生させる必要はありません。
ハードウェア実装は、最後の書き込みが不正な値であったときに CSR フィールドの読み込みで任意のビットパターンを返すことができますが、返される値は前回書き込まれた値に依存します。

すべての値を書き込み、有効値を読み取り（WARL）

いくつかの読み取り/書き込み CSR フィールドは、ビットエンコーディングのサブセットに対してのみ定義されますが、読み取られるたびに正しい値を返すことを保証しながら任意の値を書き込むことができます。
CSR の記述に他に副作用がないと仮定すると、サポートされている値の範囲は、希望の設定を書き込んだり、その値が保持されているかどうかを調べることによって判断できます。
これらのフィールドは、レジスタの説明で WARL というラベルが付けられています。

実装では、サポートされていない値の WARL フィールドへの書き込みに関する例外は発生しません。
実装は、特定の不正な値が書き込まれた後、常に確定的に同じ正当な値を返す必要があります。

-- 2018/10/28

第 3 章

マシンレベル ISA、バージョン 1.10

この章では、マシンモード（M モード）で使用可能なマシンレベルの操作について説明します。これは、RISC-V システムで最高の権限モードです。
RISC-V ハードウェア実装では、M モードのみが必須(唯一)の特権モードです。
M モードは、ハードウェアプラットフォームへの低レベルアクセスに使用され、リセット時に最初に入力されるモードです。
M モードは、ハードウェアに直接実装するにはあまりにも困難または高価な機能を実装するためにも使用できます。
RISC-V マシンレベルの ISA には、サポートされている他の特権レベルとハードウェア実装のその他の詳細に応じて拡張された共通のコアが含まれています。

3.1 マシンレベルの CSR

このセクションで説明するマシンレベルの CSR に加えて、M モードコードはより低い特権レベルですべての CSR にアクセスできます。

3.1.1 マシン ISA レジスタミサ

ミサ CSR は、ハートでサポートされている ISA を報告する XLEN ビットの WARL 読み書きレジスタです。
このレジスタはどの実装でも読み込み可能でなければなりませんが、misa レジスタが実装されていないことを示すために 0 の値が返されるため、別個の非標準メカニズムによって CPU 能力を判断する必要があります。
↑ゼロが返されると misa レジスタが実装されていないことがわかる。

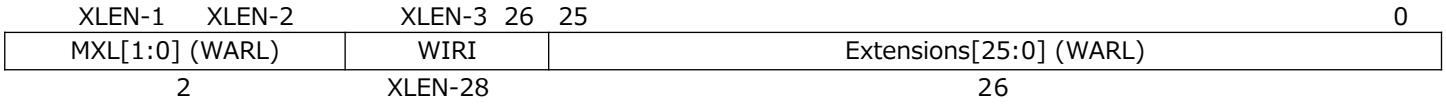


図 3.1：マシン ISA レジスタ（ミサ）。

MXL（Machine XLEN）フィールドは、表 3.1 に示すようにネイティブベースの整数 ISA 幅をエンコードします。
MXL フィールドは、複数のベース ISA 幅をサポートする実装では書き込み可能です。
M モードの有効 XLEN、M-XLEN は MXL の設定で与えられ、ミサがゼロの場合は固定値です。
MXL フィールドは、リセット時に常にサポートされている最も広い ISA バリアントに設定されます。

MXL	XLEN
1	32
2	64
3	128

表 3.1：ミサの XML フィールドのエンコーディング

ベース幅は返されたミサ値の符号による分岐と、場合によっては 1 つだけ左にシフトと、符号による 2 つ目の分岐をすることによって、迅速に確認できます。
↑ここは sign を符号とするか、なにか目印、標識のような意味にするかでちょと迷う。
これらのチェックは、マシンのレジスタ幅（XLEN）を知らずにアセンブリコードで記述することができます。
ベース幅は XLEN = 2MXL + 4 で与えられます。
ミサがゼロの場合は、即値 4 をレジスタに置き、レジスタを一度に 31 ビットだけ左にシフトすることによって、ベース幅を見つけることもできます。
1 回シフトした後にゼロになると、マシンは RV32 になります。
2 回シフトした後にゼロになると、マシンは RV64、それ以外の場合は RV128 になります。

MXL がサポートされている XLEN よりも小さい値に設定されている場合、すべての演算は、設定された XLEN よりも上位のソース・オペランド・レジスタ・ビットを無視し、サポートされている最も広い XLEN をデスティネーション・レジスタに埋めるために結果を符号拡張する必要があります。

実実装では、実装で定義された動作を避けるために、常に、基本ハードウェア・レジスタ全体を定義された値で埋める必要があります。

拡張フィールドは、アルファベット 1 文字あたり 1 ビットのビットを有する標準拡張の存在を符号化する（ビット 0 は拡張子「A」の存在を符号化し、ビット 1 は拡張子「B」を符号化し、「Z」を符号化するビット 25 まで符号化する））。
"I"ビットは RV32I、RV64I、RV128I ベース ISA に設定され、 "E"ビットは RV32E に設定されます。
拡張機能は、実装がサポートされている ISA を変更できるようにする書き込み可能なビットを含むことができる WARL フィールドです。
リセット時に、拡張フィールドにはサポートされている拡張機能の最大セットが含まれている必要があり、両方が利用可能であるなら、私は E より上を選ぶべきです。

"G"ビットはエスケープとして使用され、標準拡張名のより大きなスペースに拡張することができます。

G は組み合わせ IMAFD を示すために使用されるため、ミサ CSR には冗長であるため、追加標準拡張が存在することを示すビットを予約します。

"U"ビットと "S"ビットは、それぞれユーザモードとスーパバイザモードがサポートされている場合に設定されます。

非標準の拡張があれば "X"ビットがセットされます。

---

misa CSR は、CPU 機能の基本カタログをマシンモードコードに公開します。
マシンモードでは、他のマシンレジスタを調査し、ブートプロセスの一部としてシステム内の他の ROM ストレージを調べることにより、より詳細な情報を得ることができます。
各特権レベルで利用可能な機能を決定するために、CPU レジスタを読み取る代わりに、低い特権レベルで環境呼び出しを実行する必要があります。
これにより、仮想化レイヤーはあらゆるレベルで観察される ISA を変更でき、ハードウェア設計に負担をかけることなく、より豊富なコマンドインターフェイスをサポートします。

ビット	文字	説明
0	A	原子の拡張
1	B	ビット演算拡張用に仮予約
2	C	圧縮拡張
3	D	倍精度浮動小数点拡張
4	E	RV32E ベース ISA
5	F	単精度浮動小数点拡張
6	G	追加の標準拡張機能が存在
7	H	予約
8	I	RV32I/64I/128I ベース ISA
9	J	動的に翻訳された言語拡張のために仮予約
10	K	予約
11	L	10 進浮動小数点数の拡張のために仮予約
12	M	整数乗算/除算拡張
13	N	サポートされているユーザーレベルの割り込み
14	O	予約
15	P	パック SIMD 拡張の仮予約
16	Q	四倍精度浮動小数点数の拡張
17	R	予約
18	S	監督者モードの実装
19	T	トランザクションメモリ拡張用に仮予約
20	U	ユーザーモードの実装
21	V	ベクター拡張のために仮予約
22	W	予約
23	X	標準以外の拡張機能が存在
24	Y	予約
25	Z	予約

表 3.2：ミサの拡張フィールドのエンコード。将来使用するために予約されているすべてのビットは、読み込み時にゼロを返す必要があります。

3.1.2 マシンベンダー ID レジスタ mvendorid

mvendorid CSR は、コアのプロバイダの JEDEC 製造元 ID を提供する XLEN ビットの読み取り専用レジスタです。
このレジスタは、どの実装でも読み込み可能でなければなりませんが、フィールドが実装されていないこと、または非商用実装であることを示すために値 0 を返すことができます。

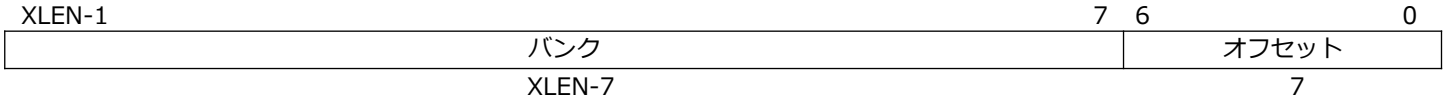


図 3.2：ベンダー ID レジスタ（mvendorid）。

JEDEC 製造者 ID は、通常、各バイトの最上位ビットに奇数パリティビットを有する 0x7f に等しくない 1 バイト ID によって終端される 1 バイト連続コード 0x7f のシーケンスとして符号化されます。
mvendorid は、バンクフィールドの 1 バイトの連続コードの数をエンコードし、オフセットフィールドの最後のバイトをエンコードして、パリティビットを破棄します。
たとえば、JEDEC 製造元 ID 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x8a（12 個の継続コードの後ろに 0x8a）は、mvendorid フィールドに 0x60a としてエンコードされます。
↑0x60a は 0x7f の数が 12、16 進で 0xc。8a の最上位パリティを飛ばすと 0x0a、0xc をパリティの位置から上に詰めると 06a。うーん、合ってるだろうか。

以前はベンダー ID は RISC-V 財団によって割り当てられた番号でしたが、これは JEDEC が製造者 ID 標準を維持する作業を複製しています。
執筆時点では、JEDEC にメーカー ID を登録するには 1 回限りの 500 ドルのコストがかかります。

3.1.3 マシンアーキテクチャ ID レジスタ marchid

marchid CSR は、ハートのベースマイクロアーキテクチャをエンコードする XLEN ビットの読み取り専用レジスタです。
このレジスタはどの実装でも読み込み可能でなければなりませんが、フィールドが実装されていないことを示す値 0 を返すことができます。
mvendorid と marchid の組み合わせは、実装されているハートマイクロアーキテクチャのタイプを一意に識別する必要があります。

図 3.3：マシンアーキテクチャ ID レジスタ（marchid）。

オープンソースのプロジェクトアーキテクチャ ID は、RISC-V 財団によってグローバルに割り当てられ、ゼロ以外のアーキテクチャ ID とゼロの最上位ビット（MSB）を持っています。
商用アーキテクチャ ID は、各商用ベンダーによって個別に割り当てられますが、MSB が設定されていて、残りの XLEN-1 ビットにゼロを含めることはできません。
↑ 残りの XLEN-1 ビットにゼロを含めることはできません って All 1 なん？ ID 区別できんじゃんって思ったけど、ここは XLEN-1 は 0 にしてはダメってことだと思う。

その目的は、アーキテクチャー ID が、特定の組織ではなく開発が行われるレポに関連するマイクロアーキテクチャーを表すことです。
オープンソース設計の商業的製作では、元のアーキテクチャ ID を保持する必要があります。（ライセンスによって必要となる可能性があります）
これにより、断片化やツールサポートのコストを削減するだけでなく、帰属を提供するのに役立ちます。
オープンソースのアーキテクチャー ID は財団によって管理されるべきであり、リリースされ機能しているオープンソースプロジェクトにのみ割り当てられるべきです。
商用アーキテクチャー ID は、登録されたベンダーによって個別に管理することができますが、ベンダーがクローズドソースとオープンソースの両方のマイクロアーキテクチャーを使用したい場合には、オープンソースアーキテクチャー ID（MSB セット）とは別の ID が必要です。

次の実装フィールド内で採用された規約は、組織を含む同じアーキテクチャ設計のブランチを分離するために使用できます。
ミサレジスタはまた、構成文字列が存在する場合、デザインの異なるバリアントを区別するのにも役立ちます。

3.1.4 マシン実装 ID レジスタ mimpid

mimpid CSR は、プロセッサ実装のバージョンを一意にエンコードします。
このレジスタはどの実装でも読み込み可能でなければなりませんが、フィールドが実装されていないことを示すために値 0 を返すことができます。
実装の値は、RISC-V プロセッサ自体の設計を反映するものであり、周囲のシステムではありません。
↑この文は Google 翻訳と Bing 翻訳で真逆の訳をする。多分こっちの Google 翻訳が正しいと思う。

図 3.4：マシン実装 ID レジスタ（mimpid）。

このフィールドのフォーマットは、アーキテクチャーのソースコードのプロバイダーに委ねられていますが、先行または後続の 0 を持たない 16 進文字列として標準ツールによって印刷されることがよくありますが、人間の可読性を容易にするために、インプリメンテーション値をニブル境界で整列されたサブフィールドで左揃えにすることができます。　（すなわち、最も重要なニブルから下に埋められる）

3.1.5 ハート ID レジスタ mhartid

mhartid CSR は、コードを実行するハードウェアスレッドの整数 ID を含む XLEN ビットの読み取り専用レジスタです。
このレジスタは、どの実装でも読み込み可能でなければなりません。
ハート ID は必ずしもマルチプロセッサーシステムでは連続して番号付けされるとは限りませんが、少なくとも 1 つのハートはハート ID がゼロでなければなりません。

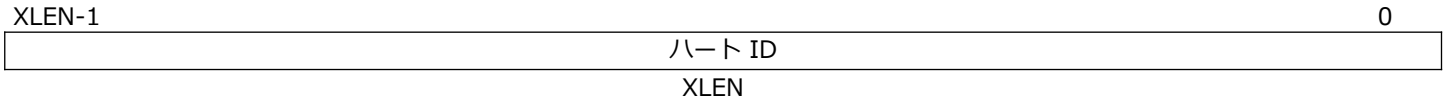


図 3.5：ハート ID レジスタ（mhartid）。

特定のケースでは、正確に 1 つのハートが（例えばリセット時に）いくつかのコードを実行することを保証するようにする必要があります。したがって、1 つのハートには既知のハート ID がゼロである必要があります。
効率のために、システム実装者は、システムで使用される最大の hartID の大きさを減らすことを目指すべきです。

3.1.6 マシンステータスレジスタ（mstatus）

mstatus レジスタは、RV32 では図 3.6、RV64 と RV128 では図 3.7 のようにフォーマットされた XLEN ビットの読み書き可能レジスタです。
mstatus レジスタは、ハートの現在の動作状態を追跡し、制御します。
mstatus レジスタの制限されたビューは、それぞれ S レベルおよび U レベルの ISA に sstatus および ustatus レジスタとして表示されます。

31	30 … 23	22	21	20	19	18	17
SD	WPRI	TSR	TW	TVM	MXR	SUM	MPRV
1	8	1	1	1	1	1	1

16	15 14	13 12	11 10	9	8	7	6	5	4	3	2	1	0
XS[1:0]	FS[1:0]	MPP[1:0]	WPRI		SPP	MPIE	WPRI	SPIE	UPIE	MIE	WPRI	SIE	UIE
2	2	2	2		1	1	1	1	1	1	1	1	1

図 3.6：RV32 のマシンモードステータスレジスタ（mstatus）

XLEN-1	XLEN-2 36	35 34	33 32	31 … 23	22	21	20	19	18	17
SD	WPRI	SXL[1:0]	UXL[1:0]	WPRI	TSR	TW	TVM	MXR	SUM	MPRV
1	XLEN-37	2	2	9	1	1	1	1	1	1

16	15 14	13 12	11 10	9	8	7	6	5	4	3	2	1	0
XS[1:0]	FS[1:0]	MPP[1:0]	WPRI		SPP	MPIE	WPRI	SPIE	UPIE	MIE	WPRI	SIE	UIE
2	2	2	2		1	1	1	1	1	1	1	1	1

図 3.7：RV64 と RV128 のマシンモードステータスレジスタ（mstatus）

3.1.7 mstatus レジスタの特権とグローバル割り込みイネーブルスタック

割込み許可ビット、MIE、SIE、および UIE は、特権モードごとに提供されます。
これらのビットは、現在の特権レベルで割り込みハンドラに関するアトミック性を保証するために主に使用されます。
ハートが特権モード x で実行されている場合、x IE = 1 のとき割り込みが有効になります。
低特権モードの割込みは常に無効になりますが、高特権モードの割込みは常に有効です。
より高い特権レベルのコードは、より低い特権レベルに制御を委譲する前に、割込みごとのイネーブルビットを使用するこ選択された割込みを無効にすることができます。

---

xIE ビットは、mstatus の下位ビットに配置され、単一の CSR 命令でアトミックにセットまたはクリアすることができます。

ネストされたトラップをサポートするために、各特権モード x には、割り込みイネーブルビットと特権モードの 2 つのレベルのスタックがあります。
x PIE は、トラップの前に割り込みイネーブルビットの値を保持し、xPP は前の特権モードを保持します。
xPP フィールドは x までの特権モードしか保持できないため、MPP は 2 ビット幅、SPP は 1 ビット幅、UPP は暗黙的にゼロです。
トラップが特権モード y から特権モード x に移されると、x PIE は x IE の値に設定されます。 x IE は 0 に設定されます。 xPP は y に設定されます。

低特権モードでは、通常、エントリ（エントリ）時に割り込みが禁止された状態で、上位の特権モードでトラップ（同期または非同期）が行われます。
上位レベルのトラップハンドラは、スタックされた情報を使用してトラップを処理して戻り、または、中断されたコンテキストに直ちに戻ってこない場合、割り込みを再び有効にする前に特権スタックを保存し、スタックごとに 1 つのエントリだけが必要です。

MRET、SRET、または URET 命令は、それぞれ M モード、S モード、または U モードのトラップから戻るために使用されます。
xRP 命令を実行するとき、xPP が値 y を保持すると仮定すると、x IE は x PIE に設定される。 特権モードは y に変更されます。 x PIE は 1 に設定されます。 xPP は U（ユーザーモードがサポートされていない場合は M）に設定されます。

スタックがポップされると、サポートされている最低限の割り込み特権モードがスタックの最下部に追加され、無効なエントリがポップされる原因となるエラーを捕捉するのに役立ちます。

xPP フィールドは、サポートされている特権モード（x および x より小さい実装された特権モードを含む）だけを格納できるようにする必要がある WLRL フィールドです。

マシンが U モードと M モードのみを提供する場合、MPP の 00 または 11 のいずれかを表すためにハードウェア記憶ビットは 1 つだけ必要です。

ユーザーレベルの割り込みは、オプションの拡張であり、ISA 拡張文字 N が割り当てられています。
ユーザーレベルの割り込みを省略すると、UIE ビットと UPIE ビットはゼロに固定されます。
他のすべてのサポートされている特権モード x では、x IE と x PIE はハードワイヤードであってはなりません。
↑ここも Google 翻訳と Bing 翻訳で真逆の意味に翻訳される。Bing 翻訳は否定形に弱いのかな。

ユーザーレベルの割り込みは主に、M モードと U モードのみのセキュアな組み込みシステムをサポートすることを目的としていますが、ユーザーレベルのトラップ処理をサポートする Unix ライクなオペレーティングシステムを実行するシステムでもサポートできます。

以前 mstatus で H モードサポートのために割り当てられていたフィールドは、現在 WPRI フィールドとして予約されています。
既存の実装との後方互換性を低下させるために、これらのフィールドを削除した後は、レジスタを圧縮しませんでした。

3.1.8 mstatus レジスタのベース ISA 制御

RV64 および RV128 システムの場合、SXL および UXL フィールドは、それぞれ S モードおよび U モードの XLEN の値を制御する WARL フィールドです。
これらのフィールドの符号化は、表 3.1 に示すミサの MXL フィールドと同じです。
S モードおよび U モードで有効な XLEN は、それぞれ S-XLEN および U-XLEN と呼ばれます。

RV32 システムの場合、SXL および UXL フィールドは存在せず、S-XLEN = 32 および U-XLEN = 32 です。

RV64 および RV128 システムの場合、S モードがサポートされていない場合、SXL はゼロにハードワイヤードされます。
それ以外の場合は、S-XLEN の現在の値をエンコードするのは WARL フィールドです。
特に、S-XLEN = M-XLEN となるように SXL を実装することがあります。

RV64 および RV128 システムの場合、U モードがサポートされていない場合、UXL はゼロに固定配線されます。
それ以外の場合は、U-XLEN の現在の値をエンコードするのは WARL フィールドです。
特に、U-XLEN = M-XLEN となるように UXL を実装することができます。

どのモードでも XLEN がサポートされている XLEN よりも小さい値に設定されると、すべての演算は、設定された XLEN より上のソース・オペランド・レジスタ・ビットを無視しなければならず、サポートされている最も広い XLEN をデスティネーション・レジスタに埋めるために結果を符号拡張する必要があります。

ハードウェアの複雑さを軽減するため、アーキテクチャでは、低特権モードで XLEN 設定が次の高位特権モード以下であることを確認しません。
実際には、このような設定はほとんど常にエラーになりますが、この場合でもマシンの動作は明確に定義されています。

3.1.9 mstatus レジスタのメモリ特権

MPRV（Modify PRiVilege）(権限の変更)ビットは、すべての特権モードでロードおよびストアが実行される特権レベルを変更します。
MPRV = 0 の場合、変換と保護は正常に動作します。
MPRV = 1 の場合、現在の特権モードが MPP に設定されているかのように、読み込みと保存のメモリアドレスは変換され、保護されます。
命令のアドレス変換と保護は影響を受けません。
U モードがサポートされていない場合、MPRV は 0 にハードワイヤードされます。

MXR（Make eXecutable Readable）(実行可能ファイルを読めるようにする) ビットは、ロードが仮想メモリにアクセスする際の特権を変更します。
MXR = 0 の場合、読み込み可能とマークされたページからの読み込みのみが成功します（図 4.15 の R = 1）。
MXR = 1 の場合、読み取り可能または実行可能（R = 1 または X = 1）のいずれかのマークが付けられたページからの読み込みは成功します。
ページベースの仮想メモリが有効でない場合、MXR は効果がありません。
S モードがサポートされていない場合、MXR は 0 にハードワイヤードされます。

MPRV および MXR 機構は、ミスアラインされた読み込みおよび保存などの欠落したハードウェア機能をエミュレートする M モードルーチンの効率を改善するために考案されました。
MPRV は、ソフトウェアでアドレス変換を実行する必要性を排除します。
MXR を使用すると、実行専用とマークされたページから命令語をロードできます。
簡単にするため、MPRV と MXR は特権モードに関係なく有効ですが、通常の使用ではマシンモードの短いシーケンスに対してのみ有効になります。

SUM（スーパーバイザユーザメモリアクセス許可）ビットは、S モードの読み込み、書き込み、および命令フェッチが仮想メモリにアクセスするための特権を変更します。
SUM = 0 のとき、U モードでアクセス可能なページへの S モードメモリアクセスは、エラーになります。（図 4.15 の U = 1）。
↑ここは Bing 翻訳の方が正しい。Google 翻訳だと最後の fault を飛ばす。うーん。
SUM = 1 の場合、これらのアクセスは許可されます。
ページベースの仮想メモリが有効でない場合、SUM は効果がありません。
SUM は S モードで実行されていないときは通常無視されますが、MPRV = 1 かつ MPP = S のときには有効です。
S モードがサポートされていない場合、SUM は 0 にハードワイヤードされます。

3.1.10 mstatus レジスタの仮想化サポート

TVM（トラップ仮想メモリ）ビットは、スーパバイザ仮想メモリ管理操作の代行受信(インターセプト)をサポートします。
TVM = 1 の場合、S-mode で実行中に satp CSR を読み書きしようとするか SFENCE.VMA 命令を実行しようとすると、不正な命令例外が発生します。
TVM = 0 の場合、これらの操作は S モードで許可されます。
S モードがサポートされていない場合、TVM は 0 にハードワイヤードされています。

TVM メカニズムは、ゲストオペレーティングシステムを U モードで古典的に仮想化するのではなく、S モードで実行できるようにすることで、仮想化の効率を向上させます。
このアプローチは、ほとんどの S モード CSRs へのアクセスをトラップする必要性を排除します。
satp アクセスと SFENCE.VMA 命令をトラッピングすると、シャドウページテーブルを遅延読み込みするのに必要なフックが提供されます。

TW（タイムアウト待機）ビットは、WFI 命令のインターセプトをサポートします（3.2.3 項を参照）。
TW = 0 のとき、WFI 命令は S モードで許可されます。
TW = 1 の場合、WFI が Smode で実行され、実装固有の制限時間内に完了しない場合、WFI 命令は不正な命令トラップを引き起こします。
制限時間は常に 0 にすることができ、この場合 TW = 1 のときに WFI は S モードで常に不正な命令トラップを引き起こします。
S モードがサポートされていない場合、TW は 0 にハードワイヤードされます。

WFI 命令をトラップすると、現在のゲストで無駄にアイドリングするのではなく、別のゲスト OS に切り替えることができます。

TSR（トラップ SRET）ビットは、スーパバイザ例外リターン命令 SRET のインターセプトをサポートします。
TSR = 1 のとき、S モードで実行中に SRET を実行しようとすると、不正な命令例外が発生します。
TSR = 0 のときは、S モードで許可されます。
S モードがサポートされていない場合、TSR は 0 にハード配線されます。

SRET をトラッピングすることは、それを提供しない実装で拡張仮想化機構（第 5 章参照）をエミュレートするために必要です。

3.1.11 mstatus レジスタの拡張コンテキスト状況

実質的な拡張をサポートすることは、RISC-V の主な目標の 1 つです。したがって、変更されていない特権モードのコード、特にスーパバイザレベルの OS が任意のユーザーモードの状態拡張をサポートできるようにする標準インターフェイスを定義します。

今日まで、浮動小数点 CSR およびデータレジスタ以外の追加の状態を定義する標準的な拡張はありません。

FS [1：0]読み取り/書き込みフィールドと XS [1：0]読み取り専用フィールドは、浮動小数点ユニットの現在の状態、および他のユーザモードの拡張をそれぞれ示す設定と追跡によってコンテキストの保存と復元のコストを削減するために使用されます
FS フィールドは、CSR fcsr および浮動小数点データレジスタ f0～f31 を含む浮動小数点ユニットのステータスをエンコードし、XS フィールドは、追加のユーザモード拡張および関連するステートのステータスをエンコードします。
これらのフィールドは、状態の保存または復元が必要かどうかを迅速に判断するためのコンテキスト切り替えルーチンによってチェックできます。
保存または復元が必要な場合は、通常、プロセスを効果的に最適化するために追加の指示と CSRs が必要です。

この設計では、ほとんどのコンテキストスイッチが浮動小数点ユニットまたはその他の拡張のいずれかまたは両方で状態を保存/復元する必要がないため、SD ビットによる高速チェックが可能になると予想しています。

FS フィールドと XS フィールドは、表 3.3 に示されているものと同じステータスエンコーディングを使用し、4 つの可能なステータス値はオフ、イニシャル、クリーン、およびダーティです。

状態	FS の意味	XS の意味
0	オフ	すべてオフ
1	イニシャル	いずれもダーティではないか、クリーンではダーティでない、いくつかはクリーン
2	クリーン	いくつかダーティ
3	ダーティ	

表 3.3：FS [1：0]および XS [1：0]ステータスフィールドのエンコード

S モードを実装せず、浮動小数点ユニットを持たないシステムでは、FS フィールドはゼロにハードワイヤードされています。

新しい状態を必要とする追加のユーザー拡張がないシステムでは、XS フィールドはゼロにハードワイヤードされます。
状態付きのすべての追加拡張は、XS 状態に相当する mstatus にローカルフィールドを追加します。
XS フィールドは、表 3.3 に示すように、すべての内線番号のステータスの要約を表します。

XS フィールドは、すべてのユーザー拡張ステータスフィールドにわたって最大ステータス値を効果的に報告しますが、個々の拡張機能は XS とは異なるエンコードを使用できます。

SD ビットは、FS フィールドまたは XS フィールドのいずれかが、拡張されたユーザーコンテキストをメモリに保存する必要があるダーティ状態の存在を示すかどうかを要約する読み取り専用ビットです。
XS と FS の両方がゼロにハードワイヤードされていると、SD も常にゼロになります。

拡張の状態がオフに設定されている場合、対応するステートを読み書きしようとする命令は例外を発生させます。
ステータスがイニシャルの場合、対応する状態は初期の定数値を持つ必要があります。
状態がクリーンの場合、対応する状態は初期値と異なる可能性がありますが、コンテキストスワップに格納されている最後の値と一致します。
ステータスがダーディの場合、対応するステートは、最後のコンテキスト保存以降に変更されている可能性があります。

コンテキスト保存中に責任ある特権コードは、その状態がダーティの場合にのみ対応する状態を書き出す必要があり、その後、その状態をクリーンにリセットすることができます。
コンテキスト復元時には、状態がクリーン（復元時にダーティであってはならない）の場合にのみコンテキストをメモリからロードする必要があります。
ステータスがイニシャルの場合、セキュリティホールが発生しないようコンテキストを復元時に初期の定数値に設定する必要がありますが、これはメモリにアクセスせずに行うことができます。
たとえば、浮動小数点レジスタはすべて即値 0 に初期化できます。

FS および XS フィールドは、コンテキストを保存する前に特権コードによって読み取られます。
FS フィールドは、ユーザコンテキストを再開するときには特権コードによって直接設定され、XS フィールドは個々の拡張のステータスレジスタに書き込むことによって間接的に設定されます。
ステータスフィールドは、特権モードに関係なく、命令の実行中に(も)更新されます。

ユーザーモード ISA への拡張には、追加のユーザーモード状態が含まれることが多く、この状態は基本整数レジスタよりもかなり大きくなる可能性があります。
拡張機能は、一部のアプリケーションでのみ使用することも、単一のアプリケーション内で短期間だけ必要とすることもあります。
パフォーマンスを向上させるために、ユーザモード拡張は、ユーザモードソフトウェアがユニットを初期状態に戻すか、またはユニットをオフにすることを可能にする追加の命令を定義することができます。
↑使わない時はそのユーザーモードを OFF にできる。例えばそのブロックのクロックを止めたり、電源を OFF したりできるってこと。

たとえば、コプロセッサを使用する前に構成する必要があり、使用後にコプロセッサを”未構成”することができます。
未構成状態は、コンテキスト保存の初期状態として表されます。
未構成と次の構成（状態をダーティに設定する）の間で同じアプリケーションが実行中のままである場合、すべての状態がユーザープロセスに対してローカルであるため、実際に未構成命令で状態を再初期化する必要はありません。 すなわち、初期状態は、コンテキスト・リストア時にコプロセッサ・ステートがすべての未構成時ではなく一定値に初期化されるだけでよいです。

ユニットをディスエーブルしてオフ状態にするユーザモード命令を実行すると、後続の命令が再びオンになる前にユニットを使用しようとすると、不正な命令例外が発生します。
ユニットをオンにするユーザモード命令は、そのユニットが別のコンテキストによって使用されている可能性があるため、ユニットの状態が適切に初期化されていることを保証しなければなりません。(確認する必要があります。)

FS の設定を変更しても、浮動小数点レジスタの状態は変化しません。
特に、FS = オフを設定しても状態が破壊されることはなく、FS = イニシャルが内容をクリアするように設定することもありません。
オフに設定すると、他の拡張機能が状態を保持しないことがあります。(保持できません。)

表 3.4 に、FS または XS ステータスビットのすべての可能な状態遷移を示します。
標準の浮動小数点拡張は、ユーザモードの構成解除または無効/有効命令をサポートしていないことに注意してください。

現在の状態 動作	オフ	イニシャル	クリーン	ダーティ
特権コードでのコンテキスト保存時				
保存状態？ 次の状態	いいえ オフ	いいえ イニシャル	いいえ クリーン	はい クリーン
特権コードでのコンテキスト復元時				
復元状態？ 次の状態	いいえ オフ	はい、イニシャルに(へ) イニシャル	はい、メモリから クリーン	N/A N/A
状態を読み出すための命令を実行				
動作？ 次の状態	例外 オフ	例外 イニシャル	例外 クリーン	例外 ダーティ
コンフィグレーションを含むステートの変更命令を実行				
動作？ 次の状態	例外 オフ	実行 ダーティ	実行 ダーティ	実行 ダーティ
未構成ユニットへの命令実行				
動作？ 次の状態	例外 オフ	実行 イニシャル	実行 イニシャル	実行 イニシャル
ユニットを無効にする命令を実行				
動作？ 次の状態	実行 オフ	実行 オフ	実行 オフ	実行 オフ
ユニットを有効にする命令を実行				
動作 次の状態	実行 イニシャル	実行 イニシャル	実行 イニシャル	実行 イニシャル

表 3.4：FS および XS の状態遷移。

-- 2018/11/24

拡張状態を初期化、保存、復元するための標準的な特権命令は、状態を不透明なオブジェクトとして扱うことによって、拡張された拡張状態の詳細から特権コードを隔離するために提供されます。

多くのコプロセッサ拡張は、制限されたコンテキストでのみ使用されます。これにより、ソフトウェアがユニットの設定を安全に解除するか、実行時にユニットを無効にすることさえできます。
これにより、大規模なステートフルコプロセッサのコンテキストスイッチオーバーヘッドが削減されます。

浮動小数点状態を他の拡張状態から分離し、浮動小数点ユニットが存在する場合、浮動小数点レジスタは標準呼び出し規約の一部であるため、ユーザモードソフトウェアは浮動小数点ユニットを無効にすることが安全であるかどうかを知ることができません。

XS フィールドは、追加されたすべての拡張状態の要約を提供しますが、コンテキストの保存と復元のオーバーヘッドをさらに削減するために、拡張で追加のマイクロアーキテクチャビットが維持される可能性があります。

SD ビットは読み出し専用であり、FS または XS ビットのいずれかがダーティ状態（すなわち、SD =（FS == 11）OR（XS == 11））を符号化するときに設定されます。
これにより、特権コードは、整数レジスタセットと PC 以外のコンテキスト保存が不要になったときを素早く判断することができます。

浮動小数点ユニットの状態は、標準命令（F、D、および/または Q）を使用して常に初期化、保存、および復元され、特権付きコードは FLEN を認識して各 f レジスタに予約する適切な領域を決定する必要があります。

スーパバイザレベルの OS では、追加のユーザモードの状態は、追加のコンテキストを一定の最大サイズの不透明なオブジェクトとして扱う SBI 呼び出しを使用して初期化、保存、復元する必要があります。
SBI の初期化、保存、および復元の実装では、コプロセッサ内のマイクロアーキテクチャ状態を初期化、保存、および復元するために、インプリメンテーション依存の特権命令が追加で必要になる場合があります。

すべての特権モードは FS と XS ビットの 1 つのコピーを共有します。
複数の特権モードを持つシステムでは、スーパーバイザモードでは通常、スーパーバイザレベルのセーブされたコンテキストに関してステータスを記録するために FS ビットと XS ビットを直接使用します。
他のより特権のあるアクティブモードは、対応するバージョンのコンテキストで拡張状態を保存して復元する際に、より慎重でなければなりません。

任意の合理的なユースケースでは、ユーザとスーパーバイザレベル間のコンテキストスイッチの数は、他の権限レベルへのコンテキストスイッチの数よりはるかに多いはずです。(上回る必要があります。)
コプロセッサは、割り込みによってユーザレベルのコンテキストスワップが発生しない限り、非同期割り込みを処理するためにコンテキストを保存および復元する必要はないことに注意してください。

3.1.12 マシントラップベクトルベースアドレスレジスタ（mtvec）

mtvec レジスタは、ベクトルベースアドレス（BASE）とベクタモード（MODE）で構成されるトラップベクタの構成を保持する XLEN ビットの読み/書きレジスタです。

XLEN-1	2	1	0
BASE[XLEN-1:2](WARL)		MODE(WARL)	
XLEN-2		2	

図 3.8：マシントラップベクタベースアドレスレジスタ（mtvec）。

mtvec レジスタは常に実装する必要がありますが、ハードワイヤードの読み取り専用値を含めることができます。
mtvec が書き込み可能な場合、レジスタが保持できる値のセットは実装によって異なる可能性があります。
BASE フィールドの値は常に 4 バイトの境界に揃えなければならず、MODE 設定は BASE フィールドの値に追加の配置制約を課す可能性があります。

トラップベクタベースアドレスの実装にはかなりの柔軟性があります。
一方では、多数の状態ビットを持つローエンドの実装に負担をかけることは望ましくありませんが、一方で、より大きいシステムでは可用性を可能にしたいと考えています。(柔軟性を許可することを希望しています。)

値	名前	説明
0	ダイレクト	すべての例外は pc を BASE に設定
1	ベクタード	非同期割り込みは、pc を BASE + 4 *cause に設定
≧2	–	予約

表 3.5：mtvec MODE フィールドのエンコーディング

MODE フィールドのエンコーディングを表 3.5 に示します。
MODE = ダイレクトの場合、マシンモードに入るすべてのトラップが、PC をベースフィールドのアドレスに設定します。
MODE = ベクターの場合、マシンモードへの同期例外はすべて PC をベースフィールドのアドレスに設定しますが、割り込みは PC をベースフィールドのアドレス+割り込み要因番号の 4 倍に設定します。

たとえば、マシンモードタイマ割り込み（表 3.6 を参照）は、PC を BASE + 0x1c に設定します。
MODE =ベクタードを設定すると、ベースに追加のアラインメント制約が課されることがあり、最大４＊ XLEN バイトのアライメントが必要です。

ベクタ割り込みが有効にされると、ユーザモードのソフトウェア割り込みに対応する割り込み要因 0 は、同期例外と同じ場所にベクトル化されます。
この曖昧さは、実際には発生しません。これは、ユーザーモードのソフトウェア割り込みが無効になっているか、権限の低いモードに委任されているためです。

リセットおよび NMI ベクタの場所は、プラットフォーム仕様に記載(指定)されています。

3.1.13 マシントラップ代理レジスタ（medeleg と mideleg）

デフォルトでは、任意の特権レベルのすべてのトラップはマシンモードで処理されますが、マシンモードハンドラはトラップを MRET 命令（3.2.2 項）で適切なレベルにリダイレクトできます。
パフォーマンスを向上させるために、実装は medeleg および mideleg 内の個々の読み取り/書き込みビットを提供して、特定の例外および割り込みを低い特権レベルで直接処理する必要があることを示します。
マシン例外委譲レジスタ（medeleg）およびマシン割り込み委譲レジスタ（mideleg）は、XLEN ビットの読み取り/書き込み・レジスタです。

3 つの特権モード（M / S / U）を持つすべてのシステムでは、medeleg または mideleg にビットを設定すると、S モードまたは U モードの対応するトラップが S モードトラップハンドラに委任されます。
U モードトラップがサポートされている場合、S モードは、sedeleg レジスタと sideleg レジスタに対応するビットを順番に設定して、U モードで発生するトラップを U モードトラップハンドラに委任することができます。

2 つの特権モード（M / U）と U モードトラップをサポートするシステムでは、medeleg または mideleg にビットを設定すると、U モードの対応するトラップが U モードトラップハンドラに委任されます。

M モードのみ、または M モードと U モードの両方を持つが U モードトラップをサポートしていないシステムでは、medeleg と mideleg レジスタは存在してはいけません。(存在しません)

1.9.1 以前のバージョンでは、ここれらのレジスタは存在していましたが、M モードでのみまたは N システムなしの M / U でゼロに固定配線されていました。
これらのケースでは、ミサレジスタが存在するかどうかを示すので、ゼロを返す必要はありません。

トラップが特権の低いモード x に委任されると、x 原因レジスタにトラップ原因が書き込まれます。
x epc レジスタには、トラップを取った命令の仮想アドレスが書き込まれます。
mstatus の xPP フィールドは、トラップ時にアクティブ特権モードで書き込まれます。
mstatus の x PIE フィールドには、トラップ時にアクティブな割り込みイネーブルビットの値が書き込まれます。
mstatus の x IE フィールドがクリアされます。
mcause および mepc レジスタと mstatus の MPP および MPIE フィールドは書き込まれません。

実装は、委譲ビットを 1 つにハードワイヤしてはなりません。すなわち、委任可能なトラップが委任されていないことをサポートしなければなりません。
実装では、デリゲート可能なトラップをサブセット化することを選択できます。サポートされている委託可能なビットは、どのビット位置が 1 を保持しているかを見るために、medeleg または mideleg の値を読み戻します。

トラップは、より特権モードから低特権モードに移行することはありません。
たとえば、M モードが S モードに不正な命令トラップを委譲し、M モードソフトウェアが後で不正命令を実行すると、S モードに委譲されるのではなく、M モードでトラップが取られます。

対照的に、トラップは水平に取られてもよい。(取られることがあります)
同じ例を使用して、M モードが不正な命令トラップを S モードに委譲し、S モードソフトウェアが後で不正命令を実行すると、トラップは S モードで取られます。(行われます)

図 3.9：マシン例外委譲登録 medeleg。

medeleg は、表 3.6 に示すすべての同期例外に対して割り当てられたビット位置を持ち、ビット位置のインデックスは mcause レジスタに返された値と等しくなります。（すなわち、ビット 8 を設定することにより、ユーザモード環境呼出しをより低い特権のトラップハンドラに委任することができます）。

図 3.10: マシン例外の委任登録 mideleg。

mideleg は個々の割り込みのトラップ委譲ビットを保持し、ビットのレイアウトは mip レジスタのものと一致します。
（すなわち、STIP 割り込み委譲制御はビット 5 に位置します）。
いくつかの例外は、特権の低いモードでは発生することができず(発生しませんし)、対応する x edeleg ビットはゼロにハードワイヤードされるべきです。(する必要があります。)
特に、medeleg [11]と sedeleg [11：9]はすべてゼロに固定されています。

3.1.14 マシン割り込みレジスタ（mip および mie）

mip レジスタは、保留中の割り込みに関する情報を含む XLEN ビットの読み/書き・レジスタです。一方、mie は、割り込みイネーブル・ビットを含む対応する XLEN ビットの読み/書き・レジスタです。
この CSR アドレスによって、低特権ソフトウェア割り込み（USIP、SSIP）、タイマ割り込み（UTIP、STIP）、および mip の外部割り込み（UEIP、SEIP）に対応するビットだけが書き込み可能です。
残りのビットは読み取り専用です。

mip レジスタと mie レジスタの制限されたビューは sip / sie として表示され、uip / uie はそれぞれ S モードと U モードで登録されます。
割り込みが mideleg レジスタにビットをセットすることによって特権モード x に委任された場合、x ip レジスタに表示され、x ie レジスタを使用してマスク可能です。
それ以外の場合、x ip と x ie の対応するビットはゼロにハードワイヤードされているように見えます。

XLEN-1	12	11	10	9	8	7	6	5	4	3	2	1	0
	WIRI	MEIP	WIRI	SEIP	UEIP	MTIP	WIRI	STIP	UTIP	MSIP	WIRI	SSIP	USIP
XLEN-12	1	1	1	1	1	1	1	1	1	1	1	1	1

図 3.11：マシン割込み保留レジスタ（mip）。

MTIP ビット、STIP ビット、UTIP ビットは、マシン、スーパバイザ、ユーザタイマ割り込みのタイマ割り込み保留ビットにそれぞれ対応します。
MTIP ビットは読み出し専用で、メモリマップされたマシンモードタイマコンペアレジスタへの書き込みによってクリアされます。
UTIP ビットと STIP ビットは、低レベルの特権レベルにタイマー割込みを送るために M モードソフトウェアによって書き込まれます。
ユーザおよびスーパバイザソフトウェアは、それぞれ AEE および SEE へのコールにより、UTIP および STIP ビットをクリアすることができます。

XLEN-1 12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI	MEIE	WPRI	SEIE	UEIE	MTIE	WPRI	STIE	UTIE	MSIE	WPRI	SSIE	USIE
XLEN-12	1	1	1	1	1	1	1	1	1	1	1	1

図 3.12：マシン割込み許可レジスタ（mie）。

M モード、S モード、U モードの各タイマ割り込みには、それぞれ MTIE、STIE、UTIE という名前の別々のタイマ割り込みイネーブルビットがあります。

各下位特権レベルには、独立した(個別の)ソフトウェア割り込み保留ビット（SSIP、USIP）があります。これは、関連するまたはそれ以上の特権レベルでローカルハートで実行されているコードから CSR アクセスによって読み書きすることができます。
マシンレベルの MSIP ビットは、メモリマップ制御レジスタへのアクセスによって書き込まれ、これはリモートハートがマシンモードのプロセッサ間割り込みを提供するために使用されます。
より低い特権レベルのためのプロセッサ間割り込みは、それぞれ AEE または SEE への ABI および SBI コールを介して実装され、それは最終的に受信側ハートの MSIP ビットへのマシンモード書き込みをもたらす可能性があります。
ハートは、同じメモリマップ制御レジスタを使用して、独自の MSIP ビットを書き込むことができます。

私たちは、適切なモードで動作している場合、ハートが独自の SSIP または USIP ビットを直接書き込むことを許可しますが、他のハートが仮想化され、より高い特権レベルによって予定されている可能性があります。
この理由から、ABI と SBI の呼び出しによってプロセッサ間割り込みが提供されています。
マシンモードのハーツは仮想化されておらず、MSIP ビットを設定することで他のハートに直接割り込みをかけることができます。
通常、プラットフォーム仕様に応じてメモリマップ制御レジスタへのキャッシュされていない I / O 書き込みを使用します。

mip の MEIP フィールドは、マシンモード外部割り込みが保留中であることを示す読み出し専用ビットです。
MEIP は、第 7 章で指定されている標準プラットフォームレベルの割り込みコントローラなど、プラットフォーム固有の割り込みコントローラによって設定およびクリアされます。
mie の MEIE フィールドは、設定時にマシンの外部割り込みを有効(可能)にします。

mip の SEIP フィールドには、単一の読み書きビットが含まれます。
SEIP は、外部割込みが保留中であることを S モードに示すために、M モードソフトウェアによって書かれることがあります。
さらに、プラットフォームレベルの割り込みコントローラは、スーパーバイザレベルの外部割り込みを生成することがあります。
ソフトウェア書き込み可能ビットと外部割り込みコントローラからの信号の論理和は、スーパーバイザへの外部割り込みを生成するために使用されます。
SEIP ビットが CSRRW、CSRRS、または CSRRC 命令で読み取られると、rd 宛先レジスタに返される値には、ソフトウェア書き込み可能ビットと割り込みコントローラからの割り込み信号の論理和が含まれます。
ただし、CSRRS 命令または CSRRC 命令のリード・モディファイ・ライト・シーケンスで使用される値は、ソフトウェア書き込み可能な SEIP ビットのみであり、外部割り込みコントローラからの割り込み値は無視されます。

SEIP フィールドの動作は、上位の特権層が外部の割り込みをきちんと模倣できるように設計されており、実際の外部割り込みを失うことはありません。
その結果、CSR 命令の動作は通常の CSR アクセスから少し変更されています。

MIP の UEIP フィールドは、ユーザモード割り込みのための N 個の拡張が実装されているとき、ユーザモードの外部割り込みを提供します。
SEIP と同様に定義されます。

mie CSR の MEIE、SEIE、および UEIE フィールドは、それぞれ M モード外部割り込み、S モード外部割り込み、および U モード外部割り込みをそれぞれ有効にします。

非マスク可能割り込みは、NMI トラップハンドラを実行するときにその存在が暗黙的に知られている(認識される)ので、mip レジスタを介して見ることはできません。(表示されません)

すべての様々な割り込みタイプ（ソフトウェア、タイマ、外部）に対して、特権レベルがサポートされていない場合、関連する保留中および割り込み可能ビットは、それぞれ mip および mie レジスタでゼロにハードワイヤードされます。
したがって、これらはすべて効果的に WARL フィールドです。

これらのレジスタの上位ビットには、プラットフォーム固有のマシンレベルの割り込みソースを追加することができますが、ほとんどの外部割り込みはプラットフォーム割り込みコントローラを経由してルーティングされ、MEIP 経由で配信されることが期待されます。(配線されます。)

割り込み i は、ビット i が mip と mie の両方に設定されている場合、および割り込みがグローバルにイネーブルされている場合に行われます。
デフォルトでは、ハートの現在の特権モードが M より小さい場合、または現在の特権モードが M で mstatus レジスタの MIE ビットが設定されている場合、M モード割込みはグローバルにイネーブルされます。
しかし、mideleg のビット i がセットされている場合、hart の現在の特権モードが委譲特権モード（S または U）に等しく、そのモードの割込み許可ビット（mstatus の SIE または UIE）が設定されている場合、または現在の特権モードが委譲特権モードよりも小さい場合、割り込みはグローバルに有効とみなされます。

同じ特権レベルの複数の同時割り込みとトラップは、次の優先順位の低い順に処理されます。：外部割込み、ソフトウェア割込み、タイマー割込み、次にすべての同期トラップがあります。

3.1.15 マシンタイマレジスタ（mtime と mtimecmp）

プラットフォームは、メモリマップされたマシンモードレジスタ mtime として公開されるリアルタイムカウンタを提供します。
mtime は一定の頻度(周波数)で実行されなければならず、プラットフォームは mtime のタイムベースを決定するためのメカニズムを提供しなければなりません。(必要があります)

mtime レジスタは、すべての RV32、RV64、および RV128 システムで 64 ビット精度を持ちます。
プラットフォームは、64 ビットのメモリマップされたマシンモードタイマコンペアレジスタ（mtimecmp）を提供し、mtime レジスタに mtimecmp レジスタの値以上の値が含まれている場合に、タイマ割り込みがポストされます。
割り込みは、mtimecmp レジスタを書き込むことによってクリアされるまでポストされたままです。
割り込みは、割り込みが許可され、MTIE ビットが mie レジスタに設定されている場合にのみ行われます。

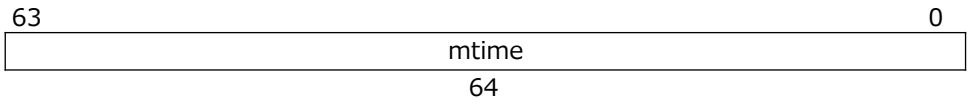


図 3.13：マシンタイムレジスタ（メモリマップ制御レジスタ）。

タイマ機能は、動的な電圧および周波数スケーリングによってエネルギーを節約するために、非常に可変なクロック周波数で動作する最新のプロセッサをサポートするサイクルカウンタではなく、ウォールクロック時間を使用するように定義されています。
↑wall-clock(壁クロック)って RTC とか場合によって周波数(周期)が変わらないクロックのことだと思う。

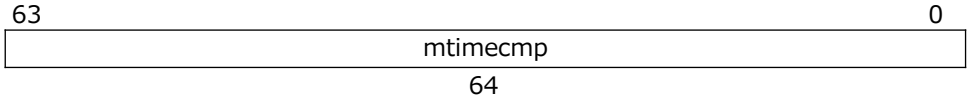


図 3.14：マシン時間比較レジスタ（メモリマップ制御レジスタ）。

正確なリアルタイムクロック（RTC）は、（水晶または MEMS 発振器を必要とする）比較的高価であり、システムの残りの部分がパワーダウンしても実行する必要がありますので、通常、プロセッサとは異なる周波数/電圧領域に位置するシステムには 1 つしか存在しません。
したがって、RTC はシステム内のすべてのハートによって共有されなければならず、RTC へのアクセスは潜在的に電圧レベルシフタとクロックドメイン交差のペナルティを招くことがあります。
したがって、mtime を CSR よりもメモリマップされたレジスタとして公開する方が自然です。
より低い特権レベルには独自の timecmp レジスタはありません。
代わりに、マシンモードソフトウェアは、次のタイマー割り込みを mtimecmp レジスタに多重化することによって、ハート上に任意の数の仮想タイマを実装できます。
シンプルな固定周波数システムでは、サイクルカウントとウォールクロックの両方に単一のクロックを使用できます。

RV32 では、mtimecmp へのメモリマップ書き込みはレジスタの 1 つの 32 ビット部分のみを変更します。
次のコードシーケンスは、comparand の中間値のために、タイマ割り込みを誤って生成することなく、64 ビットの mtimecmp 値を設定します。：

```
# New comparand is in a1:a0.
li    t0, -1
sw t0, mtimecmp      # No smaller than old value.
sw a1, mtimecmp+4 # No smaller than new value.
sw a0, mtimecmp      # New value.
```

図 3.15：レジスタが強く順序付けられた I / O 領域にあると仮定して、RV32 の 64 ビットの時間比較値を設定するためのサンプルコード。

3.1.16 ハードウェアパフォーマンスモニタ

M モードには、基本的なハードウェアパフォーマンス監視機能が含まれています。
mcycle CSR は、過去の任意の時間以来、ハートが実行したサイクル数のカウントを保持します。
minstret CSR には、ハートが退職(引退)した命令の数が過去の任意の時間からカウントされます。
↑retired をどう訳すか。実行がすでに終わったぐらいでいいのかな。
mcycle レジスタと minstret レジスタは、すべての RV32、RV64、および RV128 システムで 64 ビットの精度を持ちます。

ハードウェアパフォーマンスモニタには、29 個の追加イベントカウンタ mhpmcounter3-mhpmcounter31 が含まれています。
イベントセレクタ CSRs mhpmevent3-mhpmevent31 は、どのイベントが対応するカウンタを増加させるかを制御する WARL レジスタです。
これらのイベントの意味はプラットフォームによって定義されますが、イベント 0 は「イベントなし」を意味するために予約されています。 すべてのカウンタを実装する必要がありますが、カウンタと対応するイベントセレクタの両方を 0 にハードワイヤリングするのが法的な実装です。
↑leagal をどう訳すか。法的というより、正当なとか適切な。とかかな。

これらのカウンタはすべて、RV32、RV64、および RV128 で 64 ビットの精度を持ちます。

RV32 のみでは、mcycle、minstret、および mhpmcountern の CSR の読み取りによって 32 ビットが返され、一方、
mcycleh、minstreth、および mhpmcounternh CSR の読み取りは、対応するカウンタのビット 63-32 を返します。

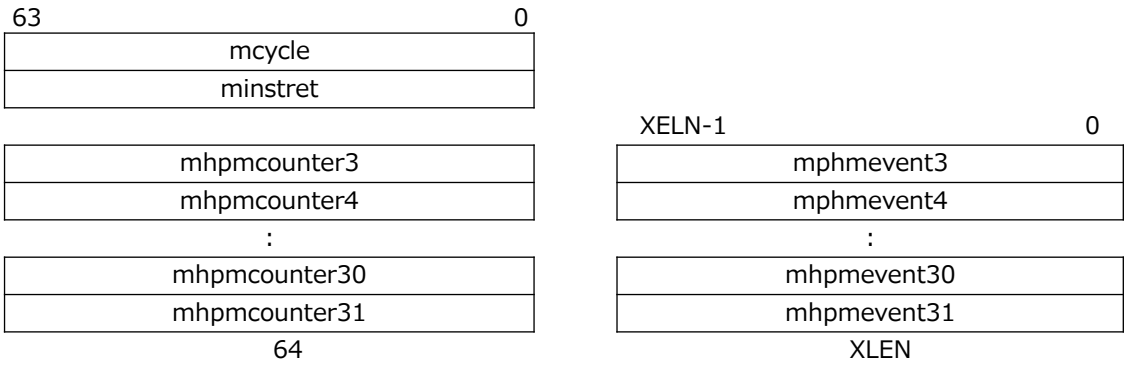


図 3.16：ハードウェアパフォーマンスモニタカウンタ。

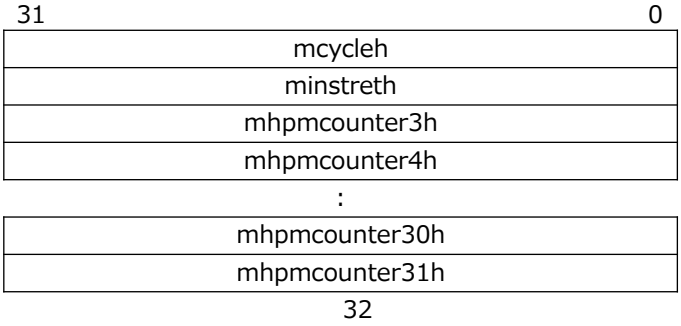


図 3.17：上位 32 ビットのハードウェアパフォーマンスモニタカウンタ、RV32 のみ。

RV128 システムでは、mcycle、minstret、および mhpmcountern の 64 ビット値は、読み込み時に 128 ビットに符号拡張されます。

RV128 システムでは、符号付き 64 ビット値と符号なし 64 ビット値の両方が正準形式で保持され、ビット 63 はすべての上位ビット位置で繰り返されます。
↑63bit 目が符号拡張されるということ
カウンタは RV128 でも 64 ビットの値であるため、カウンタ CSR は符号拡張拡張不変量を保持して読み込みます。
実装は、カウンタのより少ないビットを実装することを選択することができ、（例えば、263 命令が実行される）ラップアラウンドを経験する可能性は低く、それによって符号拡張回路を実際に実装する必要がなくなる。
↑この例では 263 まで数えられれば良いので 9bit512 まで数えられるカウンタで OK

3.1.17 カウンタイネーブルレジスタ（[m | h | s] counteren）

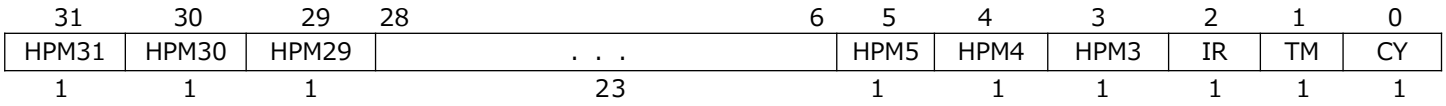


図 3.18：カウンタイネーブルレジスタ（mcounteren と scounteren）。

カウンタイネーブルレジスタ mcounteren および scounteren は、ハードウェアパフォーマンスモニタリングカウンタの可用性を、次に低い特権モードに制御します。

mcounteren レジスタの CY、TM、IR、または HPMn ビットがクリアされている場合、S モードまたは U モードで実行中に、cycle、time、instret、または hpmcountern レジスタを読み込もうとすると、不正な命令例外が発生します。

これらのビットの 1 つがセットされると、次の実装された特権モード（実装されている場合は S モード、それ以外の場合は U モード）で対応するレジスタへのアクセスが許可されます。

S モードが実装されている場合、U モードで実行している間は、scounteren レジスタの同じビット位置がこれらのレジスタへのアクセスを同様に制御します。
S モードがカウンタ・レジスタへのアクセスを許可され、対応するビットが scounteren に設定されている場合、U モードもそのレジスタにアクセスすることができます。

レジスタ mcounteren と scounteren は、U モードと S モードが実装されている場合に実装する必要がある WARL レジスタです。
いずれのビットにもハードワイヤードの値 0 が含まれている可能性があります。これは、対応するカウンタへの読み込みによって、特権レベルの低いモードで実行されているときに例外が発生することを示します。

カウンタイネーブルビットは、最小限のハードウェアで 2 つの一般的な使用例をサポートします。
高性能タイマとカウンタが不要なシステムの場合、マシンモードソフトウェアはアクセスをトラップし、ソフトウェアのすべての機能を実装できます。
高性能タイマとカウンタを必要とするが、基盤となるハードウェアカウンタの難読化には関係(考慮)しないシステムでは、カウンタを低特権モードに直接さらす(公開する)ことができます。

cycle、instret、および hpmcountern の CSRs は、それぞれ mcycle、minstret、および mhpmcountern の読み取り専用のシャドウです。
time CSR は、メモリマップされた mtime レジスタの読み取り専用の影(シャドウ)です。

実装は、time CSR の読み込みをメモリマップされた mtime レジスタへのロードに変換するか、または mxcounteren の TM ビットを 0 にハードワイヤリングし、M-mode ソフトウェアでこの機能をエミュレートすることができます。

3.1.18 マシンスクラッチレジスタ（mscratch）

mscratch レジスタは、マシンモードで使用するために専用の XLEN ビットのリード/ライトレジスタです。
通常は、マシンモードハートローカルコンテキスト空間へのポインタを保持するために使用され、M モードトラップハンドラへのエントリ時にユーザレジスタとスワップされます。

図 3.19：マシンモードスクラッチレジスタ。

MIPS ISA は、オペレーティングシステムで使用するために 2 つのユーザーレジスタ（k0 / k1）を割り当てました。
MIPS スキームは、迅速かつ簡単な実装を提供しますが、使用可能なユーザー・レジスタを減らし、さらに特権レベルやネストされたトラップに拡張されません。
潜在的なセキュリティホールを回避し、確定的なデバッグ動作を提供するために、ユーザーレベルに戻る前に両方のレジスタをクリアする必要があります。

RISC-V ユーザー ISA は、多くの可能な特権システム環境をサポートするように設計されているため、ユーザーレベルの ISA には OS 依存の機能を感染させたくありませんでした。
RISC-V CSR スワップ命令は、値を mscratch レジスタにすばやく保存/復元できます。
MIPS デザインとは異なり、OS はユーザコンテキストの実行中に mscratch レジスタに値を保持することができます。

3.1.19 マシン例外プログラムカウンタ（mepc）

mepc は、図 3.20 のように書式設定された XLEN ビットの読取り/書込みレジスタです。
mepc（mepc [0]）の下位ビットは常にゼロです。
16 ビット命令アライメントを持つ命令セット拡張をサポートしない実装では、2 つの下位ビット（mepc [1：0]）は常にゼロです。

---

mepc レジスタは、命令アドレス不一致の例外を発生させる PC 値を保持することはできません。

mepc は、有効な物理アドレスと仮想アドレスをすべて保持できる必要がある WARL レジスタです。
すべての可能な無効アドレスを保持できる必要はありません。
実装は、無効なアドレスパターンを mepc に書き込む前に他の無効なアドレスに変換する可能性があります。

トラップが M モードになると、mepc には例外が発生した命令の仮想アドレスが書き込まれます。
それ以外の場合、mepc は実装によって書き込まれることはありませんが、ソフトウェアによって明示的に記述されることがあります。

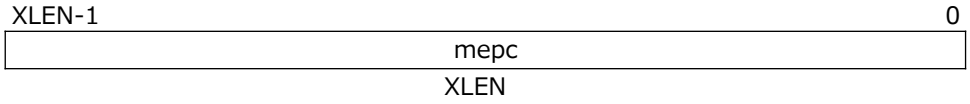


図 3.20：マシン例外プログラムカウンタレジスタ。

3.1.20 マシン要因レジスタ（mcause）

mcause レジスタは、図 3.21 に示すようにフォーマットされた XLEN ビットの読み書き可能レジスタです。
トラップを M モードにすると、トラップの原因となったイベントを示すコードが mcause に書き込まれます。
それ以外の場合、mcause は実装によって書き込まれませんが、ソフトウェアによって明示的に書き込まれることがあります。

トラップが割り込みによって発生した場合、mcause レジスタの割り込みビットがセットされます。
例外コードフィールドには、最後の例外を識別するコードが含まれています。
表 3.6 に、可能なマシンレベルの例外コードを示します。
例外コードは WLRL フィールドなので、サポートされている例外コードを保持することのみが保証されています。

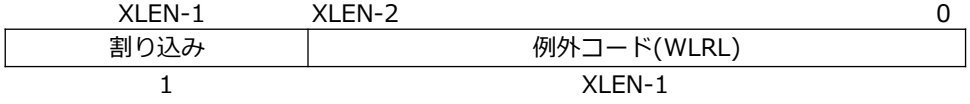


図 3.21：マシン要因レジスタ mcause。

特権命令例外と不正命令コード例外を区別しません。
これにより、アーキテクチャが簡素化され、インプリメンテーションによってどのより高い特権の命令がサポートされているかが詳細に隠されます。(非表示になります。)
トラップをサービスする特権レベルは、これらを区別する必要があるかどうか、もしそうであれば、与えられたオペコードを違法(不当)または特権として扱うべきかどうかに関するポリシーを実装できます。

割り込みは、mcause レジスタ値の符号で単一の分岐を持つ他のトラップと分離することができます。
左にシフトすると、割込みビットを除去し、例外コードをスケーリングしてトラップベクタテーブルにインデックスすることができます。

割り込み	例外コード	説明
1	0	ユーザーソフトウェア割り込み
1	1	監督者ソフトウェア割り込み
1	2	予約
1	3	マシンソフトウェア割り込み
1	4	ユーザータイマ割り込み
1	5	監督者タイマ割り込み
1	6	予約
1	7	マシンタイマ割り込み
1	8	ユーザー外部割込み
1	9	監督者外部割込み
1	10	予約
1	11	マシン外部割込み
1	≧12	予約
0	0	命令アドレス不整列
0	1	命令アクセス障害
0	2	違法(違反)命令
0	3	ブレークポイント
0	4	読み込みアドレス不整列
0	5	読み込みアクセス障害
0	6	保存/AMO アドレス不整列
0	7	保存/AMO アドレス障害
0	8	U モードからの環境呼び出し
0	9	S モードからの環境呼び出し
0	10	予約
0	11	M モードからの環境呼び出し
0	12	命令ページ障害
0	13	読み込みページ障害
0	14	予約
0	15	保存/AMO ページ障害
0	≧16	予約

表 3.6 トラップ後のマシン要因レジスタ（mcause）の値

3.1.21 マシントラップ値（mtval）レジスタ

mtval レジスタは、図 3.22 に示すようにフォーマットされた XLEN ビットの読み書きレジスタです。
トラップを M モードにすると、mtval に例外固有の情報が書き込まれ、ソフトウェアがトラップを処理する際に役立ちます。
それ以外の場合、mtval は実装によって書き込まれることはありませんが、ソフトウェアによって明示的に書き込まれることがあります。

ハードウェアブレークポイントがトリガされるか、命令フェッチ、読み込み、または保存アドレスの不整列、アクセス、またはページフォルトの例外が発生すると、mtval にはフォルトのある実効アドレスが書き込まれます。
不正な命令トラップでは、mtval はフォールト中の命令の最初の XLEN ビットで記述されます。
その他の例外については、mtval はゼロに設定されますが、将来の標準では、他の例外に対して mtval の設定を再定義すること(可能性)があります。

---

mtval レジスタは、前の仕様の mbadaddr レジスタを置き換えます。
不正なアドレスを提供することに加えて、レジスタは不正な命令トラップをトリガした（そして将来他の情報を返すために使用されるかもしれない）不良命令を提供できるようになりました。
命令ビットを戻す(返す)と命令エミュレーションが加速され、不正命令をエミュレートしようとするときに存在する可能性のある競合も取り除かれます。

ページベースの仮想メモリが有効な場合、物理メモリアクセスの例外の場合でも、mtval にはフォールト仮想アドレスが書き込まれます。
この設計は、ほとんどの実装、特にハードウェアページテーブルウォーカーを持つ実装のデータパスコストを削減します。

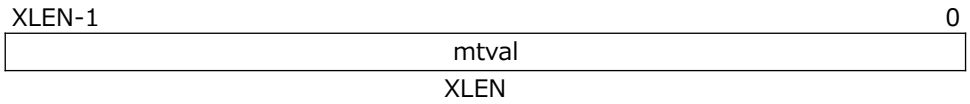


図 3.22：マシントラップ値レジスタ。

可変長命令を持つ RISC-V システム上の命令フェッチアクセス障害の場合、mtval は、命令の先頭を指し示す間に、障害の原因となった命令の部分へのポインタを含みます。

オプションで、mtval レジスタを使用して、不正命令例外（mepc がメモリ内のフォルト命令を指し示す）でフォルト命令ビットを返すこともできます。

この機能が提供されていない場合、mtval は不正な命令フォルトでゼロに設定されます。

この機能が提供されている場合、不正な命令トラップの後で、命令が XLEN ビットを超えない限り、mtval には障害発生命令全体が含まれます。
命令が XLEN ビットより短い場合、mtval の上位ビットはゼロにクリアされます。
命令の長さが XLEN ビット以上の場合、mtval には命令の最初の XLEN ビットが格納されます。

---

mtval でフォールト中の命令をキャプチャすると、命令エミュレーションのオーバヘッドが減少し、命令が不整列された場合にいくつかの部分的な命令のロードが回避される可能性があり、そして、読み込みがデータレジスタに命令をフェッチするために使用されたときにデータキャッシュミスまたは低速のキャッシュされていないアクセスが発生する可能性があります。
動的翻訳システムで発生する可能性があるように、他のエージェントが命令メモリを操作している場合は、原子性の問題もあります。

必要なのは、命令全体（または少なくとも最初の XLEN ビット）がトラップを取る前に mtval にフェッチされることです。
これは実装を制限するものではなく、命令のデコードを試みる前に命令全体をフェッチし、ソフトウェアハンドラの複雑さを避けることができます。

mtval の値が 0 の場合、その機能がサポートされていないか、不正なゼロ命令がフェッチされたことが示されます。
mepc が指す命令メモリからの読み込みを使用して、これらの 2 つのケースを区別することができます（または、実行時に適切なトラップ処理をインストールするためにシステム設定情報を調べることができます）。

mtval は、有効なすべての物理アドレスと仮想アドレスと値 0 を保持できる必要がある WARL レジスタです。
可能なすべての無効なアドレスを保持できる必要はありません。
実装は、無効なアドレスパターンを mtval に書き込む前に他の無効なアドレスに変換する可能性があります。
障害のある命令ビットを返す機能が実装されている場合、mtval は 2N 未満のすべての値も保持できる必要があり、N は XLEN のうち小さい方とサポートされる最長命令の幅です。

– 2018/11/25

3.2 マシンモード特権命令

3.2.1 環境呼び出しとブレークポイント

31 20	19 15	14 12	11 7	6 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
ECALL	0	PRIV	0	SYSTEM
EBREAK	0	PRIV	0	SYSTEM

ECALL 命令は、支援実行環境への要求を行うために使用されます。
U モード、S モード、または M モードで実行されると、U モードからの環境呼び出し例外、S モードからの環境呼び出し例外、または M モードからの環境呼び出しを生成します。 他の操作を実行しません。

ECALL は、元の特権モードごとに異なる例外を生成し、環境呼び出しの例外を選択的に委任することができます。
Unix ライクなオペレーティングシステムの典型的な使用例は、U モード例外からの環境呼び出しを S モードに委譲することですが、他のものではありません。

デバッガで EBREAK 命令を使用して、制御をデバッグ環境に戻します。
ブレークポイント例外が生成され、他の操作は実行されません。

このマニュアルの第 1 巻の「C」圧縮命令の標準拡張に記載されているように、C.EBREAK 命令は EBREAK 命令と同じ動作を実行します。

ECALL および EBREAK は、受信特権モードの epc レジスタに、次の命令のアドレスではなく、ECALL または EBREAK 命令のアドレスに設定されます。

3.2.2 トラップリターン命令

トラップから戻る命令は、PRIV マイナーオペコードの下にエンコードされます。

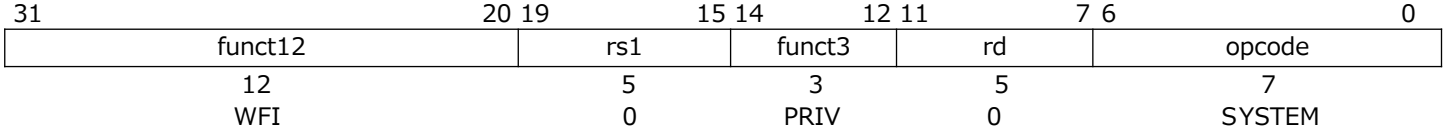
31 20	19 15	14 12	11 7	6 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
MRET/SRET/URET	0	PRIV	0	SYSTEM

トラップを処理した後に戻るには、特権レベルごとに個別のトラップリターン命令があります。 : MRET、SRET、および URET。
MRET は常に提供され、スーパーバイザモードがサポートされている場合は SRET を提供する必要があります。
URET は、ユーザーモードのトラップがサポートされている場合にのみ提供されます。
xRET 命令は、特権モード x 以上で実行することができます。下位(低)特権 xRET 命令を実行すると、関連する下位(低)特権割込みの有効化および特権モードスタックがポップされます。
セクション 3.1.7 で説明した特権スタックの操作に加えて、xRET は pc を x epc レジスタに格納された値に設定します。

以前は、ERET 命令が 1 つしかありませんでした（以前は SRET とも呼ばれていました）。
ユーザーレベルの割り込みの追加をサポートするために、ERET 命令を使用して OS コードの古典的な仮想化を引き続き許可するために、別々の URET 命令を追加する必要がありました。
次に、特権レベルごとに異なる xRET 命令をサポートするために、より直交的になりました。

3.2.3 割り込み待ち

Wait for Interrupt 命令（WFI）は、割り込みがサービスを必要とするまで、現在のハートが停止する可能性があるというヒントを実装に提供します。
WFI 命令の実行は、適切な割込みがこのハートに優先的にルーティングされるべきであることをハードウェアプラットフォームに知らせるためにも使用することができます。
WFI はサポートされているすべての S および M 特権モードで利用できますが、
オプションで U モード割り込みをサポートする実装のために U モードで使用できます。



31 20	19 15	14 12	11 7	6 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
WFI	0	PRIV	0	SYSTEM

ハートが停止している間に、有効な割り込みが存在するか、または後で存在する場合、割り込み例外は次の命令で取られます。
すなわち、トラップハンドラおよび mepc = pc + 4 で実行が再開されます。

次の命令は割り込み例外とトラップを受け取り、トラップハンドラからの単純な戻りは WFI 命令の後にコードを実行します。

WFI 命令はほんのヒントであり、法的な実装は WFI を NOP として実装することです。

実装が命令の実行時にハートを停止させない場合、割り込みは WFI を含むアイドルループ内の何らかの命令で取られ(ループの命令に対して行われ)、ハンドラからの単純な復帰時には、アイドルループは実行を再開します。

我々は実装が rs1 と rd フィールドを無視するという以前の要件を削除しましたが、これらのフィールドのゼロ以外の値が不正な命令例外を発生させるはずです。(0 以外の値は、不正な命令例外を発生させる必要があります。)

WFI 命令は、割り込みが禁止されているときに(ときにも)実行することもできます。
WFI の動作は、mstatus（MIE / SIE / UIE）および委譲レジスタ[m | s | u] ideleg のグローバル割り込みビットの影響を受けないようにする必要があります
（すなわち、局所的に使用可能な割込みが保留になった場合にはあまり特権のないモードに委任されたとしてもハートを再開しなければなりません。）、しかし個々の割り込みイネーブル（例えば、MTIE）を尊重するべきです。
（すなわち、割り込みが保留されているが個々に有効にされていない場合、実装はハートを再開しないでください）。
また、WFI は、各特権レベルでのグローバル割り込み有効化に関係なく、任意の特権レベルで保留中のローカルで有効な割り込みの実行を再開する必要があります。

ハートの実行を再開させるイベントによって割り込みが発生しない場合は、PC + 4 で実行が再開され、アクション可能なイベントがない場合に WFI を繰り返すためのループバックを含む、ソフトウェアが取るべきアクションを決定する必要があります。

割り込みが無効のときにウェイクアップを許可することにより、WFI が実行される前に現在のコンテキストを保存または破棄できるため、現在のコンテキストを保存する必要がない割り込みハンドラへの代替エントリポイントを呼び出すことができます。

実装は WFI を NOP として自由に実装できるため、ソフトウェアは、WFI に続くコード内の関連する保留中の割り込みを無効にする必要があるかどうかを明示的にチェックする必要があります。そして、適切な割り込みが検出されなかった場合は WFI にループバックする必要があります。
MIP、SIP、または UIP レジスタは、任意のそれぞれのマシン、スーパーバイザ、またはユーザ・モードの割り込みの存在を決定するために調べることができます。

WFI の操作は、委任レジスタ設定の(によって)影響を受けません。

WFI は、実装がより高い特権モードにトラップできるように定義されており、WFI に直面(発生)した直後か或る(一定の)時間間隔の後に、例えば低電力状態への機械モード移行を開始します。

同じ「イベント待機」テンプレートは、メモリの場所(ロケーション)の変更またはメッセージ到着を待つ可能性のある将来の拡張機能に使用される可能性があります。

-- 2018/12/02

3.3 リセット

リセットすると、ハートの特権モードが M に設定されます。
mstatus フィールド MIE および MPRV は 0 にリセットされます。
pc は、実装定義のリセットベクタに設定されます。
mcause レジスタは、リセットの原因を示す値に設定されます。
他のハートの状態はすべて定義されていません。

リセット後の mcause 値は実装固有の解釈を持っていますが、異なるリセット条件を区別しない実装では値 0 を返す必要があります。
異なるリセット条件を区別する実装では、最も完全なリセット（ハードリセットなど）を示すために 0 を使用する必要があります。

一部のデザインでは、リセットの原因が複数ある場合があります。（例えば、パワーオンリセット、外部ハードリセット、ブラウンアウト検出、ウォッチドッグタイマ経過、スリープモードウェイクアップ）どのマシンモードソフトウェアとデバッガが区別したいかを知ることができます。
mcause リセット値は、同期例外に続いて mcause 値に別名を付けることがあります(指定できます)。
こリセット時に pc が他のトラップと異なる値に設定されるため、このオーバーラップにあいまい性はありません。

3.4 マスカブルでない割り込み

ノンマスカブル割り込み（NMI）は、ハードウェアエラー条件でのみ使用され、ハートの割り込みイネーブルビットの状態に関係なく、M モードで動作する実装定義の NMI ベクタに即座にジャンプします。
mepc レジスタには、NMI が取られた時点で実行される次の命令のアドレスが書き込まれ、mcause には NMI の元を示す値が設定されます。
したがって、NMI は、アクティブなマシンモード割り込みハンドラで状態を上書きすることができます。

NMI 上で mcause に書き込まれる値は実装で定義されていますが、0 の値は「不明な原因」を意味するために予約されており、
mcause レジスタを介して NMI のソースを区別しない実装は 0 を返すべきです(返す必要があります)。

リセットとは異なり、NMI はプロセッサの状態をリセットせず、診断、レポート、およびハードウェアエラーの包含を可能にします。

3.5 物理メモリの属性

完全なシステムの物理メモリマップには、さまざまなアドレス範囲を含み、いくつかはメモリ領域に対応し、いくつかはメモリマップ制御レジスタに、いくつかはアドレス空間の空の穴になります。

一部のメモリ領域では、読み込み、書き込み、または実行をサポートしない場合があります。
サブワードアクセスやサブブロックアクセスをサポートしていないものもあります。
原子(アトミックな)操作をサポートしていないものもあります。
キャッシュ一貫性(コヒーレンス)をサポートしていないか、メモリモデルが異なる可能性があります。
同様に、メモリマップされた制御レジスタは、サポートされているアクセス幅、アトミック操作のサポート、および読み書きアクセスに関連する副作用があるかどうかによって異なります。
RISC-V システムでは、マシンの物理アドレス空間の各領域のこれらのプロパティと機能を物理メモリ属性（PMAs）と呼びます。
このセクションでは、RISC-V PMA の用語と、RISC-V システムが PMA を実装して確認する方法について説明します。

PMA は、基本となるハードウェアの固有の特性であり、システム動作中にはほとんど変更されません。
セクション 3.6 で説明されている物理メモリ保護値とは異なり、PMAs は実行コンテキストによって変化しません。
いくつかのメモリ領域の PMAs は、例えばオンチップ ROM 用のチップ設計時に固定されます。
他は、例えば、他のチップがオフチップバスに接続されているかどうかに応じて、基板設計時に固定されます。
オフチップ・バスは、電源サイクル（コールド・プラグ可能）またはシステム稼動中（ホット・プラガブル）に動的に変更できるデバイスもサポートします(サポートする場合もあります)。
いくつかのデバイスは、異なる PMAs を暗示(意味)する異なる用途をサポートするために、実行時に構成可能であり(可能な場合があります)、オンチップのスクラッチパッド RAM は、1 つのエンドアプリケーション内の 1 つのコアによってプライベートにキャッシュされるか、別のエンドアプリケーション内の共有された非キャッシュメモリとしてアクセスされる可能性があります。

ほとんどのシステムでは、少なくとも一部の PMAs が物理アドレスが判明した後、実行パイプラインの後のハードウェアで動的にチェックされることが必要です。一部の操作はすべての物理メモリアドレスでサポートされないため、いくつかの操作では、設定可能な PMA 属性の現在の設定を知る必要があります。
他の多くのシステムでは、仮想メモリページテーブルに PMA をいくつか指定しており、TLB を使用してこれらのプロパティをパイプラインに通知していますが、このアプローチでは、プラットフォーム固有の情報を仮想化されたレイヤーに注入し、各物理メモリ領域の各ページテーブルエントリで属性が正しく初期化されない限り、システムエラーを引き起こす可能性があります。
さらに、使用可能なページサイズが物理メモリ空間の属性を指定するのに最適ではないため(ない場合があり)、アドレス空間の断片化や高価な TLB エントリの非効率的な使用が発生する可能性があります。

RISC-V では、PMAs の仕様とチェックを別のハードウェア構造、PMA チェッカーに分けています。
多くの場合、属性は各物理アドレス領域のシステム設計時に認識され、PMA チェッカーにハードワイヤード接続することができます。
属性が実行可能である場合、プラットフォーム固有のメモリマップ制御レジスタを提供して、プラットフォーム上の各領域に適切な粒度でこれらの属性を指定することができます。
（例えば、キャッシュ可能なメモリとキャッシュ不可能なメモリとの間で柔軟に分割できるオンチップ SRAM の場合）。
PMAs は、仮想メモリから物理メモリへの変換を経たアクセスを含む、物理メモリへのアクセスについてチェックされます。
システムのデバッグを助けるため、可能であれば、RISC-V プロセッサは、PMA チェックに失敗した物理メモリアクセスを正確にトラップすることを強く推奨します。
正確にトラップされた PMA 違反は、仮想メモリのページフォルト例外とは異なり、ロード、ストア、または命令フェッチアクセス例外として現れます。
たとえば、検出メカニズムの一部としてアクセス障害を使用する従来のバスアーキテクチャを調査する場合など、正確な PMA トラップは常に可能ではない可能性があります。
この場合、スレーブデバイスからのエラー応答は不正確なバスエラー割り込みとして報告されます。

PMAs は、特定のデバイスに正しくアクセスするため、または DMA エンジンなど、メモリにアクセスする他のハードウェアコンポーネントを正しく構成するために、ソフトウェアで読み取る必要があります。
PMAs は特定の物理プラットフォームの組織に緊密に結びついているため、多くの詳細は本質的にプラットフォーム固有であり、ソフトウェアがプラットフォームの PMA 値を知ることができる手段です。
コンフィギュレーションストリング(構成文字列)（第 8 章）は、オンチップデバイス用の PMAs をエンコードすることができ、また、接続されたデバイス PMAs を検出するために動的に質問することができるオフチップバス用のオンチップコントローラを記述することもできます。

いくつかのデバイス、特にレガシーバスは、PMAs の検出をサポートしていないため、サポートされていないアクセスが試行された場合にエラー応答やタイムアウトが発生します。
通常、プラットフォーム固有のマシンモードコードは PMAs を抽出し、最終的にこの情報をいくつかの標準的な表現を使用して高レベル(上位レベル)で低特権のソフトウェアに提示します。

プラットフォームが PMAs の動的再構成をサポートする場合、プラットフォームを正しく再構成できるマシン・モード・ドライバーに要求を渡すことによって、属性を設定するためのインターフェースが提供されます。
たとえば、一部のメモリー領域でキャッシュ機能を切り替えるには、マシン・モードでのみ使用可能な、キャッシュ・フラッシュなどのプラットフォーム固有の操作が必要になることがあります。

--2018/12/07

3.5.1 メインメモリ対 I / O 対空き領域

与えられた(指定された)メモリアドレス範囲の最も重要な特徴は、それが通常のメインメモリを保持するか、または I / O デバイスであるか、または空であるかどうかです。
通常のメインメモリには、以下で指定する多数のプロパティが必要ですが、I / O デバイスの属性範囲は非常に広いです(広範囲の属性を持つことができます)。
デバイスのスクラッチパッド RAM など、通常のメインメモリに収まらないメモリ領域は、I / O 領域として分類されます。
空の領域も I / O 領域として分類されますが、アクセスがサポートされていないことを指定する属性があります。

3.5.2 サポートされているアクセスタイプ PMA

アクセスタイプは、8 ビットバイトから長いマルチワードバーストまでのどのアクセス幅がサポートされているか、また各アクセス幅に対して誤整列アクセスがサポートされているかどうかを指定します。

RISC-V ハート上で動作するソフトウェアは、メモリへのバーストを直接生成することはできませんが、ソフトウェアは I / O デバイスにアクセスするために DMA エンジンをプログラムする必要があり、したがって、どのアクセスサイズがサポートされているかを知る必要があります。

メインメモリ領域は、接続されたデバイスが必要とするすべてのアクセス幅の読み取り、書き込み、および実行を常にサポートします。

場合によっては、メインメモリにアクセスするプロセッサまたはデバイスの設計が他の幅をサポートすることもありますが、メインメモリがサポートするタイプで機能できる必要があります。

I / O 領域は、データ幅がサポートされている読み取り、書き込み、または実行アクセスの組み合わせを指定できます。

3.5.3 原子性(アトミシティ)PMA

原子性 PMA は、どのアトミック命令がこのアドレス領域でサポートされているかを記述します。
メインメモリ領域は、取り付けられているプロセッサが必要とするアトミック操作をサポートしなければなりません。(必要があります。)
I / O 領域は、プロセッサがサポートするアトミック操作のサブセットのみをサポートすることも、サポートしないこともあります。

アトミック命令のサポートは、LR / SC と AMOs の 2 つのカテゴリに分かれています。
AMO 内には、AMONone、AMOSwap、AMOLogical、AMOArithmetic という 4 つのサポートレベルがあります。
AMONone は、AMO 操作がサポートされていないことを示します。

AMOSwap は、このアドレス範囲でサポートされているのは amoswap 命令だけであることを示します。
AMOLogical は、スワップ命令とすべての論理 AMOs（amoand、amoor、amoxor）がサポートされていることを示します。
AMOArithmetic は、すべての RISC-V AMOs がサポートされていることを示します。
各レベルのサポートについては、下にあるメモリ領域がその幅の読み書きをサポートする場合、特定の幅の自然に位置合わせされた AMOs がサポートされます。

AMO クラス	サポートされる動作
AMONone AMO 無し	無し
AMOSwap AMO 交換	amoswap
AMOLogical AMO 論理	上記 + amoand, amoor, amoxor
AMOArithmetic AMO 算術	上記 + amoadd, amomin, amomax, amominu, amomaxu

表 3.7 : I / O 領域でサポートされる AMO のクラス
メインメモリ領域は、プロセッサが必要とするすべての AMO を常にサポートする必要があります。

可能であれば、少なくとも I / O 領域の AMOLogical サポートを提供することをお勧めします。
ほとんどの I / O 領域は、LR / SC アクセスをサポートしませんが、これは、キャッシュコヒーレンス方式の上に構築するのが最も便利なためです。

3.5.4 メモリ順序(オーダー)PMA

アドレス空間の領域は、フェンス命令とアトミック命令の順序付けビットによる順序付けのためにメインメモリまたは I / O として分類されます。

1 つのハートによる主メモリ領域へのアクセスは、他のハーツだけでなく、主メモリシステム（例えば、DMA エンジン）で要求を開始する能力を有する他の装置によっても観察可能です。
メインメモリ領域は、常に標準の RISC-V 緩和メモリモデルを備えています。

1 つのハートによる I / O 空間へのアクセスは、他のハーツやバスマスタリングデバイスだけでなく、対象のスレーブ I / O デバイスによっても観測されます。
I / O 内では、領域はさらに、緩やかな順序または強い順序のいずれかを実装するものとして分類することができる(してもよい)。
リラックスした I / O 領域には、フェンスと AMO 命令で強制されたものを超えて(以外の)、異なるハートや I / O デバイスが 1 つのハートでのメモリアクセスをどのように観測するかについて、順序保証はありません。
強く注文(厳密に順序づけ)された I / O 領域は、ハートがその領域に行ったすべてのアクセスが、他のすべてのハーツまたは I / O デバイスによってプログラム順にしか観察できないことを保証します。

強く順序付けられた各 I/O 領域は、番号付き順序付けチャネルを指定します。これは、異なる I / O 領域間で順序保証が提供されるメカニズムです。
チャネル 0 は、ポイントツーポイントの強力な順序付けのみを示すために使用され、単一の関連する I / O 領域へのアクセスのみが強く順序づけされます。

チャネル 1 は、すべての I / O 領域にわたってグローバルな強力な順序付けを提供するために使用されます。
ハートによるチャネル 1 に関連する任意の I / O 領域へのアクセスは、他のすべてのハートおよび I / O デバイスによってプログラム順序で発生したことのみが観察され、そのハートがリラックスした I / O 領域または異なるチャネル番号を有する強く順序付けされた I / O 領域に対して行ったアクセスに関連するものを含みます。
換言すれば、チャネル 1 の領域へのアクセスは、命令の前後にフェンス(fence) io、io 命令を実行することと等価です。

他のより大きいチャンネル番号は、同じチャンネル番号を有する任意の地域にわたってそのハートによってアクセスするための番組注文を提供する。

システムは、各メモリ領域上の注文プロパティの動的構成をサポートすることができる。

厳密(強力)な順序付け(並べ替え)を使用すると、従来のデバイスドライバコードとの互換性を向上させることができたり、または実装がアクセスの順序を変更しないことがわかっている場合に明示的な順序付け命令の挿入と比較してパフォーマンスを向上させることができます。
ハートとＩ/ Ｏデバイスの間に 1 つのインオーダー通信パスしかない場合は、ローカルの強力な順序付け（チャネル 0）が強力な順序付けのデフォルト形式です。
一般的に、同一の相互接続パスを共有し、パスが要求を並べ替えることができない場合、厳密(強く)順序付けされた異なるＩ/ Ｏ領域は順序付けハードウェアを追加すること無く同じ順序付けチャネルを共有することができます。

3.5.5 一貫性とキャッシュ可能性 PMA

一貫性は、単一の物理アドレスに対して定義されたプロパティであり、ある(1 つの)エージェントによるそのアドレスへの書き込みが、最終的にシステム内の他のエージェントから見えるようになる(する)ことを示します。
一貫性はシステムのメモリ一貫性モデルと混同されるべきではなく、それは、メモリシステム全体への読取りおよび書込みの以前の履歴が与えられたとき(履歴を元に)に、メモリ読取り値が返すことができる値を定義します。
RISC-V プラットフォームでは、ソフトウェアの複雑さ、パフォーマンス、およびエネルギーの影響により、ハードウェア非干渉性領域の使用は推奨されません。

メインメモリとＩ/ Ｏの分類、メモリの順序付け、サポートされているアクセスとアトミック操作、一貫性など、他の PMAs に反映されている違いを除いて、メモリ領域のキャッシャビリティは領域のソフトウェアビューに影響しないはずです(影響を与えるべきではありません)。
このために、私たちは、キャッシュ可能性を、マシンモードソフトウェアのみで管理されるプラットフォームレベルの設定として扱います。

プラットフォームがメモリ領域に対する構成可能なキャッシュ可能性設定(構成可能な)をサポートする場合、プラットフォーム固有のマシンモードルーチンは、必要に応じて設定を変更し、キャッシュをフラッシュし、システムはキャッシュ可能性設定間の遷移中にインコヒーレントなものに過ぎません(首尾一貫しないだけです。)。
この一時的な状態は、より低い特権レベルでは見えてはなりません(表示されません)。

RISC-V キャッシュは、次の 3 つのタイプに分類されます。：マスター、プライベート、共有、およびスレーブプライベート。
マスター/プライベート・キャッシュは単一のマスター・エージェント(すなわち、メモリシステムに読み出し/書き込み要求を出すもの)に接続されています。
共有キャッシュはマスターとスレーブの間に配置され、階層的に構成されています。
スレーブプライベートキャッシュは、単一のスレーブに対してローカルであり、マスタの他の PMAs に影響を与えないため、一貫性に影響を与えません。したがって、ここでは考慮しません(考慮されません)。
特に明記しない限り、次のセクションでプライベートキャッシュをマスタープライベートキャッシュと呼びます。

一貫性は、どのエージェントによってもキャッシュされない共有メモリ領域を提供するのは簡単です。
このような領域の PMA は、それがプライベートまたは共有キャッシュにキャッシュされるべきでないことを単に示します。

一貫性は、読み取り専用領域でも簡単です。この領域は、キャッシュ一貫性スキームを必要とせずに複数のエージェントによって安全にキャッシュできます。
この領域の PMA はキャッシュできることを示しますが、書き込みはサポートされていません。

一部の読み取り/書き込み領域は単一エージェントによってのみアクセスされる可能性があり、その場合、コヒーレンススキームを必要とせずにそのエージェントによってプライベートにキャッシュされることがあります。

そのような領域の PMA は、それらをキャッシュできることを示します。
他のエージェントがその領域にアクセスすべきでないため(アクセスしないように)、データを共有キャッシュにキャッシュすることもできます。

エージェントが他のエージェントがアクセスできる読み取り/書き込み領域をキャッシュできるかどうかに関わらず、無効な値の使用を避けるためにキャッシュコヒーレンススキームが必要です。
ハードウェア・キャッシュ一貫性のない領域（ハードウェア非干渉性領域）では、キャッシュ・コヒーレンスをソフトウェアで完全に実装できますが、ソフトウェア・コヒーレンス・スキームは正しく実装するのが難しいことが知られており、保守的なソフトウェア指向キャッシュシングの必要性のためにパフォーマンスに大きな影響を与えることがよくあります。
ハードウェアキャッシュコヒーレンス方式は、より複雑なハードウェアを必要とし、キャッシュコヒーレンスプローブによるパフォーマンスに影響を与える可能性がありますが、それ以外の場合はソフトウェアからは見えません。

ハードウェア・キャッシュ一貫性領域ごとに、PMA は、領域にコヒーレント性があり、システムに複数のコヒーレンス・コントローラがある場合にどのハードウェア・コヒーレンス・コントローラを使用するかを示します。
システムによっては(一部のシステムでは)、コヒーレンス・コントローラが外部レベルの共有キャッシュであり、それ自体がさらに外部レベルのキャッシュ・コヒーレンス・コントローラに階層的にアクセスする場合があります。

プラットフォーム内のほとんどのメモリ領域は、キャッシュされていない、読み取り専用の、ハードウェアのキャッシュ一貫性のある、または 1 つのエージェントだけがアクセスするものとして固定されるため、ソフトウェアに一貫性があります。

3.5.6 冪等性 PMAs

冪等性 PMA は、アドレス領域への読み取りと書き込みが冪等であるかどうかを示します。
主記憶領域は冪等元であると仮定されます。
I / O 領域では、読取りおよび書込みの冪等性を別々に指定することができます（例えば、読取りは冪等であるが、書込みはそうでない）。
アクセスが制限されていない(冪等で無い)場合、すなわち、読出しまたは書込みアクセスに副作用が潜在的に存在する場合、投機的または冗長アクセスは避けなければなりません。

冪等性 PMAs を定義するために、冗長アクセスによって作成された観測メモリ順序の変更は、副作用とはみなされません。

-- 冪等性（べきとうせい）は、ある操作を 1 回行っても複数回行っても結果が同じであることをいう概念。ええっと、再現性があるよ。ってことでいいかな。まえに出てきたよね。（すっかり忘れてる）

ハードウェアは、非冪等としてマークされたメモリ領域への投機的または冗長アクセスを避けるように常に設計されなければなりませんが、ソフトウェアまたはコンパイラの最適化が非冪等のメモリ領域への偽のアクセスを生成しないようにすることも必要です。

非冪等領域は、整列していないアクセスをサポートしていない可能性があり、この場合、ソフトウェアは整列していないアクセスを一連の整列したアクセスとしてエミュレートし、それぞれが副作用を引き起こす可能性があります
したがって、ポータブルソフトウェアは、非冪等領域への不整合アクセスを発行してはいけません。

3.6 物理メモリ保護

安全な(セキュリティで保護)処理をサポートし、障害を含むためには、ハート上で動作するソフトウェアによってアクセス可能な物理アドレスを制限することが望ましいです。
オプションの物理メモリ保護（PMP）ユニットは、ハート単位マシンモード制御レジスタを提供し、各物理メモリ領域に対して物理メモリアクセス特権（読取り、書込み、実行）を指定できるようにします。
PMP 値は、セクション 3.5 で説明した PMA チェックと並行してチェックされます。

PMP アクセス制御設定の粒度はプラットフォーム固有であり、プラットフォーム内では物理メモリ領域によって異なる場合がありますが、標準 PMP エンコードでは 4 バイトという小さな領域がサポートされます。
特定の領域の特権は固定式(ハードワイヤ化)にすることができます。たとえば、一部の領域はマシンモードでしか表示されないかもしれませんが、特権レベルの低い層は表示されません(存在しない可能性があります)。

プラットフォームは、物理メモリ保護の要求が大きく異なり、一部のプラットフォームでは、このセクションで説明するスキームに加えて、またはスキームの代わりに他の PMP 構造を提供する(プラットフォームも)場合があります。

ハートが S モードまたは U モードで動作している場合、PMP チェックはすべてのアクセスに適用され、
mstatus レジスタに MPRV ビットが設定され、mstatus レジスタの MPP フィールドに S または U が含まれている場合、ロードおよびストアのために使用されます。
実効特権モードが S である仮想アドレス変換のページテーブルアクセスにも PMP チェックが適用されます。
オプションで、PMP チェックがさらに M モードアクセスに適用される場合があります。その場合、PMP レジスタ自体はロックされているため、M モードソフトウェアでも(でさえ)システムリセットなしで変更することはできません。
実際には、PMP は S と U モード（デフォルトでは none）に権限を与え、デフォルトで完全な権限を持つ M モードから権限を取り消すことができます。

PMP 違反は常にプロセッサーに(で)正確にトラップされます。

3.6.1 物理メモリ保護の CSR

PMP エントリは、8 ビット構成レジスタと 1 つの XLEN ビット・アドレス・レジスタによって記述されます。
一部の PMP 設定では、前の PMP エントリに関連付けられたアドレスレジスタが追加で使用されます。
最大 16 の PMP エントリがサポートされています。
PMP エントリが実装されている場合は、すべての PMP CSRs を実装する必要がありますが、すべての PMP CSR フィールドは WARL であり、ハードウェアでゼロにすることができます(ゼロに固定される可能性があります。)。
PMP CSRs は M モードでのみアクセスできます。

PMP コンフィギュレーションレジスタはコンテキストスイッチ時間を最小限に抑えるために、密集して CSR にパックされています。
RV32 の場合、4 つの CSR、pmpcfg0-pmpcfg3 は、図 3.23 に示すように、16 個の PMP エントリに対応してコンフィギュレーション pmp0cfg-pmp15cfg を保持します。
RV64 の場合、図 3.24 に示すように、pmpcfg0 と pmpcfg2 は 16 個の PMP エントリの設定を保持します。 pmpcfg1 と pmpcfg3 は不正です。

RV64 システムでは、pmpcfg1 ではなく pmpcfg2 を使用して、PMP エントリ 8～15 の設定を保持します。
この設計は、PMP エントリ 8-11 の設定が RV32 と RV64 の両方について pmpcfg2 [31：0]に表示されるため、複数の M-XLEN 値をサポートするコストを削減します。

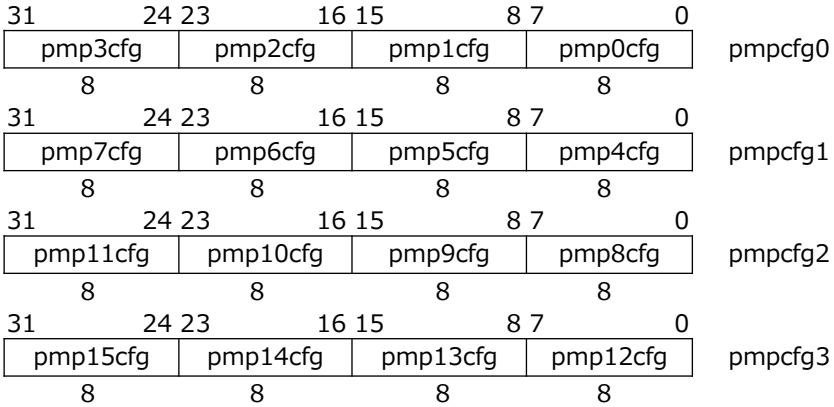


図 3.23：RV32 PMP 構成の CSR レイアウト。

PMP アドレスレジスタは、pmpaddr0-pmpaddr15 という名前の CSRs です。
各 PMP アドレスレジスタは、図 3.25 に示すように、RV32 の 34 ビット物理アドレスのビット 33-2 を符号化します。
RV64 の場合、図 3.26 に示すように、各 PMP アドレスレジスタは 56 ビットの物理アドレスのビット 55-2 をエンコードします。
すべての物理アドレスビットが実装されるわけではないので、pmpaddr レジスタは WARL です。

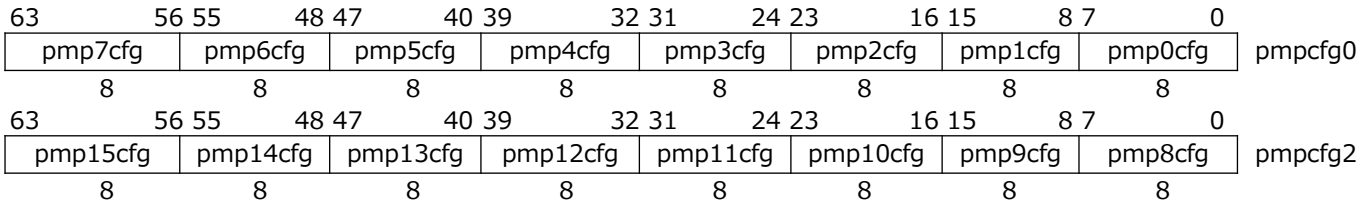
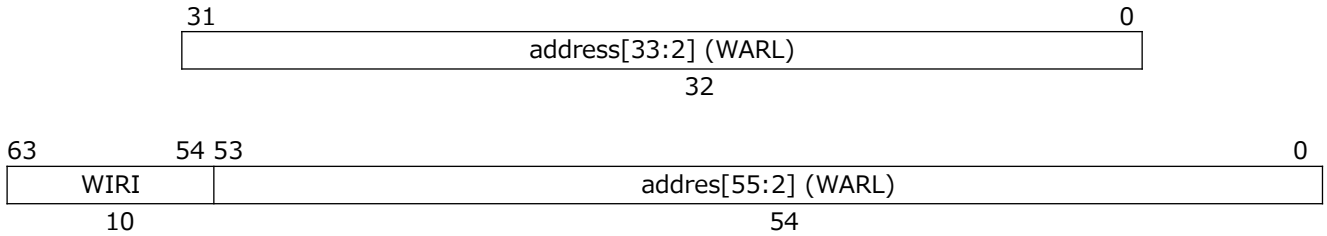


図 3.24：RV64 PMP 構成の CSR レイアウト。

セクション 4.3 で説明した Sv32 ページベースの仮想メモリスキームは、RV32 用の 34 ビット物理アドレスをサポートしているため、PMP スキームは RV32 用の XLEN より広いアドレスをサポートする必要があります。
セクション 4.4 と 4.5 で説明されている Sv39 と Sv48 のページベース仮想メモリスキームは 56 ビットの物理アドレス空間をサポートしているため、RV64 PMP アドレスレジスタは同じ制限を課しています。



– ここは元の図のビット幅が間違っている WIRI が 32 になっているが、ここは 10。 コピペして修正忘れたのかな。

図 3.26：PMP アドレスレジスタ形式、RV64。

図 3.27 に PMP コンフィグレーションレジスタのレイアウトを示します。
R ビット、W ビット、および X ビットがセットされると、PMP エントリがそれぞれ読み出し、書き込み、および命令の実行を許可することを示します。
これらのビットの 1 つがクリアされると、対応するアクセスタイプは拒否されます。
残りの 2 つのフィールド A と L については、次のセクションで説明します。

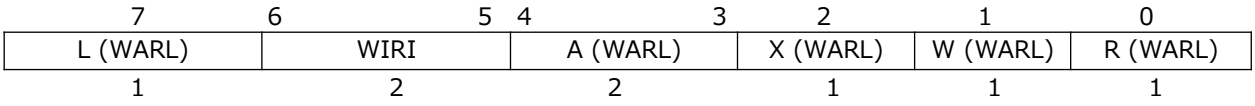


図 3.27：PMP 構成レジスタのフォーマット

アドレスマッチング(アドレスの一致)

PMP エントリのコンフィギュレーションレジスタの A フィールドは、関連する PMP アドレスレジスタのアドレスマッチングモードをエンコードします。
このフィールドのエンコーディングを表 3.8 に示します。
A = 0 の場合、この PMP エントリは無効になり、アドレスと一致しません。
その他の 2 つのアドレスマッチングモードがサポートされています。：自然に整列した 2 の累乗（power-of-2）領域（NAPOT）、自然にアライメントされた 4 バイト領域（NA4）の特別な場合を含む。任意の範囲（TOR）の上端の境界です。
これらのモードでは、4 バイトの細分性がサポートされます。

NAPOT 範囲は、表 3.9 に示すように、関連するアドレスレジスタの下位ビットを使用して範囲のサイズを符号化します。

TOR が選択されると、関連するアドレスレジスタがアドレス範囲の先頭を形成し、先行する PMP アドレスレジスタがアドレス範囲の最下部を形成します。

A	名前	説明
0	OFF	ヌル領域（無効）
1	TOR	範囲のトップ
2	NA4	自然に整列した 4 バイト領域
3	NAPOT	自然に整列した 2 の累乗≧8 バイト

表 3.8：PMP コンフィギュレーションレジスタの A フィールドのエンコーディング。

pmpaddr	pmpcfg.A	マッチタイプとサイズ
aaaa...aaaa	NA4	4-byte NAPOT 範囲
aaaa...aaa0	NAPOT	4-byte NAPOT 範囲
aaaa...aa01	NAPOT	4-byte NAPOT 範囲
aaaa...a011	NAPOT	4-byte NAPOT 範囲
…	…	…
aa01...1111	NAPOT	2^{XLEN}-byte NAPORT 範囲
a011...1111	NAPOT	2^{XLEN}+1-byte NAPORT 範囲
0111...1111	NAPOT	2^{XLEN}+2-byte NAPORT 範囲

表 3.9：PAP アドレスと構成レジスタの NAPOT 範囲の符号化。

PMP エントリ i's の A フィールドが TOR に設定されている場合、そのエントリは、pmpaddr $_{i-1}$ ≦ a <pmpaddr $_i$ のような任意のアドレス a に一致する。
PMP エントリ 0's の A フィールドが TOR に設定されている場合、下限にゼロが使用されるため、a <pmpaddr$_0$ の任意のアドレスに一致します。

ロックモードと特権モード

L ビットは、PMP エントリがロックされていること、すなわち、構成レジスタおよび関連するアドレスレジスタへの書込みは無視される事を示します。
ロックされた PMP エントリは、システムリセットでのみロック解除できます。
PMP エントリ i がロックされている場合、pmpicfg および pmpaddri への書き込みは無視されます。
さらに、pmpicfg.A が TOR に設定されていると、pmpaddr i-1 への書き込みは無視されます。

L ビットは、PMP エントリをロックすることに加えて、M モードアクセスで R / W / X パーミッション(アクセス許可)が強制(適用)されるかどうかを示します。
L ビットがセットされると、これらのパーミッション(アクセス許可)はすべての特権モードに対して強制(適用)されます。
L ビットがクリアされると、PMP エントリに一致するすべての M モードアクセスは成功します。 R / W / X パーミッション(アクセス許可)は S モードと U モードにのみ適用されます。

優先順位とマッチング(一致)ロジック

PMP エントリは静的に優先順位が付けられます。
アクセスの任意のバイトに一致する最も小さい番号の PMP エントリは、そのアクセスが成功するか失敗するかを決定します。
一致する PMP エントリは、アクセスのすべてのバイトと一致する必要があります。または L、R、W、および X ビットに関係なく、アクセスが失敗します。 -- ここは　or the access fails がどこにかかるのかな。これでいいのか。
たとえば、PMP エントリが 4 バイトの範囲 0xC-0xF に一致するように設定されている場合、PMP エントリがそれらのアドレスに一致する最も優先順位の高いエントリであると仮定すると、0x8～0xx の範囲への 8 バイトアクセスは失敗します 。

PMP エントリがアクセスのすべてのバイトに一致する場合、L、R、W、および X ビットは、アクセスが成功するか失敗するかを決定します。
L ビットがクリアされ、アクセスの特権モードが M の場合、アクセスは成功します。
それ以外の場合、L ビットがセットされているか、またはアクセスの特権モードが S または U である場合、アクセスタイプに対応する R、W、または X ビットがセットされている場合にのみアクセスが成功します。

M モードアクセスに一致する PMP エントリがない場合、アクセスは成功します。
S モードまたは U モードのアクセスと一致する PMP エントリがないのに少なくとも 1 つの PMP エントリが実装されている場合、アクセスは失敗します。

失敗したアクセスは、ロード、ストア、または命令アクセス例外を生成します。
1 つの命令が複数のアクセスを生成する可能性があり、相互にアトミックでない可能性があります(あることに注意してください)。
ある命令によって生成された少なくとも 1 つのアクセスが失敗した場合、アクセス例外が生成されますが、その命令によって生成された他のアクセスは可視の副作用で成功する可能性があります。
特に、仮想メモリを参照する命令は、複数のアクセスに分解されます。

いくつかの実装では、ミスアライメント(不整列)されたロード、ストア、および命令フェッチも複数のアクセスに分解され、そのうちのいくつかはアクセス例外が発生する前に成功する場合があります。
特に、別の部分が PMP チェックに失敗しても、PMP チェックをパス(通貨)する整列していないストア(保存)の一部が表示されることがあります。
ストアアドレスが自然に整列されている場合でも、XLEN ビット（例えば、RV32D の FSD 命令）よりも広い浮動小数点ストアについては、同じ挙動が現れることがあります。

- ↑ここ 3 章は文章はそう長くないのだけど、意味合いがどうも分かりにくいので何となく訳が変

-- 2018/12/08

第２巻：RISC-V 特権アーキテクチャ V1.10

第４章

監督者レベル ISA、バージョン 1.10

--ここまで。これ以降はこれから翻訳。 @shibatchii