RISC-V Memory Protection Unit (MPU)

Dong Du, Xu Lu, Bicheng Yang, Wenhao Li, Yubin Xia, Nick Kossifidis, Joe Xie, Paul Ku, Bill Huffman, Jonathan Behrens, Allen Baum, Robin Zheng, Zeyu Mi, RISC-V TEE Task Group

Version 0.7.0, 09/2021: This document is in the Development state. Assume anything can change. For more information see: https://riscv.org/spec-state

Table of Contents

Chapter 1. Motivation

We propose MPU (RISC-V Memory Protection Unit) to provide isolation when MMU is not available.

RISC-V based processors recently stimulate great interest in the emerging internet of things (IoT). However, as the page-based virtual memory (MMU) is usually not available on IoT devices, it is hard to isolate the S-mode OSes (e.g., RTOS) and user-mode applications. To support secure processing and isolate faults of U-mode software, it is desirable to enable S-mode OS to limit the physical addresses accessible by U-mode software on a hart.

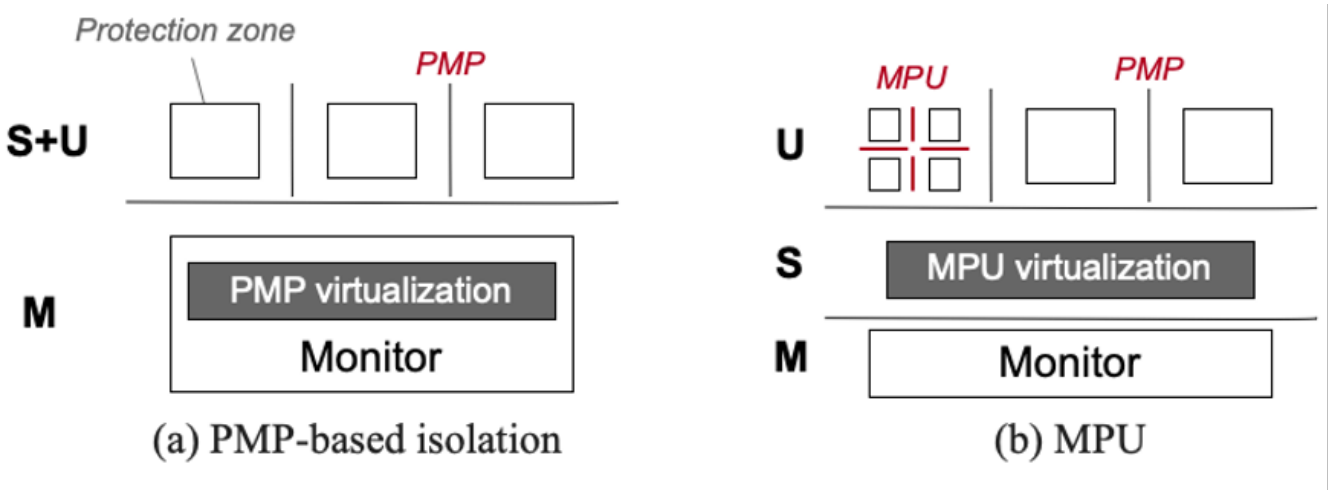

Figure 1. Comparison of PMP and MPU

Chapter 2. Memory Protection Unit(MPU)

An optional RISC-V Memory Protection Unit (MPU) provides per-hart supervisor-mode control registers to allow physical memory access privileges (read, write, execute) to be specified for each physical memory region. The MPU values are checked after the physical address to be accessed pass both the PMA checks and PMP checks described in the privileged spec.

Like PMP, the granularity of MPU access control settings are platform-specific and within a platform may vary by physical memory region, but the standard MPU encoding should support regions as small as four bytes.

MPU checks could be applied to all accesses for both U mode and S mode, depending on the values in the configuration registers. MPU registers can always be modified by M-mode and S-mode software. MPU registers can grant permissions to U-mode, which has none by default, and revoke permissions from S-mode, which has full permissions by default.

2.1. Requirements

1) S mode should be implemented

2.2. Memory Protection Unit Keys

Like PMP, MPU entries are described by an 8-bit configuration register and one XLEN-bit address register. Some MPU settings additionally use the address register associated with the preceding MPU entry. The number of MPU entries can vary by implementation, and up to 16 MPU entries are supported in standard.

The MPU configuration registers are packed into CSRs in the same way as PMP does. For RV32, four CSRs, mpucfg0-mpucfg3, hold the configurations mpu0cfg-mpu15cfg for the 16 MPU entries, as shown in Figure 2. For RV64, mpucfg0 and mpucfg2 hold the configurations for the 16 MPU entries, as shown in Figure 3; mpucfg1 and mpucfg3 are illegal.

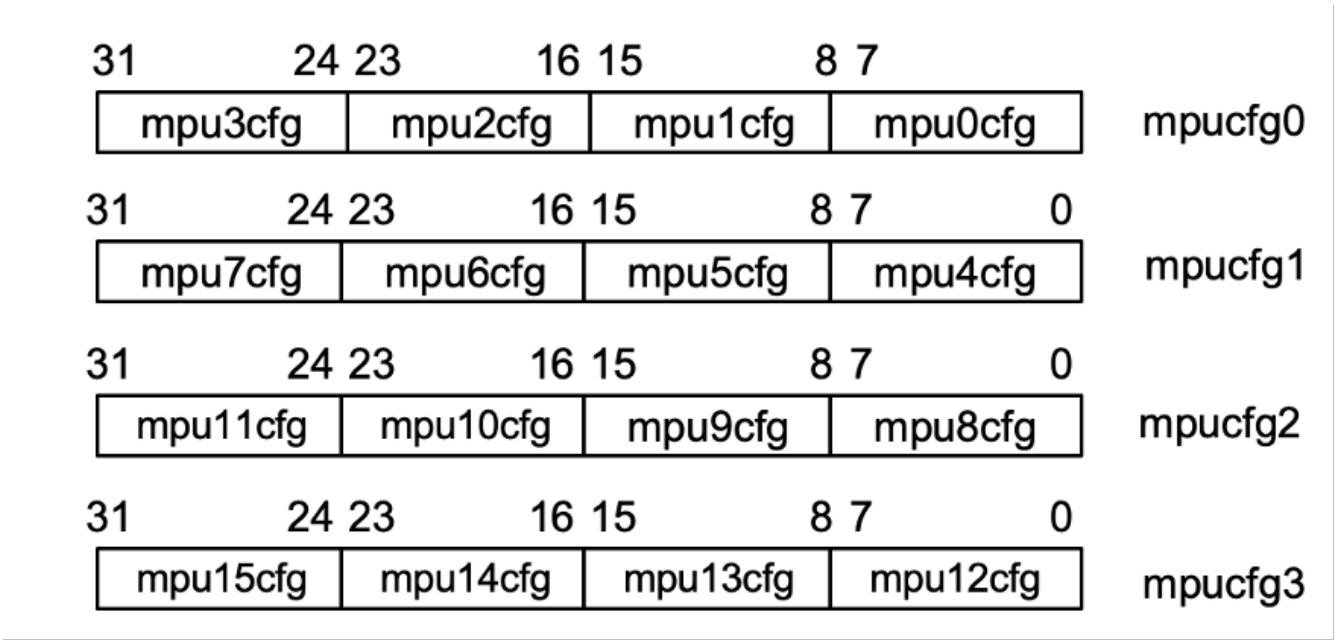

Figure 2. RV32 MPU configuration CSR layout

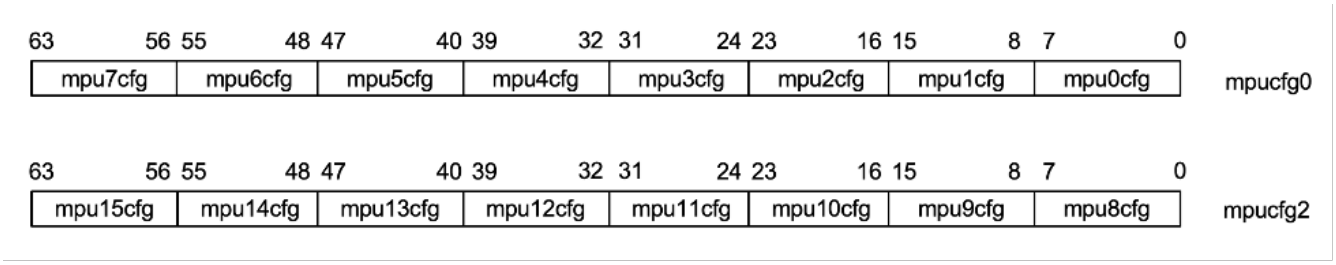

Figure 3. RV64 MPU configuration CSR layout

The MPU address registers are CSRs named mpuaddr0-mpuaddr15. Each MPU address register encodes bits 33-2 of 34-bit physical address for RV32, as shown in Figure 4. For RV64, each MPU address encodes bits 55–2 of a 56-bit physical address, as shown in Figure 5. Not all physical address bits may be implemented (but implemented RW bits must be contiguous), and so the mpuaddr registers are WARL.

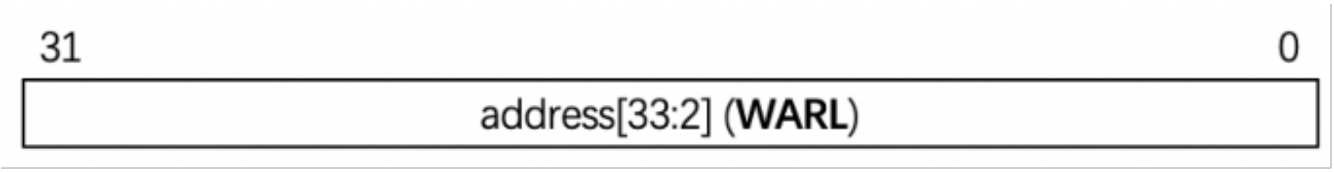

Figure 4. MPU address register format, RV32

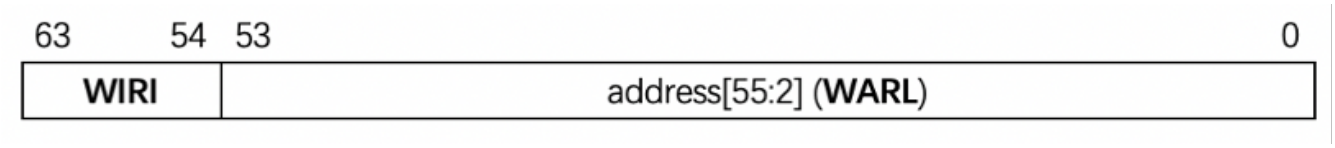

Figure 5. MPU address register format, RV64

The layout of MPU configuration registers is the same as PMP configuration registers, as is shown in Figure 6. The R, W, and X bits, when set, indicate that the MPU entry permits read, write and instruction execution, respectively. When one of these bits is clear, the corresponding access type is denied.

The S bit represents whether an MPU entry is for S-mode. When an MPU entry is for U-mode (S = 0), the permission will be enforced in U-mode and restrict the access by S-mode (for SMAP and SMEP); otherwise (S = 1), the MPU entry is for S-mode and the permission checking is enforced in S-mode. MPU also introduces shared data and code regions. The encoding for permission is shown in section 2.3, which is the same as ePMP (except MPU does not need a lockdown bit).

The remaining field, A bit, will be described in the following sections (2.2).

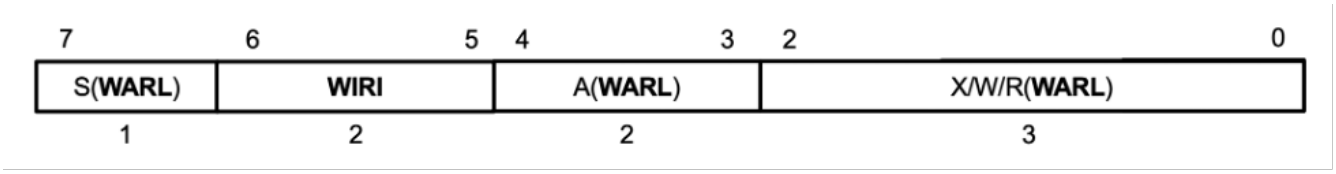

Figure 6. MPU configuration register format

The number of MPU entries: The proposal advocates 16 MPU entries, which can provide 16 isolated regions concurrently. To provide more isolation regions, the software in S-mode (usually an OS) can virtualize more isolated regions and schedule them by switching the values in MPU entries.

The reset state: The reset state of all MPU CSRs should be zero.

2.3. Address Matching

Like PMP's design, the A field in an MPU entry's configuration register encodes the address-matching mode of the associated MPU address register. The encoding of A field is the same as PMP's, as shown in Table 1. When A=0, this MPU entry is disabled and matches no address. Two other address-matching modes are supported: naturally aligned power-of-2 regions (NAPOT), including the special case of naturally aligned four-byte regions (NA4); and the top boundary of an arbitrary range (TOR). These modes support four-byte granularity.

A	Name	Description
0	OFF	Null Region (turned off)
1	TOR	Top of Range
2	NA4	Naturally aligned 4-byte region
3	NAPOT	Naturally aligned power-of-two region, >= 8 bytes

Figure 7. Encoding of A field in MPU configuration registers

NAPOT ranges make use of the low-order bits of the associated address register to encode the size of the range, as shown in Table 2.

mpuaddr	mpucfg.A	Match type and size
aaaa…aaaa	NA4	4-byte NAPOT range
aaaa…aaa0	NAPOT	8-byte NAPOT range
…	…	…
aa01…1111	NAPOT	2^{XLEN}-byte NAPOT range
a011…1111	NAPOT	2^{XLEN+1}-byte NAPOT range
0111…1111	NAPOT	2^{XLEN+2}-byte NAPOT range

Figure 8. NAPOT range encoding in MPU address and configuration registers

If TOR is selected, the associated address register forms the top of the address range, and the preceding MPU register forms the bottom of the address range. If MPU entry i's A field is set to TOR, the entry matches any address such that mpuaddr i-1 ≤ a < mpuaddr i. If MPU entry 0's A field is set to TOR, zero is used for the lower bound, and so it matches any address a < mpuaddr0.

2.4. Encoding of Permissions

MPU has three kinds of regions: U-mode region, S-mode region, and shared regions. Normally, when the S bit is set, the region could be either a S-mode region or a shared region. When the S bit is clear, the region could be either a U-mode region or a shared region.

S-mode regions indicate that the R/W/X permissions are enforced on S-mode accesses. M-mode accesses are not affected and U-mode access will trigger faults. For U-mode regions, any S-mode access matching the MPU entry

will trigger faults; the R/W/X permissions apply only to U modes. For shared regions, the permissions are enforced for both U and S modes, according to the table below.

S-mode can always modify the value in S bit.

The encoding and results are shown in the table:

Bits on *mpucfg* register				**Result**		
S	R	W	X	S Mode		U Mode
				SUM=0	SUM=1	SUM=0/1
0	0	0	0	Inaccessible region (Access Exception)		
0	0	0	1	Access Exception		Execute-only region
0	0	1	0	Shared data region: Read/write on S mode, read-only on U mode		
0	0	1	1	Shared data region: Read/write for both S and U mode		
0	1	0	0	Access Exception	Read-only region	Read-only region
0	1	0	1	Access Exception	Read-only region	Read/Execute region
0	1	1	0	Access Exception	Read/Write region	Read/Write region
0	1	1	1	Access Exception	Read/Write region	Read/Write/Execute region
1	0	0	0	Reserved		
1	0	0	1	Execute-only region		Access Exception
1	0	1	0	Shared code region: Execute only on both S and U mode		
1	0	1	1	Shared code region: Execute only on U mode, read/execute on S mode		
1	1	0	0	Read-only region		Access Exception
1	1	0	1	Read/Execute region		Access Exception
1	1	1	0	Read/Write region		Access Exception
1	1	1	1	Shared data region: Read only on both S and U mode		

SUM bit: We re-use the SUM (permit Supervisor User Memory access) bit in sstatus to modify the privilege with which S-mode loads and stores access physical memory. When SUM=0, S-mode memory accesses to U-mode regions will trigger fault, as shown in the table. When SUM=1, these accesses are permitted. The semantics of SUM in MPU is consistent with it in paging. SUM only affects S-mode behaviors on U-mode regions. It does not affect S-mode regions and shared regions. If the MPU feature is implemented, SUM must be writable. (SUM is hardwired to 0 if paging isn't implemented in current priv spec.)

MPU does not permit supervisor mode to execute instructions from U-mode regions, regardless of the SUM setting.

2.5. Priority and Matching Logic

The PMP checks only take effect after the memory access passes the MPU permission checks. An M-mode access will not be checked by MPU property.

Like PMP entries, MPU entries are also statically prioritized. The lowest-numbered MPU entry that matches any byte of an access determines whether that access succeeds or fails. The matching MPU entry must match all bytes of an access, or the access fails, irrespective of the S, R, W, and X bits.

1. If the privilege mode of the access is M, the access succeeds;
2. If the privilege mode of the access is S and no MPU entry matches, the access succeeds;
3. If the privilege mode of the access is U and no MPU entry matches, but at least one MPU entry is implemented, the access fails;
4. Otherwise, the access is checked according to the permission bits in the matching MPU entry and succeeds only if it satisfies the permission checking with the S, R, W, or X bit corresponding to the access type.

2.6. MPU and Paging

The table below shows which mechanism to use. (Assume both MMU and MPU are implemented.)

Value in satp	Isolation mechanism
0 (bare mode)	MPU only
non-zero	MMU only

We do not enable both MPU and MMU now because: (1) It will introduce one more layer to check permission for each memory access. This issue will be more serious for guest OS which may have host MPU and guest MPU. (2) MMU can provide sufficient protection.

That means, MPU is enabled when satp.mode=Bare and MPU is implemented.

If page-based virtual memory is not implemented, or when it is disabled, memory accesses check the MPU settings synchronously, so no fence is needed.

2.7. Exceptions

Failed accesses generate an exception. MPU follows the strategy that uses different exception codes for different cases, i.e., load, store/AMO, instruction faults for memory load, memory store/AMO and instruction fetch respectively.

The MPU reuses exception codes of page fault for MPU fault. This is because page fault is typically delegated to S-mode, and so does MPU, so we can benefit from reusing page fault. S-mode software(i.e., OS) can distinguish page fault from MPU fault by checking satp.mode (as mentioned in 2.6, MPU and MMU will not be activated simultaneously). The **MPU is proposing to rename page fault to MPU/MMU fault for clarity**.

Note that a single instruction may generate multiple accesses, which may not be mutually atomic.

Table of renamed exception codes:

Interrupt	Exception Code	Description
0	12	Instruction MPU/MMU fault
0	13	Load MPU/MMU fault
0	15	Store/AMO MPU/MMU fault

You can refer to the Table 3.6 in riscv-privileged spec.

Delegation: Unlike PMP which uses access faults for violations, MPU uses MPU/MMU faults for violations. The benefit of using MPU/MMU faults is that we can delegate the violations caused by MPU to S-mode, while the access violations caused by PMP can still be handled by machine mode.

2.8. Context Switching Optimization

With MPU, each context switch requires the OS to store 16 address registers and 2 configuration registers (RV64), which is costly and unnecessary. So the MPU is proposing an optimization to minimize the overhead caused by context switching.

We add one CSR called ***mpuswitch***, which is an XLEN-bit read/write register, formatted as shown in Figure 7. The low 16 bits of this register holds on/off status of the corresponding MPU region respectively. The high 48 bits is reserved for future use (e.g., extending to more MPU regions). During context switch, the OS can simply store and restore mpuswitch as part of the context. A region is activated only when both corresponding bits in mpuswitch and A field of mpuicfg are set. (i.e., mpuswitch[i] & mpu[i]cfg.A)

Figure 9. MPU domain switch register format

Chapter 3. Summary of Hardware Changes

Item	Changes
CSRs for MPU address	16 new CSRs
CSRs for MPU configuration	4 new CSRs for RV32 and 2 for RV64
CSR for Domain switch	1 new CSR
Renamed exception code	***Instruction page fault*** renamed to ***Instruction MPU/MMU fault*** ***Load page fault*** renamed to ***Load MPU/MMU fault*** ***Store/AMO page fault*** renamed to ***Store/AMO MPU/MMU fault***

Chapter 4. Interaction with other proposals

This section discusses how MPU interacts with other (ongoing) proposals.

RISC-V PMP enhancements: MPU is compatible with ePMP proposal, and uses almost the same encoding as ePMP.

J-extension pointer masking proposal: When both PM and MPU are used, MPU checking should be performed using the actual addresses generated by PM (pointer masking).

Hypervisor extension: To support both MPU and hypervisor extension, there are some further changes.

1. vMPU extension: MPU used in a guest
    - First, a set of vMPU CSRs for the VS-mode are required, including 16 vmpu address registers and 4 configuration registers. When V=1, vmpu CSR substitutes for the usual mpu CSR, so instructions that normally read or modify mpu CSR actually access vmpu CSR instead. This is consistent with the paging in VS-mode (i.e., vsatp).
    - Second, for HLV, HLVX, and HSV instructions, the hardware should perform vMPU checking before G-stage address translation (or hgMPU protection when hgatp in BARE mode).
    - Last, the vMPU checking is performed in the guest physical addresses, before G-stage address translation (or hgMPU protection when hgatp in BARE mode).
2. hgMPU extension: MPU for protecting a hypervisor from guests (only enabled when hgatp is set to BARE mode)
    - First, when hgMPU is enabled, all guest memory accesses will be checked by hgMPU; while hypervisor (in HS mode) and HU mode applications will not be affected.
    - Second, a set of hgMPU CSRs for the HS-mode are required, including 16 hgmpuaddr address registers and 4 hgmpucfg configuration registers. When V=1, and hgatp.MODE=Bare, hgmpu is used to provide isolation between hypervisor and guest VMs.
    - Third, an XLEN-bit read/write CSR hgmpuswitch is also provided in hgMPU, which is identical to mpuswitch shown in Figure 7. During context switch, the hypervisor can simply store and restore hgmpuswitch as part of the context. A region is activated only when both corresponding bits in hgmpuswitch and A field of hgmpuicfg are set. (i.e., hgmpuswitch[i] & hgmpuicfg.A)
    - Last, the hgMPU checking is performed after the guest address translation (or vMPU checking), before PMP checking.

As hgMPU does not apply on hypervisor, the encodings of configuration registers are simplified as the following table.

The encodings of hgmpucfg are shown in the table:

Bits on *hgmpucfg* register				Result
S	R	W	X	V Mode (VS + VU)
0	0	0	0	Inaccessible region (Access Exception)
0	0	0	1	Execute-only region
0	1	0	0	Read-only region
0	1	0	1	Read/Execute region

Bits on *hgmpucfg* register				Result
0	1	1	0	Read/Write region
0	1	1	1	Read/Write/Execute region
Others				Reserved