RISC-V Privileged Architecture

Allen Baum
Esperanto Technologies.
allen.baum@esperantotech.com

8th RISC-V Workshop
Barcelona, Spain
May 7, 2018
Introduction to RISC-V Privileged Architecture

- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance
Why a Privileged Architecture?

- We need ways of managing shared resources
  - Memory
  - I/O Devices
  - Cores

- We need ways of protecting shared resources
  - Memory: use virtual memory mapping
  - I/O: also virtual memory mapping
  - Access permissions: integrated into mapping (or as separate functionality)

- We need ways of insulating implementation details
  - Trapping unimplemented ops for SW emulation
  - Handling external asynchronous events
    - (IO events, Timer events, SW interrupts from other threads)
  - 2 Level address translation for Hypervisor support
### RISC-V Privileged Architecture Layers

- Provides clean split between layers of the software stack

<table>
<thead>
<tr>
<th>Layer</th>
<th>Communicates with</th>
<th>via</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Application Execution Environment (AEE)</td>
<td>Application Binary Interface (ABI)</td>
</tr>
<tr>
<td>Operating System</td>
<td>Supervisor Execution Environment (SEE)</td>
<td>System Binary Interface (SBI)</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>Hypervisor Execution Environment (HEE)</td>
<td>Hypervisor Binary Interface (HBI)</td>
</tr>
</tbody>
</table>

- **ECALL** instruction used for the communication
- All ISA levels designed to support virtualization
Introduction to RISC-V Privileged Architecture

- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance

Profiles
RISC-V is both basic and feature-rich
- Managed by distinct architectural modes and optional features

Different modes/options combinations grouped into platform profiles
- A restricted combination of all possible options

Currently platform profiles are a work-in-progress, though there are abstract classes:
- Simple Embedded Systems
- Embedded Systems w/ Protection
- Unix-like OS capable
- Cloud OS

See https://github.com/riscv/riscv-platform-specs
## Some basic platform profiles

<table>
<thead>
<tr>
<th>Profile</th>
<th>Modes</th>
<th>Trust</th>
<th>Mem Protect</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded without Protection</td>
<td>M</td>
<td>all Trusted</td>
<td>None</td>
<td>Low cost:16B ea of arch. state, timers, basic perf counters</td>
</tr>
<tr>
<td>Embedded with Protection</td>
<td>M+U</td>
<td>Apps not trusted</td>
<td>Phys Mem Protect</td>
<td>Optional N-extension for user int. handling</td>
</tr>
<tr>
<td>Unix-like OS capable</td>
<td>M+S+U</td>
<td>OS Trusted</td>
<td>Vmem + RWX</td>
<td>Vaddr size options: 32,39,48 b</td>
</tr>
<tr>
<td>Cloud OS capable</td>
<td>M+(<a href="S+U">V</a>)</td>
<td>Hypervisor Trusted</td>
<td>2-level Vmem + RWX</td>
<td>Unix+ Supports &gt;1 OS +new / background CSRs w/ swapping</td>
</tr>
</tbody>
</table>

---

The table above outlines different profiles based on their modes, trust levels, memory protection, and additional features. Each profile is characterized by its specific modes (M, M+, M+U), whether they include protection for the system (trusted), and the memory protection options (None, Phys Mem Protect). Additional details include cost considerations and supported operating systems.
- Why a Privileged Architecture?
- Profiles
- **Privileges and Modes**
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance
RISC-V Privilege Modes

- User & 2+ privileged modes (hierarchical)
  - User (U-mode), normal and virtualized* (lowest privileges)
  - Supervisor (S-mode), normal and virtualized*
  - Machine (M-mode) (highest privileges)**

- Supported combinations of modes:
  - M (simple embedded systems)
  - M, U (embedded systems with protection)
  - M, S, U (systems running Unix-like operating systems)
  - M, [V]S, [V]U (systems running multiple Oses)

- Each privileged mode add a few ops, and Control/Status Registers (CSRs) that control operations
  - CSRs accessible only by code running at a specific privilege mode or higher
  - There are (often) multiple CSR copies/views for each mode

*Virtualized modes for hypervisor support- not covered here

** An even higher mode exists (Debug) which can only be entered if a debug port is connected & enabled, w/ separate state saving CSRs. Otherwise much like M-mode
Why a Privileged Architecture?

Profiles

Privileges and Modes

**Privileged Features**

- CSRs
- Instructions

**Memory Addressing**

- Translation
- Protection

**Trap Handling**

- Exceptions
- Interrupts

**Counters**

- Time
- Performance
Privileged Machine & Supervisor (M-, S-) modes add instructions over & above the base U-mode user ISA ops
- Priv Insts can only be executed from appropriate mode (or higher)

- **All modes**
  - **ECALL:** generates <curr_mode>environment_call exception
  - **EBREAK:** generates breakpoint exception
  - **[x]RET:** returns from a trap *from* the specified mode
    - **SRET** provided only if S-mode is implemented
    - **URET** provided only if U-mode traps supported (N-extension)

- **S-mode (+M-mode): adds**
  - **SFENCE.VMA:** synchs updates to implicitly accessed memory

- **M-mode**
  - **WFI:** stall the current hart until an interrupt needs service
    - Is a hint only (could be noop, could direct interrupts to this hart)
Control/Status Registers (CSRs) have their own address space

- Each hart has its own set of 4K CSRs (1K/mode)
- CSRs are accessed by dedicated ops
  - that can implement atomic swap or bit set/clear
  - Direct address mode only
- CSRs are mode sensitive
  - Can only be accessed by code in appropriate or higher privileged mode; accesses by lower privilege modes will trap
- Many CSRs optional/ have optional fields/mode dependent
  - Accesses to non-existent CSRs will trap
  - Writes to Read_Only CSRs will trap
    - But writes to read_only fields in read/write CSRs are ignored
  - Accesses to optional CSRs read zeroes, & (if RW) ignore writes
    - Note that optional vs. non-existent can depend on architecture!

Note that optional vs. non-existent can depend on architecture!
# CSR address space

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00 XX</td>
<td></td>
<td></td>
<td></td>
<td>0b00MM xxxxxxxx</td>
</tr>
<tr>
<td>01 0X</td>
<td></td>
<td></td>
<td></td>
<td>0b01MM 0xxxxx</td>
</tr>
<tr>
<td>01 10</td>
<td></td>
<td></td>
<td></td>
<td>0b01MM 100xxxxx</td>
</tr>
<tr>
<td>01 10 DBG only</td>
<td></td>
<td></td>
<td></td>
<td>0b01MM 1010xxxxx</td>
</tr>
<tr>
<td>01 10</td>
<td></td>
<td></td>
<td></td>
<td>0b01MM 1011xxxxx</td>
</tr>
<tr>
<td>01 11</td>
<td></td>
<td></td>
<td></td>
<td>0b01MM 11xxxxxx</td>
</tr>
<tr>
<td>10 ~11</td>
<td></td>
<td></td>
<td></td>
<td>0b10MM ~11xxxxxx</td>
</tr>
<tr>
<td>10 11</td>
<td></td>
<td></td>
<td></td>
<td>0b10MM 11xxxxxx</td>
</tr>
<tr>
<td>11 ~11 (RO)</td>
<td></td>
<td></td>
<td></td>
<td>0b11MM ~11xxxxxx</td>
</tr>
</tbody>
</table>

Addr[9:8]==10 currently reserved for Hypervisor CSs
## CSRs and categories

<table>
<thead>
<tr>
<th>Category</th>
<th>CSR Name (some replicated/mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP CSRs</td>
<td>Exceptions, Rounding_Mode, Reg_State</td>
</tr>
<tr>
<td>Information</td>
<td>Vendor/ Architecture/Implementation/Thread_IDs</td>
</tr>
<tr>
<td>Trap Setup</td>
<td>Status, Trap_Vector, ISA+Extension, Int_Enable, Int/Excep_Delegation, Cntr_Enab</td>
</tr>
<tr>
<td>Trap Handling</td>
<td>Exception_PC, Scratch, Int_Pending, Trap_Cause/Value</td>
</tr>
<tr>
<td>Protection/Translation</td>
<td>Address_Translation_Protection, PhysMemProtection Config[ ]/Addr[ ]</td>
</tr>
<tr>
<td>Counter/Timers</td>
<td>Cycles, Time, Inst_Retired, PerfmonCntr [ ]</td>
</tr>
<tr>
<td>Counter Setup</td>
<td>Perfmon Event selector[ ]</td>
</tr>
<tr>
<td>Debug/Trace</td>
<td>Control/Status/PC/Scratch Trigger_RegSelect/Data[ ]</td>
</tr>
</tbody>
</table>
Memory Addressing:
Translation

- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSR
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance
Memory Address Translation: Virtual Memory

- S-mode adds provide virtual memory page mapping
  - Smallest unit of protection/mapping is 4 KB pages
- Supports multiple User mode processes w/ separate address spaces (using Addr_Space_ID field in SATP CSR)
- Page tables have multiple levels that are walked:
  - 2 levels for RV32 (Sv32)
  - 3,4 levels for RV64 (Sv39, Sv48)
  - 5,6 levels RSVĐ (Sv57, Sv64)
- Page Table walk can stop at any level to create Superpages
  - e.g. for Sv39 2 MB if stopped at 2 levels
  - or 1 GB if stopped at 1 levels
- HW Page Table walk semantics specified in Priv Mode spec
  - But could trap to M-mode for software TLB refill
- Accessed/Dirty bits optionally managed by HW
  - Updates must be atomic w.r.t. permissions check
  - Complex to implement, so all trapping when A/D clear
- Global bit indicates mapping belongs to all addr spaces (e.g. kernel pages in Unix systems)
- Page granularity permissions (User/Read/Write/eXecute)
  - (000 XWR indicates a non-leaf entry)
- Virtual Addr width, Current ASID, and PageTable root controlled by **SATP CSR**

**RISC-V Page Table Entries**

(SV32 format shown)

<table>
<thead>
<tr>
<th>Mode</th>
<th>ASID</th>
<th>Page Table Root Physical Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>16</td>
<td>44</td>
</tr>
</tbody>
</table>

(SV64 format shown)
More control: Memory Fences

- S-mode implements **SFENCE.VMA** instruction to synchronize updates to memory data structures
  - All page table levels, or just those corresponding to an addr
  - All address spaces, or just a specific address space (not global)

- Generalization of TLB flush on other architectures

- Guarantees that all prior stores are ordered before all subsequent *implicit* references in the instruction stream

- Affects only the local hart
  - Synchronization with other harts requires IPIs
Memory Addressing:

- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions

Memory Addressing
  - Translation
  - Protection

- Trap Handling
  - Exceptions
  - Interrupts

- Counters
  - Time
  - Performance
RISC-V Virtual Memory Protections

- Standard RWX permissions configurable for every page
  - Supports X-only pages
  - W & ~R combination reserved
- By default, S-mode can’t access user pages
  - Helps detect OS/driver bugs
  - Still need ability to read user memory, e.g. on system call
  - Set "Supervisor Access to User Memory" (SUM) bit in sStatus to read user memory, then turn it off again
  - S-mode cannot execute from U-mode pages even if SUM=1
- Similarly, S-mode can’t read execute-only pages
  - Set sStatus “Make eXecutable Readable” MXR bit to override
  - Useful for illegal-instruction trap handlers
- S-mode can enable/disable VM and choose page-table depth in SATP register
RISC-V Physical Memory Protection Unit

- Optional new feature in v1.10
- When implemented, modes below M-mode have no permissions by default (unless region is locked)
- Grants R/W/X permissions to up to 16 PMP regions
  - Naturally aligned $2^N$-byte regions ($N \geq 2$)
- Adjacent PMP registers can be configured to form an arbitrary base-and-bounds region instead
- PMPs can be locked (can’t be rewritten until reset), in which case they affect M-mode, too
- The fine print:
  - If VM enabled, VM (& page faults) occur before PMP checks
  - Useful for untrusted S-Mode
Physical Memory Attributes

- RISC-V systems have the concept of Physical Memory Attributes: *platform and implementation specific*
- PMA is dedicated HW that maps specific address ranges to certain access attributes, e.g.
  - Access widths allowed (e.g. 1/2/4/8/16/64B)
  - Alignment restrictions (e.g. can’t cross $2^x$ byte boundary)
  - Idempotency (enabling speculation)
  - Ordering (Strong/Weak per Channel)
  - Cacheability (incl Wt Thru, Wt Combining, etc)
  - Priority (e.g. high/low if conflicting)
  - Atomicity allowed (none, swap, logical, arithmetic)
  - Allowed access modes (M/S/U/debug)

- Some attributes could be configurable
Trap Handling: Exceptions and Interrupts

- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance
Interrupts vs Exceptions

- **Exceptions**: Synchronous events
  - Synchronous: caused by a specific instruction execution
- **Interrupts**: Asynchronous events
  - not caused by an inst: I/O, timer, SW (from another hart)
- Both handled (almost) identically by trapping:
  - **xTVEC** CSR holds handler address
  - Interrupts optionally vector to **xTVEC**+4*xCause
  - **xI*/EDELEG** CSRs: select mode to trap into (next slide)
  - **xCause** CSR (x=new mode) saves cause ID
    - MSB: interrupt vs. exception, LSBs: interrupt /exception ID code
  - **xTVAL** CSR saves additional information about cause
    - This could be an illegal address, or illegal opcode
  - **xEPC** CSR saves return Program Counter
    - could be next instruction (interrupts) or same inst (enabling retry)
  - **xSTATUS** CSR saves curr Mode/IntEn bits
    - **xSTATUS[IntEn]** cleared
Interrupt/Exception Handler Delegation

- Traps always sent to M-mode, but...
- Can be delegated to lower priv level, reducing overhead
  - Never to a less privileged mode than the one that trapped!

- Bits in delegation CSR send traps to next lower priv level
  - m[i/e]deleg: M→S (or M→U if no S-mode & N_extension)
  - s[i/e]deleg: S→U, (if delegated to S-mode & N_extension)

- Int Delegation occurs only if corresponding enable bits set (<x>ie CSR)
  - But enable bit used only for delegated mode
  - Exceptions are always enabled

- When an enabled interrupt trap, the corresponding bit in <x>ip CSR is set
## Interrupt/Exception Causes

- `<x>cause` CSR indicates which interrupt/exception occurred
- Corresponding bit is set in `<x>E/IP` CSR

### Trap Priority for simultaneous interrupts/exceptions:

<table>
<thead>
<tr>
<th>Trap code[62:0]</th>
<th>Exception (Cause[MSB]=0)</th>
<th>Interrupt (Cause[MSB]==1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Instruction addr misaligned</td>
<td>User Software Interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Instruction access fault</td>
<td>Supervisor Software Interrupt</td>
</tr>
<tr>
<td>2</td>
<td>Illegal instruction</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Breakpoint</td>
<td>Machine Software Interrupt</td>
</tr>
<tr>
<td>4</td>
<td>Load address misaligned</td>
<td>User Timer Interrupt</td>
</tr>
<tr>
<td>5</td>
<td>Load access fault</td>
<td>Supervisor Timer Interrupt</td>
</tr>
<tr>
<td>6</td>
<td>Store/AMO addr misaligned</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Store/AMO access fault</td>
<td>Machine Timer Interrupt</td>
</tr>
<tr>
<td>8</td>
<td>Environment call</td>
<td>User External Interrupt</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
<td>Supervisor External Interrupt</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>Machine</td>
<td>Machine External Interrupt</td>
</tr>
<tr>
<td>12</td>
<td>Instruction page fault</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Load page fault</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Store/AMO page fault</td>
<td></td>
</tr>
<tr>
<td>&gt;=16</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

- Interrupts > Exceptions
- M-mode > S-mode > U-mode
- Pending[N] > Pending[M] if N>M

Special case: Timer & SW interrupt priorities swapped!
- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - **Interrupts**
- Counters
  - Time
  - Performance
RISC-V Interrupt Source Categories

- **Local Interrupts**
  - Directly connected to one hart, independent of other harts
  - Cause determined directly from `<x>cause` CSR
  - Only two standard local interrupts (software, timer)

- **Global (External) Interrupts**
  - Routed to harts via Platform-Level Interrupt Controller (PLIC)
  - Actual source determined by read of PLIC MMIO CSR

- **Any interrupt can target any M/S/U mode**
  - Except for priority during simultaneous interrupts, handling is identical
Software Interrupts

- Software interrupt are how harts interrupt each other
  - Mechanism for inter-hart interrupts (IPIs)

- Setting the appropriate `<x>SIP` bit in another hart is performed by a MMIO write
  - But a hart can set its own `<x>SIP` bit if `currmode` >= `<x>

- App/OS performs inter-hart ints only via ABI/SBI calls
  - Destination virtual hart might be descheduled
  - Interrupts virtualized by M-mode software using MSIP
Timer Interrupts

- Single M-mode 64b real-time HW timer & comparator
- **NOT** a CSR, but MMIO addressed
  - Must count at a fixed rate, regardless of core clock or power
  - `mtime >= mtimecmp` causes hart’s **MTIP** bit to be set
- M-mode responsible for virtualizing the single HW timer and comparator for lower-privilege modes
  - on same hart, **and** between harts
  - U-mode CSR reads trap & are handled by M-mode

- **STIP** and **UTIP** CSR bits are handled by M-mode
  - ABI/SBI calls to set up timer
  - M-mode software writes/clears **STIP, UTIP**
External Interrupts

- Inputs from a Platform-Level Interrupt Controller (PLIC)
  - PLIC targets hart based on hart interrupt threshold & enable, and interrupt priority
- Interrupts can target multiple harts simultaneously
  - Harts must arbitrate to determine which services it
  - E.g. by racing to read MMIO mapped interrupt source CSR
- PLIC labels each output with a privilege mode
  - Which can be handled differently using delegation
- Interrupts cleared via MMIO mapped LD/ST to PLIC
- Software can inject **SEIP** and **UEIP** interrupts to support virtualizing the PLIC by writing CSR directly
- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance
Timers and Counters

- RISC-V has several architected Timers and Counters implemented (mostly) as CSRs
- All are 64 bits (split into 2 CSRs for RV32 only)

  - Real Time Clock **Time**, described in Timer Interrupt slide
    - U/S-mode read of CSR traps to M-mode, which does MMIO read

  - Instructions Retired **InstRet** Counter
    - M-mode RW, U-mode RO, used for **RDINSTRET** pseudo-instructions

  - Cycles **Cycle** Counter
    - M-mode RW, U-mode RO, used for **RDCYCLE** pseudo-instructions

  - 0..29 HW Performance Monitors **mhpmcounters**
    - each w/corresponding **HPMEvent** to select what to count
Timer/Counter protections

- Easily accessible timers have issues
  - Lack of reproducibility
  - Side channel security attacks (Meltdown, Spectre...)

- `<x>CounterEn` CSRs enables access to the counters
  - 1 bit per counter (`Time/Cycle/InstRet/HPMCounter[]`)
  - Accessing `<x>timer/counter` in a mode `<x>` will trap if corresponding bit in `<x>CounterEn` is clear for `x<y`
  - Any bit may be optionally hardwired to zero
Wrap Up
Privileged Architecture is Stable

- Latest version is v1.11 draft
- keeps compatibility with v1.9.1 for machine-mode-only implementations
- Future releases should be compatible with v1.10 for supervisor ISA, too
- Adds draft Hypervisor support
- Caveat: these are proposals; not yet ratified by Foundation
Implementation Status

- Spike and UCB Rocket-Chip conform to v1.11
- Linux port is upstreamed and conforms to v1.11  
  - works with Spike/Rocket
- QEMU port is upstreamed and conforms to v1.11
- Upstream GCC and binutils ports are compatible
Questions?

Specs available at
https://github.com/riscv/riscv-isa-manual
Backup
<x>Status CSR

(SD moves to bit 31 for SV32)

(S-mode XLEN
U-mode XLEN

Extension state is dirty

TSR  Trap if SRET is executed in S-mode
TW   Trap if WFI in S-mode exceeds timeout
TVM  Trap on S-mode execution of SFENCE.VMA

MXR Allow Ld from pages with RX=01
SUM Allow SMode accesses to U-mode pages
MPRV  Use priv mode in MPP for Ld/St

Summary (other)
0= All Off
1= No Dirty/Clean, some on
2=No dirty, Some clean
3=Some dirty

Previous Privilege Modes
M: MSU
S: SU

FP
0= Off
1= Initial
2=Clean
3=Dirty

Pre& Mode
Int Enable

SV64 only

PP, PIE, IE
read as zero if curr_mode<x

Previous
Current

In S,U-mode read as zero
In U-mode read as zero

In S,U-mode read as zero

<table>
<thead>
<tr>
<th>CSR Category</th>
<th>CSR Name</th>
<th>Comments</th>
<th>M-mode</th>
<th>S-mode</th>
<th>Umode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating-Point CRs</td>
<td>Accrued Exceptions</td>
<td>(o) Encoded JEDEC ID</td>
<td>X(DF)</td>
<td>X(DF)</td>
<td>X(DF)</td>
</tr>
<tr>
<td></td>
<td>Dynamic Rounding Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ctrl &amp; Status Reg (frm + fflags)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information</td>
<td>Vendor ID</td>
<td>(o)</td>
<td>X</td>
<td>HBX (H)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Architecture ID</td>
<td>(o)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Implementation ID</td>
<td>(o)</td>
<td>X</td>
<td>BX (H,N)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Hardware_thread ID</td>
<td>(o)</td>
<td>X</td>
<td>BX (H,N)</td>
<td>X</td>
</tr>
<tr>
<td>Trap Setup</td>
<td>Status</td>
<td>(S)</td>
<td>X</td>
<td>BX (H)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>ISAs and Extensions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exception_Delegation</td>
<td>(S)</td>
<td>X</td>
<td>BX (H)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Interrupt Enable</td>
<td>(R)</td>
<td>X</td>
<td>BX (H)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Trap_Vector_Base_Address</td>
<td>(Z)</td>
<td>X</td>
<td>BX (H)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Counter Enable</td>
<td>(Z)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Trap Handling</td>
<td>Scratch Register</td>
<td>For Trap handlers</td>
<td>X</td>
<td>BX (H)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Exception_Program_Counter</td>
<td>(P,Z)</td>
<td>X</td>
<td>BX (H)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Trap_Cause</td>
<td>(Z)</td>
<td>X</td>
<td>BX (H)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Interrupt pending</td>
<td>(R)</td>
<td>X</td>
<td>BX (H)</td>
<td>X</td>
</tr>
<tr>
<td>Protection/Translation</td>
<td>Phys_Mem Prot Config[3:0]</td>
<td>(on)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Phys_Mem Prot Addr[15:0]</td>
<td>(on)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Counter/Timers</td>
<td>Cycle_counter</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>For RTIME inst</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instr-retired_counter</td>
<td>For RDINSTRET inst</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Perfmon counters[31:3]</td>
<td>(Zn)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Upper_32b_of_cycle</td>
<td></td>
<td>X (I)</td>
<td>X (U32)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Upper_32b_of_time</td>
<td></td>
<td>X (I)</td>
<td>X (U32)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Upper_32b_of_instr</td>
<td></td>
<td>X (I,n)</td>
<td>X (U32)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Upper_32b_of_perfmon[31:3]</td>
<td>(Z)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Counter Setup</td>
<td>Perfmon Event selector[31:3]</td>
<td>(Z)</td>
<td>X(n)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Debug/Trace</td>
<td>Debug/Trace Trigger Reg Select</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Debug/Trace trigger data Reg[3:1]</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Debug Control/Status</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Debug PC</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Debug Scratch Reg</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

(DF) Optional unless D/F extension implemented
(UN) Exists only in Umode & N-extension implemented
(U32)Exists only in Umode & RV32I architecture
(n) Exact number is implementation dependent
(I) exist only if RV23I architecture
(R) Single register with restricted view
(P) Only bits corresponding to <= curr mode are visible
(H) Swapped with background CSR on VM entry/exit
(o) Optional (Read 0 if unimplemented)
(SN)Exists only if Smode | N-extension implemented
(z) Some bits may be hardwired RdOnly
(Z) May be hardwired RdOnly zero

* = optional (read as zero)

Types of CSR fields:
- WIRI (Reserved) Writes ignored, Reads ignored
- WPRI (Reserved) Writes preserved, Reads ignored
- WURL Write Legal, Read Legal
- WARL Write Any, Read Legal

H= new CSR version added for Hypervisor extension
B = background CSR added for Hypervisor extension
Interrupt/Exception Handler Delegation

Global Enable M = currMode<M + currMode==M & Mstatus.MIE
Global Enable S = currMode<S + currMode==S & Mstatus.SIE
Global Enable U = currMode<U + currMode==U & Mstatus.UIE

- Treat as 0 if no S-mode or no U-mode or no N_extension
- Treat as 1 if S-mode unimplemented
- Interrupts > Exceptions
  M > S > U-mode
  Pending[N] > Pending[M]
  if N>M
  Exception: Timer & SW interrupt
  priorities swapped!

Interruption or Exception

Deleg M

Exception or Interrupt

Deleg S

Deleg U

Global Enables

Int Enable

Priority Logic

Int

Enable

PEND

M

S

U

Global Enables

Trap

New Mode

Cause

Exceptions always enabled

Interrupts only
Interrupt Pending/Enable CSRs <x>ip,ie

- **<x>ip** reflects pending status of interrupts for hart
  - Enabled by corresponding bits of **<x>ie** with same per/mode visibility
  - In addition to global interrupt enables in **<x>status** for each privilege mode
- **Separate ints for each priv level (M/S/U)**, directed to M-mode
  - M-mode can delegate to S-mode and U-modes
  - Higher privilege modes override lower privilege modes
- **Opt. User interrupt handling (“N”) feature when U-mode present**
- **Interrupts always disabled for privimodes lower than current mode; always enabled for privilege modes higher than current mode**
foreach hart {
  find highest priority enabled interrupt
  if (interrupt_priority > hart_priority) {
    present interrupt[highest] to hart
  }
}
Hypervisor mode

- Feedback led us to HW support for Type-2 hypervisors (like KVM)
  - Can also support type-1
- Hypervisors run in S-mode
- Guests run in virtualized (V) S and U modes
  - Major difference is 2-level page table walk
  - Certain operations can be inhibited or trap (individually)
    - Execution of WFI if the wait exceeds some limit can be trapped
    - SRET, FENCE.VMA, SATP and counters CSR accesses can be trapped
      - But CYCLE and INSTRET will still count
    - Force translation to use supervisor previous priv level
  - Additional bits added to STATUS CSR
    - Previous Virtualization mode
    - Translation fault level
  - Some control bits interpreted differently: SPRV, SPV, SPP
- Vmode changes cause CSR swap/selection w/background versions
  - Use HSTATUS, HEDELEG, HIDELEG, HTVAL
  - Swaps SSTATUS, SIE, SIP, STVEC, SSCRATCH, SCAUSE, SEPC, STVAL, SATP