3.1.5 Hart ID Register `mhartid`

The `mhartid` CSR is an MXLEN-bit read-only register containing the integer ID of the hardware thread running the code. This register must be readable in any implementation. Hart IDs might not necessarily be numbered contiguously in a multiprocessor system, but at least one hart must have a hart ID of zero. Hart IDs must be unique within the execution environment.

MXLEN-1 … 0
Hart ID
MXLEN

Figure 3.5: Hart ID register (`mhartid`).

In certain cases, we must ensure exactly one hart runs some code (e.g., at reset), and so require one hart to have a known hart ID of zero.

For efficiency, system implementers should aim to reduce the magnitude of the largest hart ID used in a system.

3.1.6 Machine Status Registers (`mstatus` and `mstatush`)

The `mstatus` register is an MXLEN-bit read/write register formatted as shown in Figure 3.6 for RV32 and Figure 3.7 for RV64. The `mstatus` register keeps track of and controls the hart's current operating state. A restricted view of `mstatus` appears as the `sstatus` register in the S-level ISA.

31	30 … 23	22	21	20	19	18	17
SD	**WPRI**	TSR	TW	TVM	MXR	SUM	MPRV
1	8	1	1	1	1	1	1

16 15	14 13	12 11	10 9	8	7	6	5	4	3	2	1	0
XS[1:0]	FS[1:0]	MPP[1:0]	VS[1:0]	SPP	MPIE	UBE	SPIE	**WPRI**	MIE	**WPRI**	SIE	**WPRI**
2	2	2	2	1	1	1	1	1	1	1	1	1

Figure 3.6: Machine-mode status register (`mstatus`) for RV32.

63	62 … 38	37	36	35 34	33 32	31 … 23	22	21	20	19	18
SD	**WPRI**	MBE	SBE	SXL[1:0]	UXL[1:0]	**WPRI**	TSR	TW	TVM	MXR	SUM
1	25	1	1	2	2	9	1	1	1	1	1

17	16 15	14 13	12 11	10 9	8	7	6	5	4	3	2	1	0
MPRV	XS[1:0]	FS[1:0]	MPP[1:0]	VS[1:0]	SPP	MPIE	UBE	SPIE	**WPRI**	MIE	**WPRI**	SIE	**WPRI**
1	2	2	2	2	1	1	1	1	1	1	1	1	1

Figure 3.7: Machine-mode status register (`mstatus`) for RV64.

For RV32 only, `mstatush` is a 32-bit read/write register formatted as shown in Figure 3.8. Bits 30:4 of `mstatush` generally contain the same fields found in bits 62:36 of `mstatus` for RV64. Fields SD, SXL, and UXL do not exist in `mstatush`.

31 ... 6	5	4	3 ... 0
WPRI	MBE	SBE	**WPRI**
26	1	1	4

Figure 3.8: Additional machine-mode status register (`mstatush`) for RV32.

3.1.6.1 Privilege and Global Interrupt-Enable Stack in `mstatus` register

Global interrupt-enable bits, MIE and SIE, are provided for M-mode and S-mode respectively. These bits are primarily used to guarantee atomicity with respect to interrupt handlers in the current privilege mode.

The global x*IE bits are located in the low-order bits of* `mstatus`*, allowing them to be atomically set or cleared with a single CSR instruction.*

When a hart is executing in privilege mode x, interrupts are globally enabled when xIE=1 and globally disabled when xIE=0. Interrupts for lower-privilege modes, w<x, are always globally disabled regardless of the setting of any global wIE bit for the lower-privilege mode. Interrupts for higher-privilege modes, y>x, are always globally enabled regardless of the setting of the global yIE bit for the higher-privilege mode. Higher-privilege-level code can use separate per-interrupt enable bits to disable selected higher-privilege-mode interrupts before ceding control to a lower-privilege mode.

A higher-privilege mode y *could disable all of its interrupts before ceding control to a lower-privilege mode but this would be unusual as it would leave only a synchronous trap, non-maskable interrupt, or reset as means to regain control of the hart.*

To support nested traps, each privilege mode x that can respond to interrupts has a two-level stack of interrupt-enable bits and privilege modes. xPIE holds the value of the interrupt-enable bit active prior to the trap, and xPP holds the previous privilege mode. The xPP fields can only hold privilege modes up to x, so MPP is two bits wide and SPP is one bit wide. When a trap is taken from privilege mode y into privilege mode x, xPIE is set to the value of xIE; xIE is set to 0; and xPP is set to y.

For lower privilege modes, any trap (synchronous or asynchronous) is usually taken at a higher privilege mode with interrupts disabled upon entry. The higher-level trap handler will either service the trap and return using the stacked information, or, if not returning immediately to the interrupted context, will save the privilege stack before re-enabling interrupts, so only one entry per stack is required.

An MRET or SRET instruction is used to return from a trap in M-mode or S-mode respectively. When executing an xRET instruction, supposing xPP holds the value y, xIE is set to xPIE; the privilege mode is changed to y; xPIE is set to 1; and xPP is set to the least-privileged supported mode (U if U-mode is implemented, else M). If xPP$\neq$M, xRET also sets MPRV=0.

Setting x*PP to the least-privileged supported mode on an* x*RET helps identify software bugs in the management of the two-level privilege-mode stack.*

xPP fields are **WARL** fields that can hold only privilege mode x and any implemented privilege mode lower than x. If privilege mode x is not implemented, then xPP must be read-only 0.

M-mode software can determine whether a privilege mode is implemented by writing that mode to MPP then reading it back.

If the machine provides only U and M modes, then only a single hardware storage bit is required to represent either 00 or 11 in MPP.

3.1.6.2 Base ISA Control in `mstatus` Register

For RV64 systems, the SXL and UXL fields are **WARL** fields that control the value of XLEN for S-mode and U-mode, respectively. The encoding of these fields is the same as the MXL field of `misa`, shown in Table 3.1. The effective XLEN in S-mode and U-mode are termed *SXLEN* and *UXLEN*, respectively.

For RV32 systems, the SXL and UXL fields do not exist, and SXLEN=32 and UXLEN=32.

For RV64 systems, if S-mode is not supported, then SXL is read-only zero. Otherwise, it is a **WARL** field that encodes the current value of SXLEN. In particular, an implementation may make SXL be a read-only field whose value always ensures that SXLEN=MXLEN.

For RV64 systems, if U-mode is not supported, then UXL is read-only zero. Otherwise, it is a **WARL** field that encodes the current value of UXLEN. In particular, an implementation may make UXL be a read-only field whose value always ensures that UXLEN=MXLEN or UXLEN=SXLEN.

Whenever XLEN in any mode is set to a value less than the widest supported XLEN, all operations must ignore source operand register bits above the configured XLEN, and must sign-extend results to fill the entire widest supported XLEN in the destination register. Similarly, `pc` bits above XLEN are ignored, and when the `pc` is written, it is sign-extended to fill the widest supported XLEN.

We require that operations always fill the entire underlying hardware registers with defined values to avoid implementation-defined behavior.

To reduce hardware complexity, the architecture imposes no checks that lower-privilege modes have XLEN settings less than or equal to the next-higher privilege mode. In practice, such settings would almost always be a software bug, but machine operation is well-defined even in this case.

If MXLEN is changed from 32 to a wider width, each of `mstatus` fields SXL and UXL, if not restricted to a single value, gets the value corresponding to the widest supported width not wider than the new MXLEN.

3.1.6.3 Memory Privilege in `mstatus` Register

The MPRV (Modify PRiVilege) bit modifies the *effective privilege mode*, i.e., the privilege level at which loads and stores execute. When MPRV=0, loads and stores behave as normal, using the translation and protection mechanisms of the current privilege mode. When MPRV=1, load and store memory addresses are translated and protected, and endianness is applied, as though