While hardware should always be designed to avoid speculative or redundant accesses to memory regions marked as non-idempotent, it is also necessary to ensure software or compiler optimizations do not generate spurious accesses to non-idempotent memory regions.

Non-idempotent regions might not support misaligned accesses. Misaligned accesses to such regions should raise access-fault exceptions rather than address-misaligned exceptions, indicating that software should not emulate the misaligned access using multiple smaller accesses, which could cause unexpected side effects.

For non-idempotent regions, implicit reads and writes must not be performed early or speculatively, with the following exceptions. When a non-speculative implicit read is performed, an implementation is permitted to additionally read any of the bytes within a naturally aligned power-of-2 region containing the address of the non-speculative implicit read. Furthermore, when a non-speculative instruction fetch is performed, an implementation is permitted to additionally read any of the bytes within the *next* naturally aligned power-of-2 region of the same size (with the address of the region taken modulo 2^{XLEN}). The results of these additional reads may be used to satisfy subsequent early or speculative implicit reads. The size of these naturally aligned power-of-2 regions is implementation-defined, but, for systems with page-based virtual memory, must not exceed the smallest supported page size.

3.7 Physical Memory Protection

To support secure processing and contain faults, it is desirable to limit the physical addresses accessible by software running on a hart. An optional physical memory protection (PMP) unit provides per-hart machine-mode control registers to allow physical memory access privileges (read, write, execute) to be specified for each physical memory region. The PMP values are checked in parallel with the PMA checks described in Section 3.6.

The granularity of PMP access control settings are platform-specific, but the standard PMP encoding supports regions as small as four bytes. Certain regions' privileges can be hardwired—for example, some regions might only ever be visible in machine mode but in no lower-privilege layers.

Platforms vary widely in demands for physical memory protection, and some platforms may provide other PMP structures in addition to or instead of the scheme described in this section.

PMP checks are applied to all accesses whose effective privilege mode is S or U, including instruction fetches in S and U mode, data accesses in S and U mode when the MPRV bit in the `mstatus` register is clear, and data accesses in any mode when the MPRV bit in `mstatus` is set and the MPP field in `mstatus` contains S or U. PMP checks are also applied to page-table accesses for virtual-address translation, for which the effective privilege mode is S. Optionally, PMP checks may additionally apply to M-mode accesses, in which case the PMP registers themselves are locked, so that even M-mode software cannot change them until the hart is reset. In effect, PMP can *grant* permissions to S and U modes, which by default have none, and can *revoke* permissions from M-mode, which by default has full permissions.

PMP violations are always trapped precisely at the processor.

3.7.1 Physical Memory Protection CSRs

PMP entries are described by an 8-bit configuration register and one MXLEN-bit address register. Some PMP settings additionally use the address register associated with the preceding PMP entry. Up to 64 PMP entries are supported. Implementations may implement zero, 16, or 64 PMP CSRs; the lowest-numbered PMP CSRs must be implemented first. All PMP CSR fields are **WARL** and may be read-only zero. PMP CSRs are only accessible to M-mode.

The PMP configuration registers are densely packed into CSRs to minimize context-switch time. For RV32, sixteen CSRs, `pmpcfg0`–`pmpcfg15`, hold the configurations `pmp0cfg`–`pmp63cfg` for the 64 PMP entries, as shown in Figure 3.31. For RV64, eight even-numbered CSRs, `pmpcfg0`, `pmpcfg2`, ..., `pmpcfg14`, hold the configurations for the 64 PMP entries, as shown in Figure 3.32. For RV64, the odd-numbered configuration registers, `pmpcfg1`, `pmpcfg3`, ..., `pmpcfg15`, are illegal.

RV64 systems use `pmpcfg2`*, rather than* `pmpcfg1`*, to hold configurations for PMP entries 8–15. This design reduces the cost of supporting multiple MXLEN values, since the configurations for PMP entries 8–11 appear in* `pmpcfg2`*[31:0] for both RV32 and RV64.*

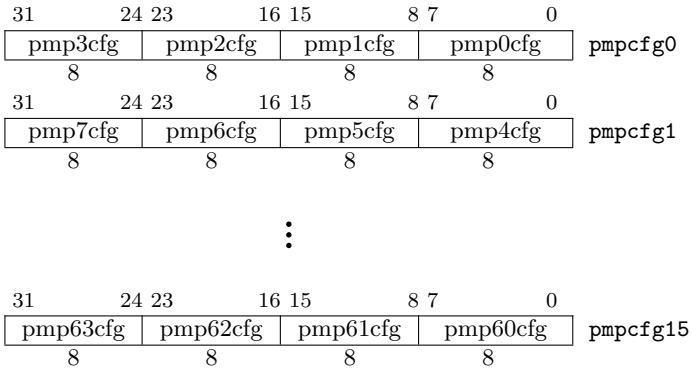

Figure 3.31: RV32 PMP configuration CSR layout.

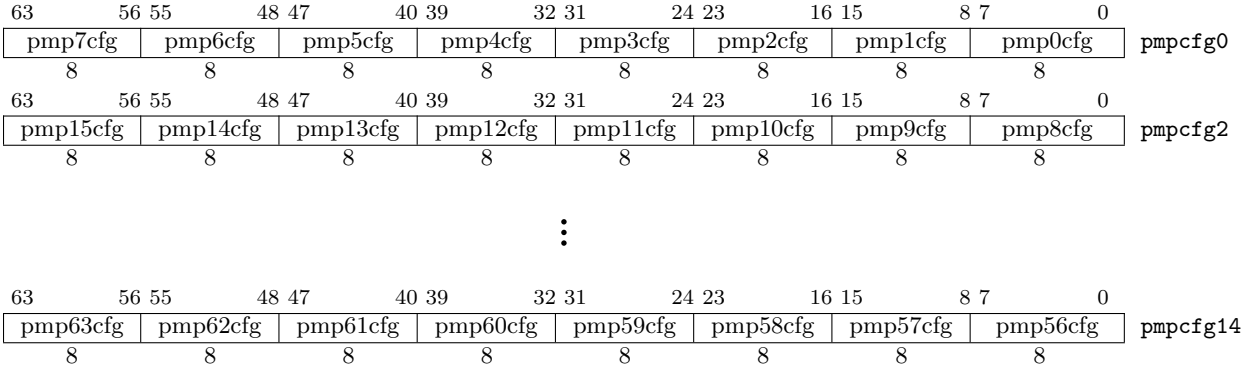

Figure 3.32: RV64 PMP configuration CSR layout.

The PMP address registers are CSRs named `pmpaddr0`–`pmpaddr63`. Each PMP address register encodes bits 33–2 of a 34-bit physical address for RV32, as shown in Figure 3.33. For RV64, each PMP address register encodes bits 55–2 of a 56-bit physical address, as shown in Figure 3.34. Not all physical address bits may be implemented, and so the `pmpaddr` registers are **WARL**.

The Sv32 page-based virtual-memory scheme described in Section 4.3 supports 34-bit physical addresses for RV32, so the PMP scheme must support addresses wider than XLEN for RV32. The Sv39 and Sv48 page-based virtual-memory schemes described in Sections 4.4 and 4.5 support a 56-bit physical address space, so the RV64 PMP address registers impose the same limit.

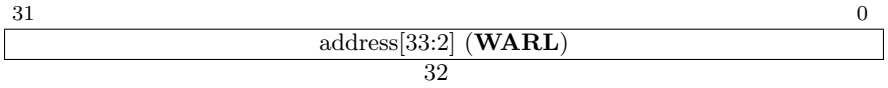

Figure 3.33: PMP address register format, RV32.

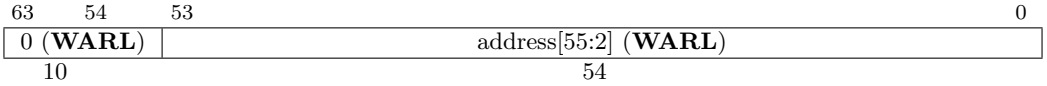

Figure 3.34: PMP address register format, RV64.

Figure 3.35 shows the layout of a PMP configuration register. The R, W, and X bits, when set, indicate that the PMP entry permits read, write, and instruction execution, respectively. When one of these bits is clear, the corresponding access type is denied. The R, W, and X fields form a collective **WARL** field for which the combinations with R=0 and W=1 are reserved. The remaining two fields, A and L, are described in the following sections.

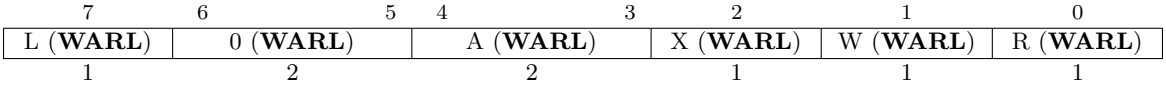

Figure 3.35: PMP configuration register format.

Attempting to fetch an instruction from a PMP region that does not have execute permissions raises an instruction access-fault exception. Attempting to execute a load or load-reserved instruction which accesses a physical address within a PMP region without read permissions raises a load access-fault exception. Attempting to execute a store, store-conditional, or AMO instruction which accesses a physical address within a PMP region without write permissions raises a store access-fault exception.

If MXLEN is changed, the contents of the `pmpxcfg` fields are preserved, but appear in the `pmpcfgy` CSR prescribed by the new setting of MXLEN. For example, when MXLEN is changed from 64 to 32, `pmp4cfg` moves from `pmpcfg0`[39:32] to `pmpcfg1`[7:0]. The `pmpaddr` CSRs follow the usual CSR width modulation rules described in Section 2.6.

Address Matching

The A field in a PMP entry's configuration register encodes the address-matching mode of the associated PMP address register. The encoding of this field is shown in Table 3.10. When A=0, this PMP entry is disabled and matches no addresses. 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 (disabled)
1	TOR	Top of range
2	NA4	Naturally aligned four-byte region
3	NAPOT	Naturally aligned power-of-two region, ≥8 bytes

Table 3.10: Encoding of A field in PMP 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 3.11.

`pmpaddr`	`pmpcfg`.A	Match type and size
`yyyy...yyyy`	NA4	4-byte NAPOT range
`yyyy...yyy0`	NAPOT	8-byte NAPOT range
`yyyy...yy01`	NAPOT	16-byte NAPOT range
`yyyy...y011`	NAPOT	32-byte NAPOT range
...	...	...
`yy01...1111`	NAPOT	2^{XLEN}-byte NAPOT range
`y011...1111`	NAPOT	$2^{\text{XLEN}+1}$-byte NAPOT range
`0111...1111`	NAPOT	$2^{\text{XLEN}+2}$-byte NAPOT range
`1111...1111`	NAPOT	$2^{\text{XLEN}+3}$-byte NAPOT range

Table 3.11: NAPOT range encoding in PMP address and configuration registers.

If TOR is selected, the associated address register forms the top of the address range, and the preceding PMP address register forms the bottom of the address range. If PMP entry i's A field is set to TOR, the entry matches any address y such that $\texttt{pmpaddr}_{i-1} \leq y < \texttt{pmpaddr}_i$ (irrespective of the value of $\texttt{pmpcfg}_{i-1}$). If PMP entry 0's A field is set to TOR, zero is used for the lower bound, and so it matches any address $y < \texttt{pmpaddr}_0$.

If $\texttt{pmpaddr}_{i-1} \geq \texttt{pmpaddr}_i$ *and* $\texttt{pmpcfg}_i\texttt{.A}$*=TOR, then PMP entry i matches no addresses.*

Although the PMP mechanism supports regions as small as four bytes, platforms may specify coarser PMP regions. In general, the PMP grain is 2^{G+2} bytes and must be the same across all PMP regions. When $G \geq 1$, the NA4 mode is not selectable. When $G \geq 2$ and $\texttt{pmpcfg}_i$.A[1] is set, i.e. the mode is NAPOT, then bits $\texttt{pmpaddr}_i$[G-2:0] read as all ones. When $G \geq 1$ and $\texttt{pmpcfg}_i$.A[1] is clear, i.e. the mode is OFF or TOR, then bits $\texttt{pmpaddr}_i$[G-1:0] read as all zeros. Bits $\texttt{pmpaddr}_i$[G-1:0] do not affect the TOR address-matching logic. Although changing $\texttt{pmpcfg}_i$.A[1] affects the value read from $\texttt{pmpaddr}_i$, it does not affect the underlying value stored in that register—in particular, $\texttt{pmpaddr}_i$[G-1] retains its original value when $\texttt{pmpcfg}_i$.A is changed from NAPOT to TOR/OFF then back to NAPOT.

Software may determine the PMP granularity by writing zero to `pmp0cfg`*, then writing all ones to* `pmpaddr0`*, then reading back* `pmpaddr0`*. If G is the index of the least-significant bit set, the PMP granularity is* 2^{G+2} *bytes.*

If the current XLEN is greater than MXLEN, the PMP address registers are zero-extended from MXLEN to XLEN bits for the purposes of address matching.

Locking and Privilege Mode

The L bit indicates that the PMP entry is locked, i.e., writes to the configuration register and associated address registers are ignored. Locked PMP entries remain locked until the hart is reset. If PMP entry i is locked, writes to `pmp`i`cfg` and `pmpaddr`i are ignored. Additionally, if PMP entry i is locked and `pmp`i`cfg`.A is set to TOR, writes to `pmpaddr`i-1 are ignored.

Setting the L bit locks the PMP entry even when the A field is set to OFF.

In addition to locking the PMP entry, the L bit indicates whether the R/W/X permissions are enforced on M-mode accesses. When the L bit is set, these permissions are enforced for all privilege modes. When the L bit is clear, any M-mode access matching the PMP entry will succeed; the R/W/X permissions apply only to S and U modes.

Priority and Matching Logic

PMP entries are statically prioritized. The lowest-numbered PMP entry that matches any byte of an access determines whether that access succeeds or fails. The matching PMP entry must match all bytes of an access, or the access fails, irrespective of the L, R, W, and X bits. For example, if a PMP entry is configured to match the four-byte range `0xC`–`0xF`, then an 8-byte access to the range `0x8`–`0xF` will fail, assuming that PMP entry is the highest-priority entry that matches those addresses.

If a PMP entry matches all bytes of an access, then the L, R, W, and X bits determine whether the access succeeds or fails. If the L bit is clear and the privilege mode of the access is M, the access succeeds. Otherwise, if the L bit is set or the privilege mode of the access is S or U, then the access succeeds only if the R, W, or X bit corresponding to the access type is set.

If no PMP entry matches an M-mode access, the access succeeds. If no PMP entry matches an S-mode or U-mode access, but at least one PMP entry is implemented, the access fails.

If at least one PMP entry is implemented, but all PMP entries' A fields are set to OFF, then all S-mode and U-mode memory accesses will fail.

Failed accesses generate an instruction, load, or store access-fault exception. Note that a single instruction may generate multiple accesses, which may not be mutually atomic. An access-fault exception is generated if at least one access generated by an instruction fails, though other accesses generated by that instruction may succeed with visible side effects. Notably, instructions that reference virtual memory are decomposed into multiple accesses.

On some implementations, misaligned loads, stores, and instruction fetches may also be decomposed into multiple accesses, some of which may succeed before an access-fault exception occurs. In particular, a portion of a misaligned store that passes the PMP check may become visible,

even if another portion fails the PMP check. The same behavior may manifest for floating-point stores wider than XLEN bits (e.g., the FSD instruction in RV32D), even when the store address is naturally aligned.

3.7.2 Physical Memory Protection and Paging

The Physical Memory Protection mechanism is designed to compose with the page-based virtual memory systems described in Chapter 4. When paging is enabled, instructions that access virtual memory may result in multiple physical-memory accesses, including implicit references to the page tables. The PMP checks apply to all of these accesses. The effective privilege mode for implicit page-table accesses is S.

Implementations with virtual memory are permitted to perform address translations speculatively and earlier than required by an explicit memory access, and are permitted to cache them in address translation cache structures—including possibly caching the identity mappings from effective address to physical address used in Bare translation modes and M-mode. The PMP settings for the resulting physical address may be checked (and possibly cached) at any point between the address translation and the explicit memory access. Hence, when the PMP settings are modified, M-mode software must synchronize the PMP settings with the virtual memory system and any PMP or address-translation caches. This is accomplished by executing an SFENCE.VMA instruction with *rs1*=`x0` and *rs2*=`x0`, after the PMP CSRs are written.

If page-based virtual memory is not implemented, memory accesses check the PMP settings synchronously, so no SFENCE.VMA is needed.