RISC-V Privileged Spec Summary

Privilege Levels

- Every RISC-V hart runs at every time on one of the three privilege levels that are encoded as a mode in one or more CSRs.

Level	Encoding	Name	Abbreviation
0	00	User/Application	U
1	01	Supervisor	S
2	10	*Reserved*	
3	11	Machine	M

- Privilege levels provide protection between different components of the software stack the attempt of execute a instruction, that is not permitted on the current privilege mode must raise an exception -> trap into an underlying execution environment
- Code run on machine-mode can be used to manage secure execution environments
- User-mode and Supervisor-mode are intended vor applications and operating system usage
- Each privilege level has a core set of privileged ISA extensions (+ optional extensions & variants)
- Implementations might provide 1-3 privilege modes:

Number of levels	Supported Modes	Intended Usage
1	M	Simple embedded systems
2	M, U	Secure embedded systems
3	M, S, U	Systems running Unix-like operating systems

- All hardware implementations must provide M-mode
- The suggested hart/core will implement only M-mode and the RV32I ISA with the Zicsr extension

Control and Status Registers (CSRs)

CSR Address Mapping Conventions

- 12-bit encoding space for up to 4096 CSRs
- In order to safe BRAM inside an FPGA, only the used CSRs must be implemented, accesses to every other must raise an „illegal instruction exception“
- Attempts to access a CSR without privilege level or to write a read-only register raises an illegal instruction exception
- read/write registers may contain read-only bits -> writes to those bits are ignored
- CSRs 0x7A0-0x7BF: reserved for debug-system -> therefore not implemented
- A Listing of allocated CSRs can be found in the privileged spec in chapter 2.2 „CSR Listing“

CSR Field Specifications

- Reserved Writes Preserve Values, Reads Ignore Values *(WPRI)*
  - Some whole read/write fields are reserved for future use
  - Software should ignore the values from this field
  - This fields should not be overwritten
  - Implementations that do not implement these fields must wire them to zero
- Write/Read Only Legal Values *(WLRL)*
  - Some read/write CSR fields specify behavior for only some encodings
  - Software should not write anything other than legal values to those fields
  - Software should not expect a legal value, except if the last write was a legal value or the register has not been written since it was set to a legal value, i.e. by a reset
  - Hardware need only enough state bits to differentiate between the supported values, but must return the complete specified bit-encoding
  - Implementations could raise an illegal instruction exception if an instruction attempts to write a non-supported value to such a filed
- Write Any Values, Reads Legal Values *(WARL)*
  - read/write CSR fields only defined for a subset of encodings
  - Guarantee to return a legal value when read
  - Allow any value to be written
  - Implementations can return any legal value on a read, if the last write was an illegal value
  - The returned value should deterministically depend on the illegal written value & the value of the field prior to the write

CSR Width Modulation

- The implementation shall not allow CSR Width Modulation!
- If the width of a CSR is changed (e.g. by changing MXLEN|UXLEN) the values of the writable fields and bits of the new-width CSR are determined from the previous-width CSR:
  - Copy value of previous-CSR to a register of same width
  - Set read-only bits of the previous-CSR in the copy register to zero
  - Change width of the temporary register to the new width
  - New width < previous width:
    - LSBs of the temp register are retained
    - MSBs of the temp register are discarded

- New width > previous width:
  - Zero-extend temp reg values
- Copy the temp reg to new-width CSR (only writable fields)

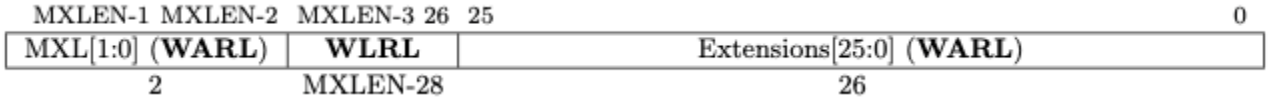
- Changing a width is no read or write of the CSR and thus does not trigger any side effects

Machine-Level ISA

- Machine-Level is the highest and only privilege-mode
- Can access als CSRs of machine-level and lower privilege level
- MXLEN shall be 32

Machine Level CSRs

Machine ISA Register ***misa***:

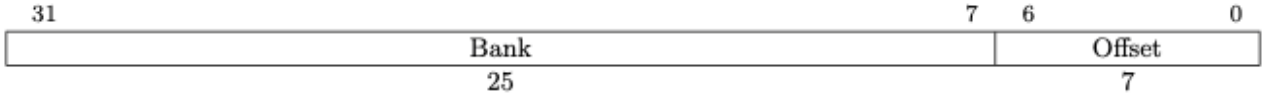


- WARL read-only register
- MXLEN bits wide
- Reports the ISA supported by the hart
- Must be implemented,
- Bits should only be writable, if the implementation allows it to change e.g. what extensions are supported
- MXL: Machine XLEN, encodes base integer ISA width must be 1 (=32)

Bit	Character	Description
0	A	Atomic extension
1	B	*Tentatively reserved for Bit-Manipulation extension*
2	C	Compressed extension
3	D	Double-precision floating-point extension
4	E	RV32E base ISA
5	F	Single-precision floating-point extension
6	G	Additional standard extensions present
7	H	Hypervisor extension
8	I	RV32I/64I/128I base ISA
9	J	*Tentatively reserved for Dynamically Translated Languages extension*
10	K	*Reserved*
11	L	*Tentatively reserved for Decimal Floating-Point extension*
12	M	Integer Multiply/Divide extension
13	N	User-level interrupts supported
14	O	*Reserved*
15	P	*Tentatively reserved for Packed-SIMD extension*
16	Q	Quad-precision floating-point extension
17	R	*Reserved*
18	S	Supervisor mode implemented
19	T	*Tentatively reserved for Transactional Memory extension*
20	U	User mode implemented
21	V	*Tentatively reserved for Vector extension*
22	W	*Reserved*
23	X	Non-standard extensions present
24	Y	*Reserved*
25	Z	*Reserved*

- Extensions: encodes the presence of standard extensions:
  - Extensions = „00b“&„0x000100“

Machine Vendor ID Register ***mvendorid***:

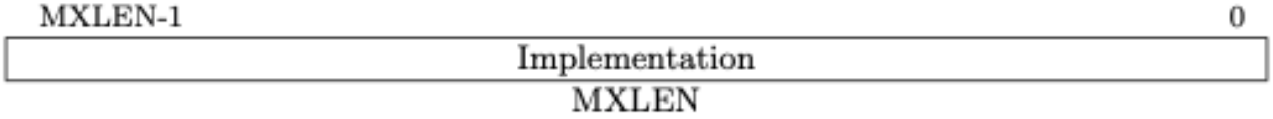


- 32-bit, read-only
- Set to zero, since the implementation is not commercial

Machine Architecture ID Register ***marchid***:

- MXLEN-bit, read-only
- Open-source project architecture IDs are allocated globally by the RISC-V Foundation
- Since there is no ID allocated for this project, it shall be implemented, but hardwired to zero (for now)

Machine Impelementation ID Register ***mimpid***:



- MXLEN-bit, read-only
- Unique encoding of the version of the processor implementation
- This is version one therefore mimpid shall return 0x00000001

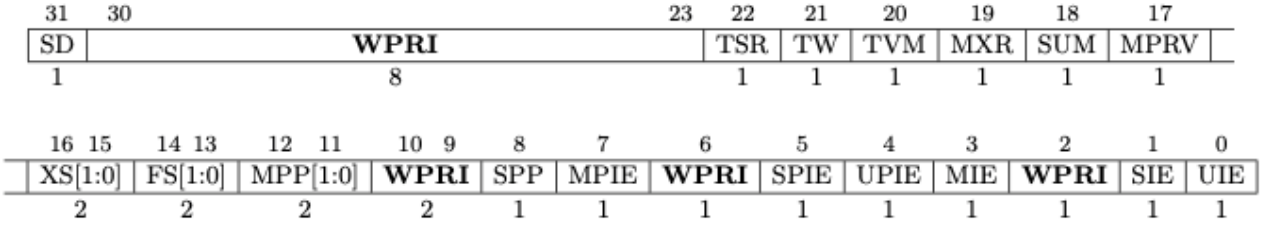
Hart ID Register ***mhartid***:

- MXLEN-bit, read-only
- Contains the integer ID of the hart running the code
- Hart IDs must be unique in a multiprocessor system, one must be zero

- Since there will be only one hart the value must hardwired to zero

---

Machine Status Register ***mstatus***:



- MXLEN-bit, read/write
- Reflects and controls the hart's current operating state

Privilege and Global Interrupt-Enable Stack in *mstatus* register:
- Global Interrupt-enable bits: MIE, SIE, UIE
- S- & U-mode are not implemented -> SIE & UIE are hardwired to zero
- Interrupts are globally enabled, if MIE = 1, disabled if MIE = 0
- Nested traps do not need to be supported since there is only one privilege level
  - MPIE = 0
  - MPP = M = „11“
- UPP = SPP = 0
- UPIE = SPIE = 0

Memory Privilege in *mstatus* register:
- MPRV bit modifies privilege level at which loads and stores execute in all privilege modes, since there is only one privilege level: MPRV = 0

- MXR: (Make eXecutable Readable) modifies the privilege with wich loads access virtual memory, since S-mode is not supported: MXR = 0

- SUM: (permit Supervisor User Memory access) modifies privilege with wich S-mode loads and stores access virtual memory, S-Mode is not supported: SUM = 0

Virtualization Support in *mstatus* register:
- TVM: (Trap Virtual Memory) supports intercepting supervisor virtual-memory management operations, S-mode is not supported: TVM = 0

- TW: (Timeout Wait) supports intercepting the WFI instructions
- Since only M-Mode is implemented, TW is hardwired to 0

- TSR: (Trap SRET) supports intercepting supervisor exception return instruction (SRET)
- Since S-mode is not supported: TSR is hardwired to 0

Extension Context Status in *mstatus* Register:

- FS[1:0] and XS[1:0] used to reduce the cost of context space and restore by tracking and setting the current state of the FPU & any other user mode extension
- Since S-mode & floating points are not implemented, the FS field is hardwired to 0
- Since no additional user extension requires new state, the XS field is hardwired to 0

- SD: read-only, summarizes whether either the FS field or XS field signals the presence of some dirty state
- Since both XS & FS are hardwired to 0, SD is hardwired to 0

Machine Trap-Vector Base-Address Register ***mtvec***:

- MXLEN-bit, read/write register
- Holds trap vector configuration (base address: BASE and vector mode: MODE)
- BASE value musst align on a 4-byte boundary
- Since no asynchronous interrupts shall be implemented, MODE must be zero -> all calls jump to Base

Machine Trap Delegation Registers ***medeleg*** and ***mideleg***:

- MXELN-bit, read/write registers
- In systems with only M-mode or M & U but without U-mode trap support, medeleg & mideleg should not exist

Machine Interrupt Registers (mip and mie)

MXLEN-1	12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI		MEIP	**WPRI**	SEIP	UEIP	MTIP	**WPRI**	STIP	UTIP	MSIP	**WPRI**	SSIP	USIP
MXLEN-12		1	1	1	1	1	1	1	1	1	1	1	1

Machine interrupt-pending register (mip)

MXLEN-1	12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI		MEIE	**WPRI**	SEIE	UEIE	MTIE	**WPRI**	STIE	UTIE	MSIE	**WPRI**	SSIE	USIE
MXLEN-12		1	1	1	1	1	1	1	1	1	1	1	1

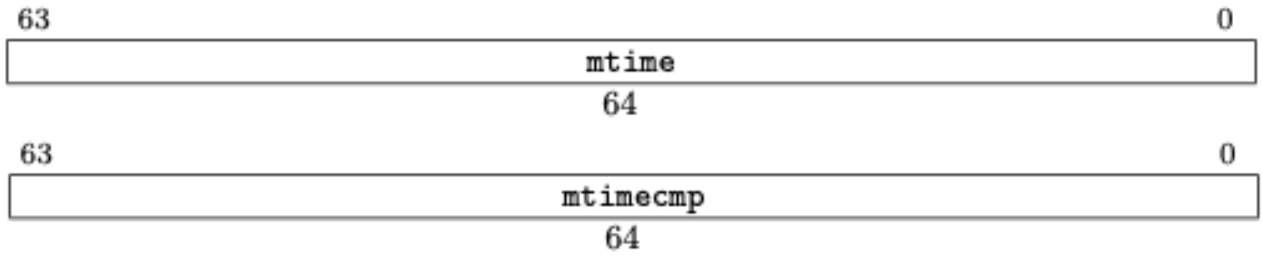
Machine interrupt-enable register (mie)

- MXLEN-bit, read/write (mie) and read-only (mip) register
- Since U- and S-mode are not implemented, all associated pending & enable bits are hardwired to 0
- An interrupt will be taken if: pending & enable are set and interrupts are globally enabled
- Interrupts are globally enabled, if MIE in mstatus is set
- Interrupt priorities:
  - Highest mode interrupt first
  - Order: MEI, MSI, MTI

mip:
- xTIP: correspond to timer interrupt-pending bits for the corresponding privilege-level
  - Hardwired to 0 for S & U
- MSIP:
  - can be read and written
  - MSIP is written by access to memory-mapped control registers, used by remote harts to provide interprocessor interrupts
  - A hart can write its own MSIP, using the same memory mapped CSR
- USIP & SSIP: not implemented -> hardwired to 0
- MEIP: read-only, indicates a machine mode external interrupt is pending, set and cleared by a platform-specific interrupt controller -> hardwired to 0, since no external interrupts are implemented
- SEIP:
  - Hardwired to 0
- UEIP: hardwired to 0

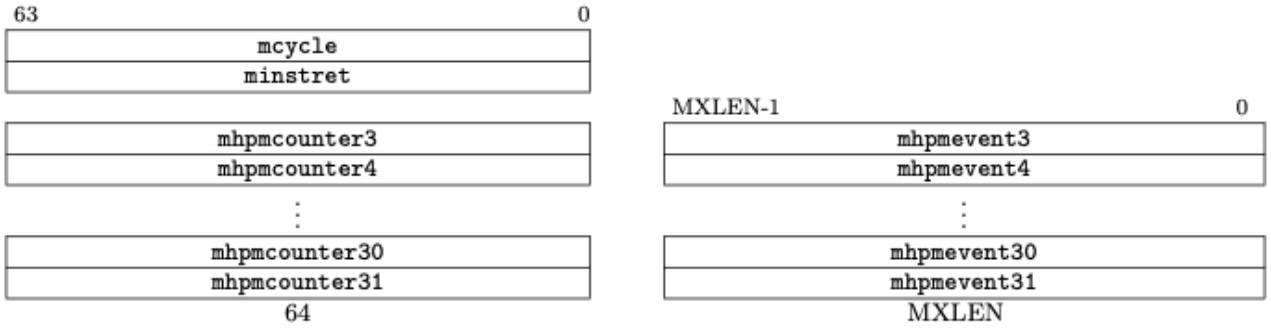
mie:
- MEIE: enables machine external interrupts, when set, hardwired to 0
- MTIE: timer interrupt enable bits for the corresponding privilege-level
- STIE & UTIE: hardwired to 0
- SEIE, UEIE: enable external interrupts for the respective mode, hardwired to 0

Machine Time Registers (***mtime*** and ***mtimecmp***)



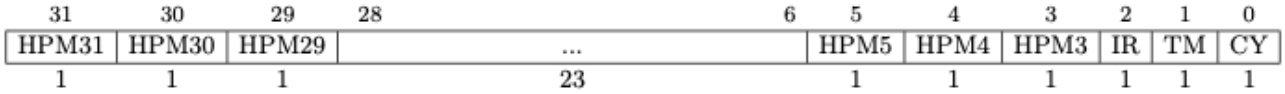
- 64-bit, read/write
- mtime must run at a constant frequency
- The platform must provide a mechanism for determining the timebase of mtime
- mtimecmp: causes timer interrupt to be posted, when grater than or equal to mtime, remains posted until mtimecmp is written

Hardware Performance Monitor



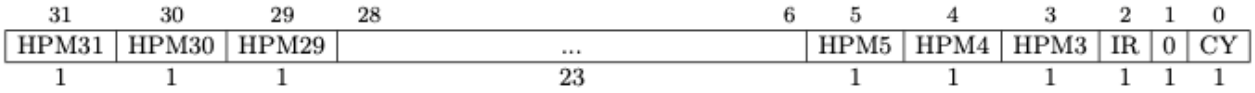
- 64-bit, read/write registers (except event: MXLEN)
- In RV32I, every 64-bit register is made out of two 32-bit registers.
- mcycle: counts the number of clock cycles executed by the processor core
- minstret: counts the number of instructions the harts has retired
- mhpmcounter3-31: incremented on the corresponding mhpevent-event
- Event 0 means no event, others are platform-specific.
- All registers should be implemented

Counter-Enable Registers (***[m|s]counteren***)



- 32-bit, read/write registers
- Control the availability of the hardware performance-monitoring centers to the next-lowest privilege mode (only if accesses are allowed, the counting happens anyways)
- Shall not be implemented

Machine Counter-Inhibit CSR (***mcountinhibit***)

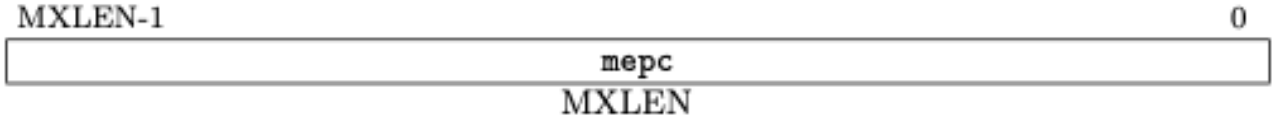


- 32-bit, WARL register
- Controls which of the hardware performance-monitoring counters increment.
- If the corresponding bit is set, the corresponding counter does not incremen

Machine Scratch Register (***mscratch***)

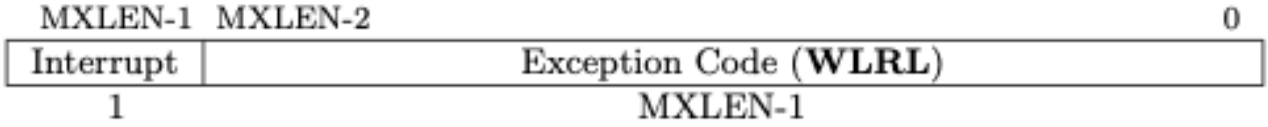
- MXLEN-bit, read/write register
- Dedicated for use by machine made

Machine Exception Program Counter (***mepc***)



- MXLEN-bit, read/write register
- Since IALIGN = 32, two LSBs are always zero
- WARL register, must be able to hold all valid physical and virtual addresses
- If a trap is taken into M-mode, mepc is written with the virtual address (beginning) of the instruction that was interrupted or that encountered the exception

Machine Cause Register (***mcause***)



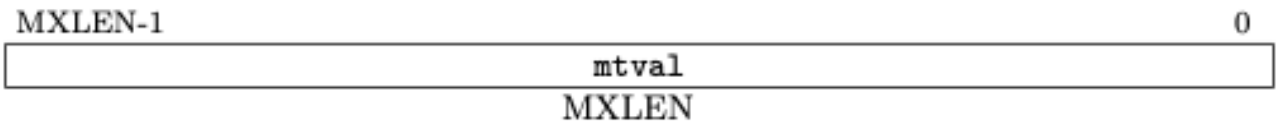
- MXLEN-bit, read/write register
- Trap taken into M-Mode: mcause is written with a code indicating the trap causing event
- Instruction raises multiple synchronous exceptions

  -> take exception in decreasing order of priority:

Priority	Exception Code	Description
Highest	3	Instruction address breakpoint
	12	Instruction page fault
	1	Instruction access fault
	2	Illegal instruction
	0	Instruction address misaligned
	8, 9, 11	Environment call
	3	Environment break
	3	Load/Store/AMO address breakpoint
	6	Store/AMO address misaligned
	4	Load address misaligned
	15	Store/AMO page fault
	13	Load page fault
	7	Store/AMO access fault
Lowest	5	Load access fault

Interrupt	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	*Reserved for future standard use*
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	*Reserved for future standard use*
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	*Reserved for future standard use*
1	11	Machine external interrupt
1	12–15	*Reserved for future standard use*
1	≥16	*Reserved for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	*Reserved*
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved for future standard use*
0	15	Store/AMO page fault
0	16–23	*Reserved for future standard use*
0	24–31	*Reserved for custom use*
0	32–47	*Reserved for future standard use*
0	48–63	*Reserved for custom use*
0	≥64	*Reserved for future standard use*

Machine Trap Value Register (***mtval***)



- MXLEN-bit, read/write register
- Trap is taken into M-mode: mtval set to zero or written with exception-specific information
- Hardware breakpoint, instruction-fetch, load or store address-misaligned access or page-fault exception -> write mtval with faulting virtual address

- Illegal instruction trap -> write mtval with the first XLEN or ILEN bits of the faulting instruction (or zero)
- Others -> set mtval to zero

Machine-Mode Privileged Instructions

Environment Call and Breakpoint

31 20	19 15	14 12	11 7	6 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
ECALL	0	PRIV	0	SYSTEM
EBREAK	0	PRIV	0	SYSTEM

- ECALL and EBREAK cause the receiving privilege mode's epc (e.g. mepc) to be set to the address of the ECALL or EBREAK instruction, not the following one
- ECALL:
  - Make a request to the supporting execution environment
  - When executed in x-Mode it creates a „environment-call-from-x-mode exception", x being U,S or M
- EBREAK:
  - Generates a breakpoint exception

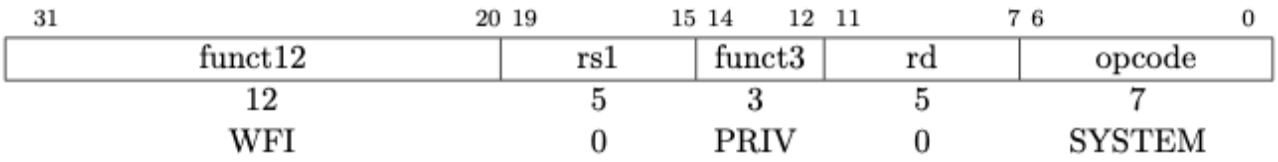
Trap-Return Instructions

31 20	19 15	14 12	11 7	6 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
MRET/SRET/URET	0	PRIV	0	SYSTEM

- xRET (x bein M, S or U)
  - Can be executed in privilege mode x or higher
  - On execution:
    - If current-mode > x:
    - xIE is set to xPIE
    - the privilege mode is changed y
    - xPIE is set to 1 and xPP is set to U (or M, if user mode is not supported)
    - Set PC to the value stored in the xepc register
    - Else:
    - Pc = xepc

- MRET
  - Always provided
- SRET
  - Not supported, raise an illegal instruction exception
- URET
  - Not supported, raise an illegal instruction exception

Wait For Interrupt



- Provides a hint to the implementation, current hart may be stalled until an interrupt might need servicing
- Raise an illegal instruction exception if TW = 0 in mstatus
- If an enabled interrupt is present or later becomes present while the hart is stalled, the interrupt exception will be taken on the following instruction

  => execution resumes in the trap handler and mepc = pc + 4
- WFI can be performed, if the interrupts are disabled
  - Operation of WFI must be unaffected from global IE (xIE) as well as ([m|s]ideleg)
  - Implementation should avoid resuming the hart, if the interrupt is pending but not individually enabled
  - WFI musst resume execution for locally enabled interrupts pending at any privilege level, regardless of xIE
  - If the event that causes the hart to resume execution does not cause an interrupt to be taken, execution will resume at pc+4

Reset

- On reset:
  - Set harts privilege Mode to M
  - Set mstatus.MIE and mstatus.MPRV to 0
  - Set misa to support maximal set of supported extension and widest MXLEN
  - Reset PC 0x00000000
  - Set mcause to 0x00000000
  - Set writable PMP registers' A and L fields to 0

Non-Maskable Interrupts

- Only used fro hardware error conditions
- Cause an immediate jump to an implementation-defined NMI vector running in M-mode, regardless of the interrupt enable bits.
- mepc: written with the address of the next instruction
- mcause: write value to indicate source of the NMI (0 = unknown cause)

Physical Memory Attributes

- Physical memory map:
  - Various address ranges (memory regions, memory-mapped control registers and vacant space)
  - Some regions may support reads, writes or/and execution
  - Some might not support subword or subblock access
  - Some might not support atomic operations
  - Some might not support cache coherence
  - Some might use different memory models

=> PMAs (Phyiscal Memory Attributes)

- RISC-V: specification and checking of PMAs in separate hardware structure: *PMA checker*
  - platform-specific memory-mapped control registers to specify these attributes (use the ones from PMP module)
  - PMAs are checked for any access to physical memory
  - Precisely trap physical memory accesses that fail PMA checks (load, store or instruction-fetch access exceptions & distinct from virtual-memory page-fault exceptions)
  - Error responses from slave devices will be reported as imprecise bus-error interrupts
  - PMAs must be readable by software
  - If dynamic reconfigurable PMAs are implemented, an interface will be provided to set the attributes by passing requests to a machine-mode driver -> can correctly reconfigure the platform (e.g. cash flushes)

Main Memory versus I/O versus Empty Regions

- Characterization of a region:
  - Regular main memory, I/O devices or empty?
  - Main memory needs a number of poperties

- I/O has a lot more properties
- Memory regions that do not fit into regular main memory are I/O regions
- Empty regions are I/O regions, supporting no accesses

Supported Access Type PMAs

- Specify which access widths are supported + if misaligned accesses are supported
- Read those from PMP
- Main memory always support read, write and execute of all access widths required
- I/O regions may specify which combination of read, write or execute accesses to which data widths are supported

Atomicity PMAs

- Describe which atomic instructions are supported in the specified region
- Main memory must support the atomic operations required by the processor
- I/O regions may only support a subset of the atomic operations
- Two categories for atomic instruction support: LR/SC and AMOs

AMO Class	Supported Operations
AMONone	*None*
AMOSwap	`amoswap`
AMOLogical	above + `amoand`, `amoor`, `amoxor`
AMOArithmetic	above + `amoadd`, `amomin`, `amomax`, `amominu`, `amomaxu`

- Memory regions that support LR/SC must define the condition under which LR/SC sequences succeed or fail (Load reserved/ Store Conditional)
- More information in the privileged spec!

Memory-Ordering PMAs

- Main Memory
  - Access by one hart to main memory regions are observable from other devices
  - Main memory regions always have either the RVWMO or RVTSO memory model
- I/O Memory
  - Either relaxed or strong ordering
  - relaxed: no ordering guarantees on how memory accesses by one hart are observable by others, beyond those enforced by FENCE and AMO instructions
  - strong: all accesses are observable in program order
  - Each strong ordered I/O region specifies a numbered ordering channel
    - Channel 0:
      - Indicate point-to-point strong ordering only (accesses by the hart to the single associated region are strongly orderd)
    - Channel 1:
      - Provide global strong ordering across al I/O regions
      - All accesses by a hart to any I/O region associated with Channel 1 can be observed to have appeared in program order by all other harts or I/O devices
      - Equivalent to a fence io, io instruction before and after the instruction

Coherence and Cacheability PMAs

- Coherence: defines that writes to that address by one agent will eventually be made visible to other agents in the system (strongly encouraged for all addresses)
- Cacheability: should not affect the software view of the region
- More information on caches in the spec!

Idempotency PMAs

- Describes if reads and writes to an address region are idempotent
- Not idempotent means there can be side effects on any access
- Main memory: assumed to be idempotent
- I/O: idempotency for reads & writes can be specified separately

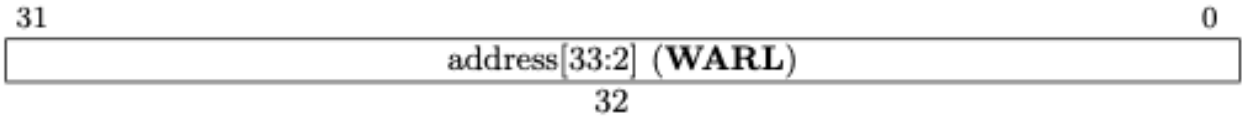
Physical Memory Protection

- optional: PMP (Physical Memory Protection) Unit to provides per-hart M-mode control registers to allow physical memory access privileges to be specified fro each physical memory region
- PMP values are checked in parallel with the PMA checks
- Done for all accesses in the S&U mode and loads & stores if the MPRV bit is set in the mstatus register & the MPP field in the mstatus register contains S or U.
- Also applied to page-table accesses for virtual-address translation (effective privilege mode is S)
- PMP may be also applied to M-mode accesses -> PMP registers are locked, M-mode software cannot change them without a system reset
- PMP violations are always trapped precisely at the processor
- PMP can grant permission to S & U-mode (have none by default) and revoke permissions from M-mode (has all by default)

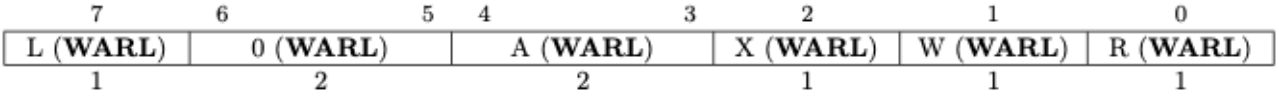
PMP CSRs

- PMP entries are described by an 8-bit config register and one MXLEN-bit address reg.
- Some PMP settings use the address register associated with the preceding PMP entry
- 16 PMP entries are supported, if one is implemented all must be implemented
- Can be hardwired to 0 if not used
- Four CSRs (*pmpfg0-3*) hold the configurations (*pmp0cfg-15cfg*)

- The PMP address registers pmpaddr0-15:



- The address register encodes bits 33-2 of the 34-bit physical address for RV32 (34 bits are supported by the page-based virtual-memory scheme)
- PMP configuration register format:



- R=read, W=write X=execute
- Fetching an instruction from a PMP region with X=0 raises an „fetch access exception“
- A load or load-reserved to an address without read permissions raises a load access exception
- A store or store-conditional or AMO to an address with out write permission raises a store access exception

Address Matching

- The A field in a PMP entry’s configuration register encodes the address-matching mode of an associated PMP address register

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

- A=0
  - Entry is disabled, matches no addresses
- A=1
  - Associated address register forms the top of the address range
  - 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 pmpaddr(i-1).<= y < pmpaddr(i)
  - If i = 0: y < pmpaddr0
- A = 2|3

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^{XLEN+1}-byte NAPOT range
0111...1111	NAPOT	2^{XLEN+2}-byte NAPOT range
1111...1111	NAPOT	2^{XLEN+3}-byte NAPOT range

Locking and Privilege Mode

- L-bit indicates PMP entry is locked
- Writes to the configuration register and associated registers are ignored
- Only unlockable with a system reset
- If L is set, R/W/X checks are performed on memory accesses of all privilege levels
- If L is clear, any M-mode access matching the PMP entry will succeed (independent of the R/W/X settings)

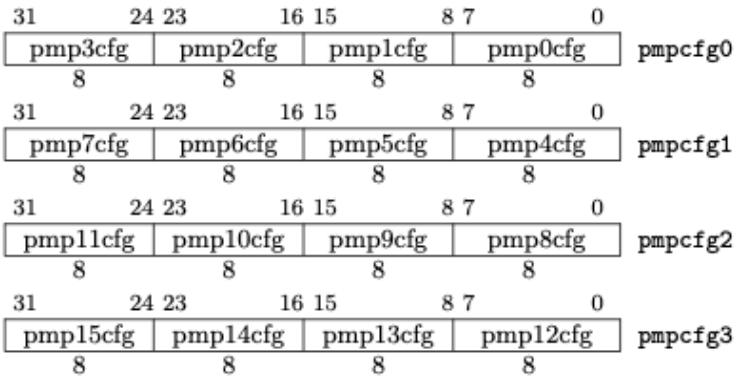
Priority and Matching Logic

- PMP entries are statically prioritized -> the lowest-numbered PMP entry that matches **any** byte of an access determines whether it fails or succeeds
- If a PMP entry matches all bytes of an access, then the L,R,W&X bits determine the success
- If L-bit is clear and the privilege Mode is M the access succeeds
- Otherwise (L-bit set | S- | U-mode) R/W/X check is performed
- If no PMP entry matches a M-mode access, the access succeeds
- If no PMP entry matches a S- or U-mode access, but at least one PMP entry is defined it fails
- Failed accesses generate a load, store or instruction access exception

---

Physical Memory Protection and Paging

More information in the privileged spec !


31 24 23 16 15 8 7 0
pmp3cfg
pmp2cfg
pmp1cfg
pmp0cfg
pmpcfg0
8 8 8 8
31 24 23 16 15 8 7 0
pmp7cfg
pmp6cfg
pmp5cfg
pmp4cfg
pmpcfg1
8 8 8 8
31 24 23 16 15 8 7 0
pmp11cfg
pmp10cfg
pmp9cfg
pmp8cfg
pmpcfg2
8 8 8 8
31 24 23 16 15 8 7 0
pmp15cfg
pmp14cfg
pmp13cfg
pmp12cfg
pmpcfg3
8 8 8 8