Vol.2021-ARC-244 No.12
Vol.2021-SLDM-194 No.12
Vol.2021-EMB-56 No.12
2021/3/25

RISC-V 向けハイパーバイザを用いた H/W 追加機能分離の検討

山本遼介† 桐村昌行†

概要： RISC-V は、オープンなアーキテクチャという特徴から今後組込み機器での採用が増加する予想される。特にオープンで提供される標準の ISA(以降、標準 ISA)に対して、ベンダ独自の ISA(以降、拡張 ISA)を容易に追加できることを特長としている。一方で、独自の改変が多い拡張 ISA はベンダ共通で利用する標準 ISA と比較し、障害発生の頻度が高くなる傾向があり、拡張 ISA で最適化した OS に至ってはシステム全体へ影響を及ぼす可能性が高い。そこで、拡張 ISA の利用環境を主要システムから隔離することで、不具合の影響を局所化する手法を提案する。本手法の実現に向け、RISC-V の仮想化支援機能を用いることで、標準 ISA と拡張 ISA の OS 環境を分離する方法を検討した。その結果、ドラフト版の RISC-V 仮想化支援機能であっても提案手法実現の見込みを得た。

キーワード：RISC-V 仮想化支援機能 ハイパーバイザ OS 信頼性

The Isolated ISA Extension Environment for the RISC-V ISA with Virtualization Technology

Ryosuke Yamamoto† Masayuki Kirimura†

Keywords: RISC-V Virtualization Hypervisor Operating System Reliability

1. はじめに

オープンアーキテクチャである RISC-V は、ライセンス料や拡張性の観点から、今後組込み機器での採用が増加すると予想される。RISC-V の利用方法として、ベースとなる RISC-V の ISA(以降、標準 ISA と定義)に対してベンダ独自の命令やレジスタを追加することで、特定機能に特化した機能(以降、拡張 ISA と定義)を追加できることを特徴としている。例えば、AI 向け行列演算処理を機械語命令として実装し、使用頻度の高い計算を H/W にオフロードさせることで、演算を高速化させるといった使い方が期待できる。一方で、CPU に対して独自機能を追加することは脆弱性をを内在させてしまう危険性がある。例えば、Spectre[1]や Meltdown[2]は様々な CPU に内在する脆弱性であり、広く影響を与えている。このような脆弱性への対応として、OS やファームウェアの S/W アップデートが挙げられる。しかし、アップデートにより、OS に対する変更が必要となるが、OS に対する機能追加は不具合発生が懸念される。特に、OS 内の不具合はシステム全体に影響を及ぼし、機器が動作不能となる場合も考えられる。

そこで、標準 ISA のみを利用する動作環境と標準 ISA と拡張 ISA を両方利用する動作環境を分離することで、拡張 ISA における障害発生時の影響を局所化する手法を提案する。具体的には、ハイパーバイザを用いることで、標準 ISA のみを利用する仮想マシン(VM#1)と拡張 ISA を利用する仮想マシン(VM#2)の 2 種類を用意し、これを実現する。VM#1 上ではシステムの主体となる OS が動作し、必要に応じて VM#2 上のソフトウェアに拡張 ISA を用いた機能を要求することでシステム全体の動作を実現する。これにより、VM#2 において障害が発生した場合でも VM#2 内のみに障害影響を抑えることができ、VM#2 の単体再起動等の方法で復旧できると考えられる。

本手法の実現するためには、下記の機能が必要である。

(A) VM#1 での拡張 ISA に対するアクセス制御
(B) VM#2 への処理要求

機能(A)(B)の実現性について、RISC-V の仮想化支援機能[3]をベースに検討する。RISC-V 仮想化支援機能はドラフト版(Ver0.61)であるものの、いくつかの機能について定義されている。仮想化支援機能搭載の RISC-V プロセッサは、ゲスト OS とハイパーバイザ(ホスト OS)で異なる特権レベルを設定することが可能であり、一部の命令やレジスタを特権レベルごとにアクセス制御されている。一方で、ゲスト OS に対するページングやゲスト OS 内の例外をハイパーバイザでトラップする機能がある。以上より、機能(A)(B)は実現可能である見込みを得た。また、機能(A)(B)における処理オーバヘッドを削減するための H/W 拡張機能についても検討した。

そこで、本稿ではこれらの検討結果について述べる。2 章で課題、3 章で提案手法、4 章で実現性検討、5 章で今後の課題について述べ、6 章でまとめる。

1 三菱電機株式会社 情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation

Vol.2021-ARC-244 No.12
Vol.2021-SLDM-194 No.12
Vol.2021-EMB-56 No.12
2021/3/25

2. 課題

本章では、RISC-V に対して拡張 ISA を実装する際の課題について述べる。

2.1 CPU 脆弱性に対する課題

拡張 ISA を実装する際、CPU の脆弱性への対応を検討する必要がある。近年、Spectre や Meltdown といった CPU 脆弱性が問題となっている。これらの脆弱性は、Intel、AMD といった CPU ベンダによって開発された CPU に対しても確認されている。RISC-V においては、CPU ベンダだけでなく、より多くの開発者によって拡張 ISA を実装され、公開されることが想定される。このような拡張 ISA を搭載した RISC-V の CPU を製品適用する際には、CPU 脆弱性への対応が求められることになる。

これらの脆弱性への対応として、S/W アップデートを可能としておくことは製品化において重要である。例えば、製品発売後に脆弱性を確認した場合、S/W アップデートで対応不可能であれば、製品回収が必要となる。そのため、拡張 ISA に対して、特定の S/W だけが直接操作可能としておき、当該 S/W をアップデートできるような仕組みが必要となる。

S/W アップデート可能とするためには、拡張 ISA を特権化することで、OS 内でのみ操作可能とする方法が考えられる(図 1)。特権化せずに拡張 ISA を実装した場合、アクセス権限の確認無しに命令やレジスタにアクセスできるため、第三者アプリによる直接的なアクセスが可能となる。この場合、S/W によるアクセス制御は不可能であるため、拡張 ISA の特権化は必須である。以上を踏まえると、OS 内で拡張 ISA を利用する機能を用意し、この機能をアップデートできるようにしておく必要がある。

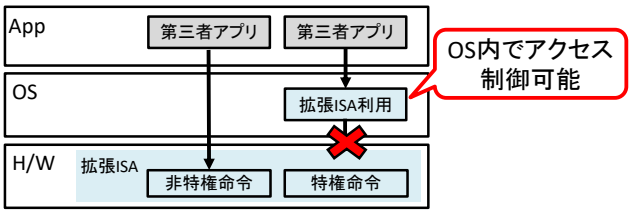

図 1 OS 内でのアクセス制御

2.2 OS 内の不具合

前節で述べたとおり、拡張 ISA 実装時、拡張 ISA を利用する S/W は OS 内で動作させることが求められる。しかし、拡張 ISA 利用 S/W を OS 内で動作させる際、不具合への対策を検討する必要がある。これは、OS 内で動作する S/W の不具合はシステム全体へ影響を及ぼすためである。例えば、Linux であれば、OS 内で障害を検出した際に kernel panic を発生させ、システムを停止させる。その結果、システム全体の再起動といった対応が必要となる。

以上より、拡張 ISA 利用 S/W を動作させる場合に、不具合によるシステム全体への影響が課題となる。

3. 方式検討

3.1 要件

拡張 ISA 利用 S/W での不具合発生時の影響を局所化するためには、拡張 ISA 利用 S/W を隔離する方法が考えられる。拡張 ISA 利用 S/W での不具合発生時は動作不定であるため、OS 内では障害影響を抑えきれない可能性がある。そのため、OS 外で拡張 ISA 利用 S/W を隔離する手法を検討する。

3.2 提案方式

ハイパーバイザを用いることで、拡張 ISA 利用 S/W を隔離する手法を提案する。図 2 に提案手法の概要を示す。ハイパーバイザ上に 2 種類の VM を動作させる。VM#1 は、標準 ISA のみ搭載した仮想環境を提供し、VM#2 は、標準 ISA と拡張 ISA の両方を搭載した仮想環境を提供する。VM#1 上では機器の主要機能を果たす OS やアプリケーションを動作させる。VM#2 では拡張 ISA を利用する処理を行う。VM#1 は、拡張 ISA の機能が必要になった際に VM#2 に処理を依頼し、結果を受け取る。この構成であれば、拡張 ISA を利用する場合には、VM#2 を必ず経由するため、VM#2 の S/W アップデートによって脆弱性への対応が可能となる。また、VM#2 で不具合が発生した場合でも VM#1 への影響を抑えることができる。

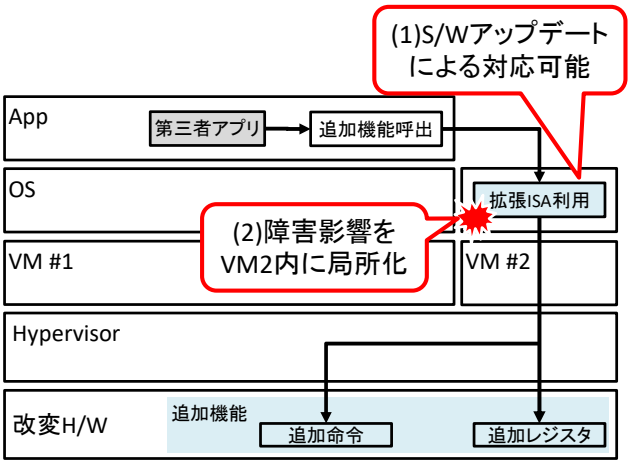

図 2 提案手法

3.3 要求機能

本方式を実現するためには、下記 2 機能が必要である。

(A) VM#1 での拡張 ISA に対するアクセス制御
(B) VM#2 への処理要求

Vol.2021-ARC-244 No.12
Vol.2021-SLDM-194 No.12
Vol.2021-EMB-56 No.12
2021/3/25

機能(A)が無い場合、VM#1 から拡張 ISA へのアクセスを防ぐことができず、S/W アップデートによる対処が不可能になる。また、機能(B)が無い場合、拡張 ISA を利用することができない。

4. 実現性検討

RISC-V アーキテクチャにおける提案方式の実現性を検討する。RISC-V は、2021 年 1 月現在、仮想化支援機能のドラフト版(Ver0.61)が仕様として公開されている。そこで、ドラフト版の RISC-V 仮想化支援機能をベースとして、検討を進める。本章では、提案方式に利用可能な RISC-V の仮想化支援機能について述べた後、実現性について検討する。

4.1 RISC-V の仮想化支援機能

4.1.1 特権モード

仮想化支援機能が搭載されている場合、通常の特権モードとは異なる構成で動作する。表 1 に仮想化支援機能非搭載時、表 2 に仮想化支援機能搭載時の特権モード一覧を示す。また、各特権モードの S/W 構成を図 3 に示す。

表 1 仮想化支援機能非搭載時の特権モード一覧

略称	説明
M-mode	ファームウェアが動作するマシンモード。最も高い権限を持つ。
S-mode	スーパーバイザモード。OS が動作する。
U-mode	ユーザアプリが動作するユーザモード。

表 2 仮想化支援機能搭載時の特権モード一覧

略称	説明
M-mode	ファームウェアが動作するマシンモード。最も高い権限を持つ。
HS-mode	ハイパーバイザ拡張スーパーバイザモード。ハイパーバイザ、ホスト OS が動作する。
U-mode	ホスト OS のユーザアプリが動作するユーザモード。
VS-mode	仮想スーパーバイザモード。ゲスト OS が動作する。
VU-mode	仮想ユーザモード。ゲスト OS 上のユーザアプリが動作する

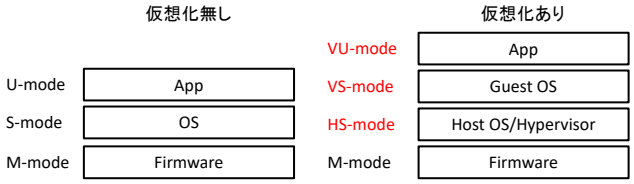

図 3 各特権モードでの動作 S/W 構成

特権モード専用の命令やレジスタ(Control and Status Register(以降、CSR))が存在する。例えば、HLV(ハイパーバイザ向け仮想マシンロード)命令は、M-mode、HS-mode、U-mode でのみ実行可能な命令である。また、hstatus レジスタは、HS-mode でのみ操作可能なレジスタである。これらの命令やレジスタに対して許可されていない特権モードでアクセスした場合、例外が発生する。これを利用することで、ゲスト OS での命令やレジスタに対するアクセスを制御することが可能である。

4.1.2 例外のトラップ

仮想マシン上で発生した例外は、任意の特権モードでトラップ可能である。基本的に VS-mode や VU-mode で発生した例外は、M-mode でトラップされる。その際、M-mode において、medeleg や mideleg といったレジスタを設定することで、HS-mode に例外を委任することが可能である。また、HS-mode においても hedeleg や hideleg レジスタを設定することで、VS-mode に例外を委任することが可能である。以上より、VU-mode で発生した例外をゲスト OS やハイパーバイザにトラップさせ、制御を移すことが可能である。

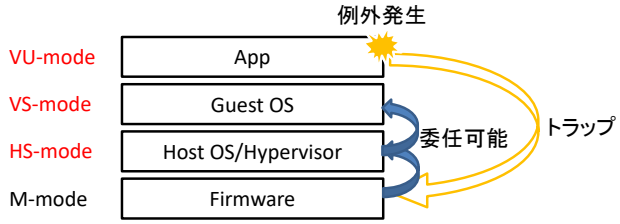

図 4 例外のトラップ

4.1.3 2 段アドレス変換

ゲスト OS に対して仮想的な物理アドレスを提供することで、ゲスト OS ごとに利用できるアドレス空間を制御できる。図 4 に各レイヤでのアドレス空間を示す。ハイパーバイザはゲスト OS に対して仮想の物理アドレスを提供する。さらに、ゲスト OS がアプリケーションに対して、仮想アドレスを提供することが可能である。これにより、ゲスト OS ごとにアクセス可能な物理アドレス空間を制御できる。

Vol.2021-ARC-244 No.12
Vol.2021-SLDM-194 No.12
Vol.2021-EMB-56 No.12
2021/3/25

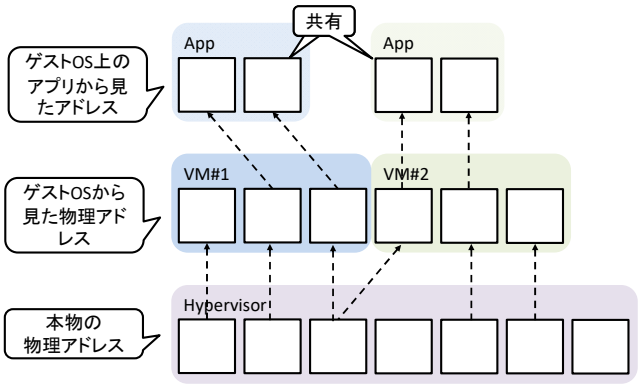

図 5 各レイヤへのアドレス空間の提供

本機能を用いることで、アクセス制御が可能なだけでなく、複数の VM 上で共通のアドレス空間を持たせることが可能である。これにより、VM 間でのデータ共有が可能となる。

4.2 機能(A) VM#1 での拡張 ISA に対するアクセス制御

前章で述べた機能(A)について、実現性を検討する。拡張 ISA は、命令、レジスタ(CSR)、メモリ(MMIO)を介して機能を提供するものと想定し、検討する。

RISC-V アーキテクチャにおいて、命令やレジスタに対するアクセス権は、特権モードで判断される。つまり、VM#1 で動作する VU-mode から拡張 ISA の命令やレジスタへのアクセスを制限するためには、VU-mode よりも高い権限、つまり VS-mode、HS-mode や M-mode 用の命令・レジスタとして追加する必要がある。一方で、VM#2 にはアクセス権を与える必要がある。そのため、VS-mode でのみアクセス可能な命令・レジスタを実装する。これにより、VU-mode での拡張 ISA アクセスがあった場合、例外が発生し、アクセスを防ぐことが可能となる。

メモリに対してのアクセス制御は、ハイパーバイザが VM#1 にのみ、当該アドレスを提供しないことで可能である。

4.3 機能(B) VM#2 への処理要求

4.3.1 実現方法の検討

前章で述べた機能(B)について、実現性を検討する。処理要求を VM#2 で受け付ける方法は次の 2 種類に分類できる。

- 定期的に VM#1 からの要求を確認する
- VM#1 から VM#2 への割り込み/例外を発行する

前者は、VM#2 が自発的に VM#1 の要求を確認する方法である。例えば、VM#1 は、特定の要求を共有メモリに書き込み、VM#2 はポーリングで共有メモリを監視するといった方法である。しかし、この方法は VM#2 が定期的に動作する必要があり、VM#2 が CPU 時間を必要以上に消費してしまうケースが想定される。

後者は、VM#1 が必要なタイミングでのみ VM#2 を呼び出し、処理を要求する方法である。例えば、VM#1 が特定のタイミングで、ハイパーバイザへ割込みを発行し、ハイパーバイザが VM#2 へ割込みを発行することで、VM#2 に処理を要求する。この方法であれば、前者のような CPU 利用時間の浪費が防げる。

4.3.2 例外発生方法の検討

機能(B)の具体的な実現方法について検討する。RISC-V では、例外を発生させることで、ハイパーバイザやゲスト OS に対して制御を移すことが可能である。例外は、hedeleg レジスタや medeleg レジスタでトラップ先を決定することが可能である。例外の種類はいくつか存在するが、ゲスト OS でトラップする必要が無く、ゲスト OS 上から発生可能な例外は下記のとおりである。

(1) Environment call from VS-mode
(2) Virtual instruction

例外(1)は VS-mode(ゲスト OS)において、ecall 命令を実行した際に発生する例外である。この例外を用いることで、ハイパーバイザを呼び出すことが可能ではあるが、一度 VS-mode を経由する必要がある。つまり、VM#1 のゲスト OS 内で独自 S/W を動作させる必要がある。ハイパーバイザを呼び出すための S/W で不具合が発生した場合、VM#1 内に影響を及ぼす。また、ゲスト OS を呼び出すため、オーバヘッドが生じる。以上より、前者の例外は本手法の採用に適していない。

例外(2)は、ハイパーバイザ用の命令やレジスタにアクセスした際に発生する。つまり、拡張 ISA として追加した命令・レジスタへのアクセス時、この例外が発生するように実装することで VM#1 上では拡張 ISA に直接アクセスする場合と同様の方法でアクセスできる。この例外であれば VS-mode を経由する必要が無いため、前者のような課題は生じない。よって、追加した命令・レジスタへのアクセス時に Virtual instruction 例外を発生させる方法が適している。

VM#1 から VM#2 へ制御を移すためには、VM#1 から VM#2 にコンテキストを切り替える必要がある。具体的には、例外ベクタのアドレスやページテーブルを VM#2 向けに切り替える必要がある。そのため、VM#2 に制御を移すためには、ハイパーバイザを一度経由する必要がある。

以上より、Virtual instruction 例外発生時、HS-mode でトラップするように medeleg レジスタを設定することで、本手法を実現することが適切である。

Vol.2021-ARC-244 No.12
Vol.2021-SLDM-194 No.12
Vol.2021-EMB-56 No.12
2021/3/25

4.4 RISC-V における提案手法の構成

以上より、現状の RISC-V 仮想化支援機能であっても提案手法の実現は可能であると推測される。その場合、図 6 のような構成になる。VM#1 上のアプリケーションから拡張 ISA にアクセスすることで、ハイパーバイザに対して例外が発生する。ハイパーバイザは、VM 切替え処理を行い、VM#2 を呼び出す。VM#2 では拡張 ISA を用いた処理を実行する。

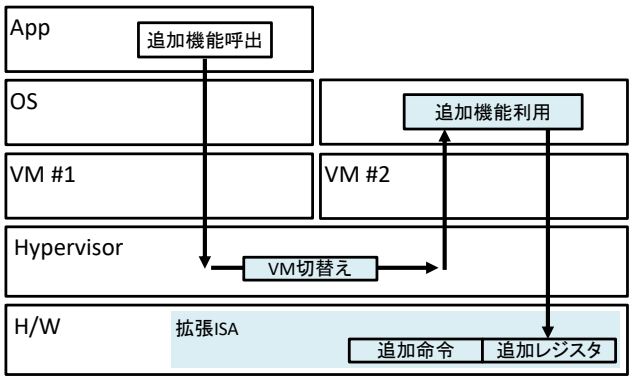

図 6 RISC-V における提案手法の構成

5. 今後の課題

図 6 の構成の場合、下記の課題がある。本章は下記の課題を解決する方法について検討する。

(ア) VM#1 上のカーネル空間へのアクセス制御
(イ) 拡張 ISA 利用時のオーバヘッド

5.1 VM#1 上のカーネル空間へのアクセス制御

本構成では、VM#1 のゲスト OS からアクセス可能である。つまり、ユーザアプリだけでなく、カーネルも含めたアクセス制限は不可能であると考えられる。そのため、カーネル空間をユーザに提供するようなシステム、例えば、デバイスドライバのインストールを許可するようなシステムでは採用できない。課題(ア)の原因として、RISC-V 仮想化支援機能(Ver0.61)は、VM ごとのアクセス制御が不可能であることが挙げられる。この解決策として、仮想化支援機能に対して、各 VM を識別できるような仕組みを実装する方法が考えられる。既存技術としては、Intel の仮想化支援機能(Intel VT)が挙げられる。Intel VT は、Virtual Machine Control Structure (VMCS)というデータ構造を用いて VM ごとにいくつかの設定が可能である。VMCS を用いることで、VM ごとに命令やレジスタへのアクセスを制御することができる。このように、特定のデータ構造を作成し、VM ごとに命令やレジスタへのアクセス権を設定可能であれば、VM#1 からはアクセス不可能かつ VM#2 からはアクセス可能といった構成が実現できる。

5.2 拡張 ISA 利用時のオーバヘッド

前章で述べたとおり、VM#2 へ要求を出すためには、ハイパーバイザを経由する必要がある。そのため、命令・レジスタへのアクセスにオーバヘッドが生じる。そのため、高頻度にアクセスするような用途には向かない。この原因として、VM 間の直接的な制御の遷移が困難なことが挙げられる。この課題に対しても、VM ごとの区別が可能であれば、HW 内でコンテキストの切替えが可能であるため、VM 間での直接的な制御の遷移が可能となる。

6. おわりに

本稿は、ハイパーバイザを用いることで、 拡張 ISA を分離し、障害を局所化する手法を提案した。本手法は、仮想化技術を用いることで、標準 ISA のみを利用する仮想マシンと拡張 ISA を利用可能な仮想マシンを用意する。拡張 ISA を利用する場合は、後者の仮想マシンに対して要求を出す。これにより、拡張 ISA 利用時における S/W の障害を局所化する。検討の結果、RISC-V アーキテクチャにおいて本手法を実現する場合、仮想化支援機能を用いることで実現できる見込みを得た。しかし、現状の仮想化支援機能(Ver0.61)においては、アクセス制御や処理要求におけるオーバヘッドが生じる見込みがある。これを解決するためには、VM ごとの区別をつける機能を H/W に持たせる方法が考えられる。今後は、H/W 側の改変を検討し、前述した課題解決を目指す。

参考文献

[1] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg,M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M.and Yarom, Y.: Spectre Attacks: Exploiting SpeculativeExecution, ArXiv e-prints (2018)
[2] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg “Meltdown” ArXiv e-prints (2018).
[3] “RISC-V Instruction Set Manual”. https://github.com/riscv/riscv-isa-manual (参照 2021-02-03).