

# Proposal of MPU: RISC-V Memory Protection Unit

Dong Du, Xu Lu, Bicheng Yang, Wenhao Li, Yubin Xia, Shanghai Jiao Tong University and RISC-V TEE Task Group

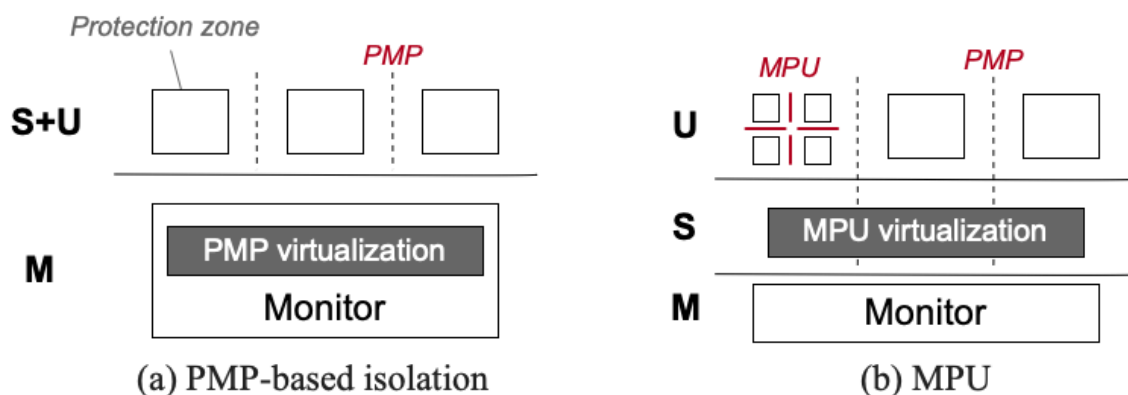
**Contributors:** Nick Kossifidis, Joe Xie, Paul Ku, Bill Huffman, Jonathan Behrens, Allen Baum, Robin Zheng, Zeyu Mi, Members of the TEE Task Groups

**Note:** The doc is archived and you can refer to RISC-V TEE's github for the latest spec (<https://github.com/riscv/riscv-tee/blob/main/Ssmpu/Ssmpu.pdf>).

## 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. MPU can provide isolation in M-S-U when MMU is not available.

## 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. M-mode accesses are not affected. 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

- a) S mode should be implemented

### 2.2. Memory Protection Unit CSRs

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 64 MPU entries are supported in standard.

The MPU configuration registers are packed into CSRs in the same way as PMP does. For RV32, 16 CSRs, `mpucfg0`-`mpucfg15`, hold the configurations `mpu0cfg`-`mpu63cfg` for the 64 MPU entries. For RV64, even numbered CSRs (i.e., `mpucfg0`, `mpucfg2`, ... , `mpucfg14`) hold the configurations for the 64 MPU entries; odd numbered CSRs (e.g., `mpucfg1`) are illegal. Figure 2 and 3 demonstrate the first 16 entries of MPU, the layout of rest entries is similar.



Figure 2: RV32 MPU configuration CSR layout

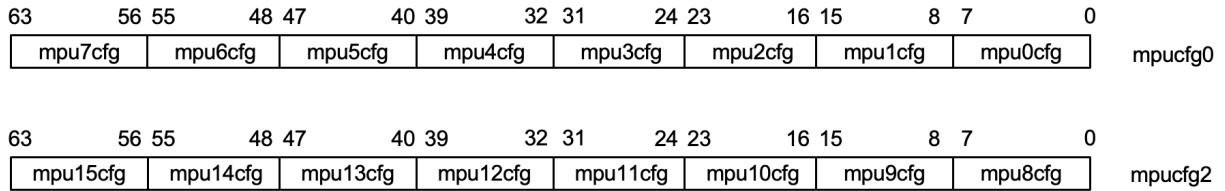


Figure 3: RV64 MPU configuration CSR layout

The MPU address registers are CSRs named  $\text{mpuaddr}_0$ - $\text{mpuaddr}_{63}$ . 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. Fewer address bits may be implemented for systems with a smaller physical address space. Implemented address bits must be contiguous and have to go from lower to higher bits.

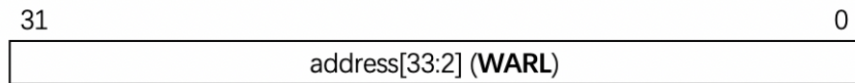


Figure 4: MPU address register format, RV32

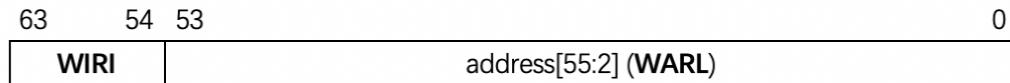


Figure 5: MPU address register format, RV64

The layout of MPU configuration registers is similar to PMP configuration registers, as is shown in Figure 6. The whole register is WARL.

The S bit marks a rule as *S-mode-only* when set and *U-mode-only* when unset. The reserved encoding of *mpucfg.RW=01*, and the encoding *mpucfg.SRWX=1111*, now encode a *Shared-Region*. The rules and encodings for permission are explained in section 2.4, which resembles the encoding of ePMP (except MPU doesn't use locked rules).

Bit 5 and 6 are reserved for future use.

The A bit will be described in the following sections (2.3).

The X/W/R bits control instruction execution, write and read permissions.

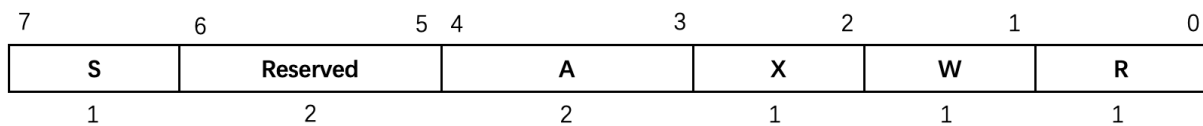


Figure 6: MPU configuration register format

**The number of MPU entries:** The proposal advocates 64 MPU entries, which can provide 64 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:** On system reset, the A field of *mpu\_cfg* should be zero.

## 2.3. Address Matching

The A field in an MPU entry's configuration register encodes the address-matching mode of the associated MPU address register. It is the same as PMP/ePMP.

## 2.4. Encoding of Permissions

MPU has three kinds of rules: U-mode-only, S-mode-only, and Shared-Region rules. The S bit marks a rule as S-mode-only when set and U-mode-only when unset. The encoding *mpucfg.SRWX=1000* is reserved and the encoding *mpucfg.RW=01* and the encoding *mpucfg.SRWX=1111* encode a Shared-Region.

An *S-mode-only* rule is *enforced* on Supervisor mode and *denied* on User mode.

A *U-mode-only* rule is *enforced* on User modes and *denied/enforced* on Supervisor mode depending on the value of *sstatus.SUM* bit:

1. If sstatus.SUM is set, a *U-mode-only* rule is *enforced* yet non-executable on Supervisor mode in order to ensure SMEP.
2. If sstatus.SUM is unset, a *U-mode-only* rule is *denied* on Supervisor mode.

A *Shared-Region* rule is *enforced* on both Supervisor and User modes, with restrictions depending on the mpucfg.S and mpucfg.X bits:

1. If mpucfg.S is not set the region can be used for sharing data between S-mode and U-mode so is not executable. S-mode has RW access to that region and U-mode has read access if mpucfg.X is not set, or RW access if mpucfg.X is set.
2. If mpucfg.S is set the region can be used for sharing code between S-mode and U-mode so is not writeable. Both S-mode and U-mode have execute access on the region and S-mode may also have read access if mpucfg.X is set.
3. The encoding mpucfg.SRWX=1111 can be used for sharing data between S-mode and U mode, where both modes only have read-only access to the region.

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 modes		
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 sstatus.SUM (permit Supervisor User Memory access) bit to modify the privilege with which S-mode loads and stores access physical memory. The semantics of SUM in MPU is consistent with it in paging.

## 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 is allowed 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 is allowed. If the privilege mode of the access is S and no MPU entry matches, the access is allowed; 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.
2. Otherwise, the access is checked according to the permission bits in the matching MPU entry and is allowed 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.**

**Note:** 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 MMU 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

Note: 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 64 address registers and 8 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 two CSRs called *mpuswitch0* and *mpuswitch1*, which are XLEN-bit read/write registers, formatted as shown in Figure 7. For RV64, only *mpuswitch0* is used. Each bit of this register holds on/off status of the corresponding MPU entry respectively. During context switch, the OS can simply store and restore *mpuswitch* as part of the context. An MPU entry is activated only when both corresponding bits in *mpuswitch* and *A field* of *mpu.cfg* are set. (i.e.,  $mpuswitch[i] \& mpu.cfg.A$ )



Figure 7: MPU domain switch register format (RV64)

## 3. Summary of Hardware Changes

<b>CSRs for MPU address:</b>	64 new CSRs
<b>CSRs for MPU configuration:</b>	16 new CSRs for RV32 and 8 for RV64
<b>Renamed exception code:</b>	Instruction page fault -> Instruction MPU/MMU fault Load page fault -> Load MPU/MMU fault Store/AMO page fault -> Store/AMO MPU/MMU fault
<b>CSR for Domain switch:</b>	2 new CSRs for RV32 and 1 for RV64

## 4. Interaction with hypervisor extension

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



## 4.1 vMPU extension

*This extension describes how MPU is used in a guest VM.*

1. A set of vMPU CSRs for the VS-mode are required, including 64 vmpu address registers and 16 configuration registers(RV32). 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).
2. 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).
3. The vMPU checking is performed in the guest physical addresses, before G-stage address translation (or hgMPU protection when hgatp in BARE mode).

## 4.2 hgMPU extension

This extension describes how MPU is used for protecting a hypervisor from guests (only enabled when hgatp is set to BARE mode).

1. 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.
2. A set of hgMPU CSRs for the HS-mode are required, including 64 *hgmpuaddr* address registers and 16 *hgmpucfg* configuration registers (RV32). When V=1, and *hgatp.MODE=Bare*, *hgmpu* is used to provide isolation between hypervisor and guest VMs.
3. XLEN-bit read/write *hgmpuswitch0* and *hgmpuswitch1* CSRs are also provided in hgMPU, which are identical to *mpuswitch0* and *mpuswitch1* shown in Figure 7. Only *hgmpuswitch0* is used for RV64. During context switch, the hypervisor can simply store and restore *hgmpuswitch* (we use *hgmpuswitch* to represent either *hgmpuswitch0* or *hgmpuswitch1*) as part of the context. An hgMPU entry is activated only when both corresponding bits in *hgmpuswitch* and *A* field of *hgmpuicfg* are set. (i.e., *hgmpuswitch[i]* & *hgmpuicfg.A*)
4. 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 <i>hgmpucfg</i> 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
0	1	1	0	Read/Write region
0	1	1	1	Read/Write/Execute region
Others				Reserved

## 5. Interaction with other proposals

This section discusses how MPU interacts with other 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).