RISC-V Pointer Masking

Version 0.3, 10/2022: Development

Table of Contents

Preface

The proposed mechanism is an independent ISA extension developed as part of the RISC-V J Extension. The identifier of this extension is `Zjpm`.

Authors: Adam Zabrocki, Martin Maas, Lee Campbell, RISC-V TEE and J Extension Task Groups

Contributors: Jecel Assumpcao, Allen Baum, Paul Donahue, Greg Favor, Andy Glew, John Ingalls, Christos Kotselidis, Ian Rogers, Josh Scheid, Kostya Serebryany, Boris Shingarov, Foivos Zakkak, Members of the TEE and J Extension Task Groups

Chapter 1. Introduction

RISC-V Pointer Masking (PM) is a feature that, when enabled, causes the MMU to ignore the top N bits of the effective address (these terms will be defined more precisely in the Background section). This allows these bits to be used in whichever way the application chooses. Most commonly, these bits are used to store various types of tags, which can be leveraged by a number of hardware/software features, including sandboxing mechanisms and dynamic safety checkers such as HWASAN [1].

This extension only adds the pointer masking functionality. Extensions that make use of the masked bits will be ratified independently and layered on top of the basic pointer masking functionality. Examples of such extensions include:

- **Tag checks:** When a masked address is accessed, the tag stored in the masked bits is compared against a range-based tag. This is used for dynamic safety checkers such as HWASAN [1].
- **Sandbox enforcement:** When a masked address is accessed, the masked bits are checked against a particular sandbox tag to enforce that addresses are limited to a particular range in memory. This is used for isolating heap and runtime memory in a managed runtime, and isolation of untrusted code in M mode.
- **Read barriers:** When a masked address is accessed, the masked bits are checked against (e.g.,) a generation in a garbage collected environment, redirecting to slow path code on mismatch.
- **Object type checks:** When a masked address is accessed, the masked bits are checked against a fixed type tag in an object-oriented runtime, redirecting to slow path code on mismatch.

All of these use cases can be implemented in software or hardware. If implemented in software, pointer masking still provides performance benefits since all non-checked accesses do not need to unmask the address before every memory access. Hardware implementations are expected to provide even larger benefits due to performing tag checks out-of-band and hardening security guarantees derived from these checks.

Chapter 2. Background

2.1. Definitions

We now define basic terms. Note that these rely on the definition of an “ignore” transformation, which is defined in Chapter 2.2.

- **Effective address (as defined in the RISC-V Base ISA):** An address generated by the instruction fetch and load/store effective addresses sent to the memory subsystem. There is no special distinction of physical vs. virtual memory. This does not include addresses corresponding to implicit accesses, such as page table walks.
- **Masked bits:** The top N bits of an address, where N is a configurable parameter (we will use N consistently throughout this document to refer to this parameter).
- **Masked address:** An effective address after the ignore transformation has been applied to it.
- **Address translation mode:** The MODE of the currently active address translation scheme as defined in the RISC-V privileged specification. This could, for example, refer to Bare, Sv39, Sv48, and Sv57. In accordance with the privileged specification, non-Bare translation modes are refered to as virtual-memory schemes.
- **Address canonicity:** The RISC-V privileged spec defines canonicity of an address based on the privilege mode and address translation mode that is currently in use (e.g., Sv57, Sv48, Sv39, etc.). Canonicity enforces that for the translated addresses all bits in the unused portion of the address must be the same as the Most Significant Bit (MSB) of the used portion (for virtual addresses). For example, when page-based 48-bit virtual memory (Sv48) is used, instruction fetch addresses and load/store effective addresses, which are 64 bits, must have bits 63–48 all set to bit 47, or else a page-fault exception will occur. For untranslated addresses (i.e., for Bare and M-mode) canonicity requirement is for the high bits to be all zeros.
- **NVBITS:** The upper bits within an address that have no effect on addressing memory and are only used for canonicity checks. For example, in Sv48, these are bits 63-48.
- **VBITS:** The bits within an address that affect which memory is addressed.

2.2. The “Ignore” Transformation

The ignore transformation (Listing 1) is expressed independently of the current address translation mode. Intuitively, it replaces the top N bits with the sign extension of the N+1st bit from the top. If N is smaller or equal to NVBITS for the current address translation mode, this is equivalent to ignoring the NVBITS of the address, without actually performing the sign extension operation.

Listing 1. "Ignore" Transformation

```
transformed_effective_address =
  {{N{effective_address[XLEN-N-1]}}, effective_address[XLEN-N-1:0]}
```

When pointer masking is enabled, this transformation will be applied to every explicit memory access that is subject to the regular memory access path (i.e., subject to the current address translation mode). This includes, for example, atomics operations and floating point loads/stores. The transformation **does not** apply to implicit accesses such as page table walks, with the exception of instruction fetches

if the corresponding pointer masking feature is enabled.

When pointer masking is enabled, address canonicity checks will be disabled.

2.3. Allowed values of N

Ignoring bits is deeply connected to the address translation mode (e.g., Bare, Sv48, Sv57). In particular, applying the above transformation is cheap if it covers only the NVBITS (as it is equivalent to switching off canonicity checks) but expensive if the masked bits extend into the VBITS portion of the address (as it requires performing the actual sign extension).

Systems may therefore choose to disallow certain values of N. The minimum set of values of N that must be supported may be defined by an ISA profile. The supported values of N may differ between privilege modes and between address translation modes.

When trying to enable pointer masking and setting a value of N in the corresponding CSR (Chapter 3), an implementation may choose a different value of N and reflect this in the updated register value. It may choose a larger number of bits than the original requested number if and only if the additional bits that are masked have no impact on addressing under the current privilege mode and address translation mode. If a smaller value of N is returned, the implementation should choose the largest supported value smaller or equal to the requested number of masked bits.

We expect that the initial profile will only mandate that systems support one value of N: The number of NVBITS of the current privilege mode and address translation mode (which is equivalent to disabling the canonicity check).

2.4. Example

Table 1 shows an example of address translation when PM is enabled for RV64 under Sv57 with N=8.

Table 1. Example of PM address translation for RV64 under Sv57

Page based profile	Sv57 on RV64
Effective Address	0xAAFFFFFF12345678 NVBITS[1010101] VBITS[0111111111111111111111110001...000]
N	8
Mask	0x00FFFFFFFFFFFFFF NVBITS[0000000] VBITS[0111111111111111111111111111...111]
N+1st bit from the top	1
Transformed address	0xFFFFFFFF12345678 NVBITS[1111111] VBITS[1111111111111111111111110001...000]

Chapter 3. ISA Extension

The `zjpm` extension adds a new PM configuration CSR called mpm. Restricted views of the mpm register appear as the hspm, spm and upm registers in the HS-mode, (V)S-mode and (V)U-mode ISAs respectively. This results in a total of four new CSRs being assigned to pointer masking. All four CSRs are read/write.

The layout of mpm on RV64 can be found in Figure 1a, 1b, 1c and 1d for M, HS, (V)S and (V)U-mode. On RV32, the layout is equivalent but the **MPM.Ignore** bits occupy 4 bits instead of 6. On RV128 (if and when it is ratified), they may tentatively occupy 7 bits.

Table 1 describes the meaning of the PM bits for RV64.

*Figure 1a: Pointer Masking register **(mpm)** M-mode*

mpm[XLEN-1:36]	mpm[35:27]	mpm[26:18]	mpm[17:9]	mpm[8:0]
WPRI	M-mode PM	HS-mode PM	(V)S-mode PM	(V)U-mode PM

*Figure 1b: Pointer Masking register **(hspm)** HS-mode*

hspm[XLEN-1:27]	hspm[26:18]	hspm[17:9]	hspm[8:0]
WPRI	HS-mode PM	(V)S-mode PM	(V)U-mode PM

*Figure 1c: Pointer Masking register **(spm)** for VS-mode*

spm[XLEN-1:18]	spm[17:9]	spm[8:0]
WPRI	(V)S-mode PM	(V)U-mode PM

*Figure 1d: Pointer Masking register **(upm)** for (V)U-mode*

upm[XLEN-1:9]	upm[8:0]
WPRI	(V)U-mode PM

Table 1: Meaning of PM bits for RV32, RV64 and RV128

PM Bits	Name	Meaning
PM[0:0]	Enabled	0 – PM is disabled (*default*) 1 – PM is enabled The value of *MPM.Enabled* controls if PM is enforced for the corresponding privilege mode. It allows for the fine grain control of the local PM state.
PM[1:1]	Current	0 – ***Enabled*** and ***Ignore*** bits can only be modified by higher privilege modes (*default*) 1 – ***Enabled*** and ***Ignore*** bits can be modified by the same privilege mode ***Note***: The *MPM.Current* bit for M-mode (mpm[28:28]) is a WARL bit hardwired to 1

PM Bits	Name	Meaning
PM[2:2]	Instruction	0 – PM applies to Data accesses only (*default*) 1 – PM applies to Instruction and Data accesses ***Note***: The *MPM.Instruction* bit enables more use-cases. However, implementations can hardwire this bit to 0 if they choose not to implement instruction pointer masking
PM[8:3]	Ignore	The value of *MPM.Ignore* defines how many contiguous top-bits are ignored, in binary. For RV64, its value may range from 0 to 63. A detailed description of *MPM.Ignore* bits is provided in Chapter 2. This value was referred to as N above.

Note: The *default* state in the Table 1 references to the initial state of the PM after hart reset.

Writing one of the CSRs may result in the **MPM.Ignore** bits being set to a different value than what was requested. It may also result in no update at all, which indicates that an unsupported value of N was requested. In this case, no other field is updated either.

When a privilege mode's address translation mode changes, the corresponding PM bits are reset to 0. A privilege mode's address translation may change, for example, because the corresponding atp register was updated or because the privilege mode was changed while the corresponding atp register was holding a different address translation mode than when this privilege mode was last exited.

Bibliography

[1] Serebryany, Kostya, Evgenii Stepanov, Aleksey Shlyapnikov, Vlad Tsyrklevich, and Dmitry Vyukov. "Memory tagging and how it improves C/C++ memory safety." arXiv preprint arXiv:1802.09517 (2018).


RISC-V