RISC-V

System-on-Chip Design

Harris, Stine, Thompson, & Harris

Chapter 8:

Privileged Operations

Chapter 8 :: Topics

Privileged Operations

8.1 Principles of Privileged Operations

8.2 RISC-V Practices

- 8.2.1 Privilege Modes
- 8.2.2 Privileged Instructions
- 8.2.3 CSRs (control and status registers)
- 8.2.4 Reset Behavior
- 8.2.5 Traps
- 8.2.6 Trap Handler

8.3 Test Plan (not covered in slides)

8.4 Wally Implementation

Chapter 8: Privileged Operations

Principles of Privileged Operations

Introduction

Privileged operations and modes enable a processor to enable **protection and security** and **respond to unusual events** (traps)

Issues privileged modes address:

- Basic integer core can access any feature or memory address, so a faulty or malicious program can crash or compromise machine.
- Processors must be able to respond to traps:
  - **Exceptions** (illegal instruction, access to non-existent memory, etc.)
  - **Interrupts** (i.e., external events like a key press, timer reaching a certain value, etc.)

Principles of Privileged Operations

- **Security and Protection**
  - **Security: general problem of preventing unauthorized persons from accessing a computer system and mitigating damage if it does happen**
  - **Protection:** the specific hardware and software to prevent security breaches
- **Privileged Modes** (from high to low privilege)
  - **Machine-mode (M-mode): can access everything**
  - **Supervisor-mode (S-mode): for operating system (OS)**
  - **User-mode (U-mode): for user programs**
- **Traps = Exceptions and interrupts**
  - **Exceptions:** from unusual program behavior
  - **Interrupts:** asynchronous, from peripherals
- **Precise Traps**
  - All instructions before trap are retired (completed)
  - No instructions after trap impact architectural state
- **Control and Status Registers (CSRs)**
  - Control traps
  - Enable optional features
  - Include performance counters

Chapter 8: Privileged Operations

RISC-V Practices

Privilege Modes

From highest to lowest privilege:

- **M-mode** (machine-mode)**:** can access everything
- **S-mode** (Supervisor-mode): for operating system (OS)
- **U-mode** (User-mode): for user programs

Code	Mode
00	User
01	Supervisor
11	Machine

Privilege mode is not directly visible to a program.
Programs change it by taking traps or using `mret` and `sret`.

Privileged Instructions

Instruction	Description
`mret`	Return from M-mode trap
`sret`	Return from S-mode trap
`wfi`	Wait for interrupt/exception
`sfence.vma`	Synchronize virtual memory (discussed in Chapter 8)
`ecall`	Environment call
`ebreak`	Debugger breakpoint
`csr*`	CSR access instructions (i.e., csrrw, csrrs, csrrc)

Chapter 8: Privileged Operations

CSRs

CSR Access Instructions

Format: `csrr* rd, csr, rs1`

CSR access instructions:

- **`csrrw:`** CSR read and write (`rd = CSR, CSR = rs1`)
- **`csrrs:`** CSR read and set (`rd = CSR, CSR |= rs1`)
- **`csrrc:`** CSR read and clear (`rd = CSR, CSR &= ~rs1`)

Often used in pseudoinstruction form:

- **`csrr:`** CSR read
  - Example: `csrr rd, csr # rd = csr`
  - Actual RISC-V instruction: `csrrs rd, csr, x0`
- **`csrw:`** CSR write
  - Example: `csrw csr, rs1 # csr = rs1`
  - Actual RISC-V instruction: `csrrw x0, csr, rs1`

Immediate versions also exists: `csrrwi, csrrsi, csrrci`

Format: `csrr?i rd, csr, imm`

`imm`: 5-bit unsigned immediate

M-Mode CSRs

CSR	Address	Type	Bits	Description
Machine Information Registers				
`mvendorid`	F11	MRO	32	JEDEC machine ID
`marchid`	F12	MRO	XLEN	Vendor-specific architecture ID
`mimpid`	F13	MRO	XLEN	Vendor-specific implementation ID
`mhartid`	F14	MRO	XLEN	Unique hart ID within the system; one hart has ID of 0
`mconfigptr`	F15	MRO	XLEN	Physical addr. of configuration data structure, or NULL
Machine Status and ISA				
`mstatus / h`	300/10	MRW	64	Machine status register
`misa`	301	MRW	XLEN	Instruction Set Architecture
Machine Traps				
`medeleg`	302	MRW	XLEN	Machine exception delegation
`mideleg`	303	MRW	12	Machine interrupt delegation
`mie`	304	MRW	12	Machine interrupt enable
`mip`	344	MRW	12	Machine interrupt pending
`mtvec`	305	MRW	XLEN	Machine trap vector
`mscratch`	340	MRW	XLEN	Machine scratch register
`mepc`	341	MRW	XLEN	Machine exception program counter
`mcause`	342	MRW	XLEN	Machine exception cause
`mtval`	343	MRW	XLEN	Machine trap value

S-Mode CSRs

CSR	Address	Type	Bits	Description
Supervisor Status and Protection/Translation				
`sstatus`	100	SRW	64	Supervisor status register
`satp`	180	SRW	XLEN	Supervisor address translation and protection
Supervisor Traps				
`sie`	104	SRW	12	Supervisor interrupt enable
`sip`	144	SRW	12	Supervisor interrupt pending
`stvec`	105	SRW	XLEN	Supervisor trap vector
`sscratch`	140	SRW	XLEN	Supervisor scratch register
`sepc`	141	SRW	XLEN	Supervisor exception program counter
`scause`	142	SRW	XLEN	Supervisor exception cause
`stval`	143	SRW	XLEN	Supervisor trap value

SRW: S-mode Read/Write
MRW: M-mode Read/Write
URO: User Read Only

Counter CSRs

CSR	Address	Type	Bits	Description
Counters				
mcounteren	306	MRW	32	Machine counter enable
mcountinhibit	320	MRW	32	Machine counter inhibit
mcycle/h	B00 / B80	MRW	64	Clock cycle counter
minstret/h	B02 / B82	MRW	64	Instructions retired counter
mhpmcounter3/h … mhpmcounter31/h	B03 / B83 … B1F / B9F	MRW	64	Other hardware perf. monitor counters
mhpmevent3 … mhpmevent31	323 … 33F	MRW	XLEN	Hardware perf. monitor event selector
scounteren	106	SRW	32	Supervisor counter enable
cycle/h	C00 / C80	URO	64	Clock cycle counter (read only)
time/h	C01 / C81	URO	64	Hardware timer (read only)
instret/h	C02 / C82	URO	64	Instruction retired counter (read only)
hpmcounter3/h … hpmcounter31/h	C03 / C83 … C1F / C9F	URO	64	Other hardware performance monitor counters (read only)

status Register

- **Most complex CSR**
  - mstatus or sstatus (when referring to either, we use status)
  - mstatus can access all bits of status register
  - sstatus can only access a subset of bits of status register
- **64-bit CSR**
  - **RV64:** status (64-bit register)
  - **RV32:** status/h (two 32-bit registers)

RV64 mstatus:

mstatus

SD	WPRI	MBE	SBE	SXL[1:0]	UXL[1:0]
63	62:38	37	36	35:34	33:32

WPRI	TSR	TW	TVM	MXR	SUM	MPRV	XS[1:0]	FS[1:0]	MPP[1:0]	VS[1:0]	SPP	MPIE	UBE	SPIE	WPRI	MIE	WPRI	SIE	WPRI
31:23	22	21	20	19	18	17	16:15	14:13	12:11	10:9	8	7	6	5	4	3	2	1	0

RV64 vs. RV32 mstatus Register

Similar, but **SD** (state dirty) in bit 31 of mstatus, for easy access, and SXL and UXL not supported.

RV64 mstatus:

mstatus

SD	WPRI	MBE	SBE	SXL[1:0]	UXL[1:0]
63	62:38	37	36	35:34	33:32

WPRI	TSR	TW	TVM	MXR	SUM	MPRV	XS[1:0]	FS[1:0]	MPP[1:0]	VS[1:0]	SPP	MPIE	UBE	SPIE	WPRI	MIE	WPRI	SIE	WPRI
31:23	22	21	20	19	18	17	16:15	14:13	12:11	10:9	8	7	6	5	4	3	2	1	0

RV32 mstatus/h:

mstatush

WPRI	MBE	SBE	WPRI
31:6	5	4	3:0

mstatus

SD	WPRI	TSR	TW	TVM	MXR	SUM	MPRV	XS[1:0]	FS[1:0]	MPP[1:0]	VS[1:0]	SPP	MPIE	UBE	SPIE	WPRI	MIE	WPRI	SIE	WPRI
31	30:23	22	21	20	19	18	17	16:15	14:13	12:11	10:9	8	7	6	5	4	3	2	1	0

mstatus vs. sstatus Registers

- **sstatus:** MIE, MPIE, MPP, MPRV, TVM, TW, TSR, SXL, SBE, MBE inaccessible
- Other fields the same as mstatus
- RV32 sstatus/h: SD moved to sstatus[31] and no SXL/UXL bits

RV64 mstatus:

mstatus

SD	WPRI	MBE	SBE	SXL[1:0]	UXL[1:0]
63	62:38	37	36	35:34	33:32

WPRI	TSR	TW	TVM	MXR	SUM	MPRV	XS[1:0]	FS[1:0]	MPP[1:0]	VS[1:0]	SPP	MPIE	UBE	SPIE	WPRI	MIE	WPRI	SIE	WPRI
31:23	22	21	20	19	18	17	16:15	14:13	12:11	10:9	8	7	6	5	4	3	2	1	0

RV32 sstatus/h:

sstatus

SD	WPRI	UXL[1:0]
63	62:34	33:32

WPRI	MXR	SUM	WPRI	XS[1:0]	FS[1:0]	WPRI	VS[1:0]	SPP	WPRI	UBE	SPIE	WPRI	SIE	WPRI
31:20	19	18	17	16:15	14:13	12:11	10:9	8	7	6	5	4:2	1	0

`status` Register Fields

- **`MPP/SPP:`** previous privilege
  - `mret` restores privilege mode to value saved in `MPP`
  - `sret` restores privilege mode to value saved in `SPP`
- **`MIE/SIE:`** global interrupt enables
- **`MPIE/SPIE:`** previous global interrupt enables
  - When trap to M-mode occurs:
    - `MIE` = `MPIE`
    - `MIE` = `0`
  - When trap returns (i.e., `mret` executes):
    - `MIE` = `MPIE`
- **`MBE/SBE/UBE:`** Endianness of loads/stores
  - 0 = little-endian, 1 = big-endian
  - Instructions are always little-endian

?PP	Mode
00	User
01	Supervisor
11	Machine

`status` Register Fields, cont'd

- **FS/VS/XS/SD:** Floating-Point/Vector/Extension/State Status
  - Indicates whether need to store state on context switch
  - XS is for user-defined extension
  - SD = FS | VS | XS
- **UXL/SXL:** User/Supervisor XLEN
  - Allows processor to emulate RV32
  - Hardwired to 10 (XLEN = 64 bits) for many processors
  - M-mode XLEN encoded in `misa` register

?S	Meaning	Description
00	Off	All off
01	Initial	None dirty or clean, some on
10	Clean	None dirty, some clean
11	Dirty	Some dirty

?XL	XLEN
01	32
10	64
11	128

misa Register (Machine ISA)

Bit	Character	Description	Chapter
0	A	Atomic	14
1	B	Bit Manipulation	
2	C	Compressed	11
3	D	Double-Precision Floating-Point	13
4	E	RV32E Embedded ISA	1.2.7
5	F	Single-Precision Floating-Point	13
6	G	Base ISA and MAFD extensions	
7	H	Hypervisor	
8	I	RV32I/64I/128I Base ISA	
9	J*	Dynamically-Translated Languages	
10	K	Cryptography	
11	L*	Decimal Floating-Point	
12	M*	Integer Multiply/Divide	
13	N	User-Level Interrupts	
14	O	Currently Unused	
15	P	Packed SIMD	

* Not ratified

misa Register (Machine ISA), cont'd

Bit	Character	Description	Chapter
16	Q	Quad-Precision Floating-Point	13
17	R	Currently Unused	
18	S	Supervisor Mode	
19	T*	Transactional Memory	
20	U	User Mode	
21	V	Vector	
22	W	Currently Unused	
23	X	Non-Standard Extensions	
24	Y	Currently Unused	
25	Z	Currently Unused	
XLEN-1: XLEN-2	MXL	Machine XLEN See Table 5.4	

* Not ratified

Other Extensions

Name	Extension
Zicsr	Control and Status Registers
Zifencei	Instruction-Fetch Fence
Zfh, Zfhmin	Half-Precision Floating-Point
Svnapot, Svpbmt, Svinval	Memory extensions
Zicbom, Zicbop, Zicboz	Cache management extensions

Accessible using structure pointed to by `mconfigptr` CSR

Chapter 8: Privileged Operations

Performance Counters

Performance Counters

- **32 counters** (hardware performance monitor (hpm) counters):
  - **0-2:** mcycle/h, mtime/h, minstret/h (# cycles, sys. time, # instr. retired)
  - **3-31:** mhpmcounters3/h – mhpmcounter31/h
- **Enables for S-/U-mode:**
  - `mcounteren`/`scounteren`: turn off access from S-/U-mode, respectively
- **Indicate what (i.e., the event) to count:**
  - **mhpmevent3 - mhpmevent31** (machine hpm event)
- **32-bit `mcounthinibit`:**
  - disable unused counters to reduce energy consumption
- **Read-only views of performance counters (for S/U-modes):**
  - **`cycle/h`, `time/h`, `instret/h`, `hpmcounter3/h` – `hpmcounter31/h`**

Accessing Performance Counters

- **Read cycle counter:** `rdcycle rd`
  - Equivalent instruction: `csrr rd, cycle`
- **Read system time:** `rdtime rd`
- **Read # of instructions retired:** `rdinstret rd`
- **In RV32, for reading upper 32 bits:**
  - `rdcycleh`, `rdtimeh`, `rdinstreth`

CLINT & PLIC

- **CLINT:** Core Local Interruptor
  - Contains:
    - System timer (`time`)
    - Timer compare registers
    - Software interrupt registers (per hart)
  - Returns `time` register
- **PLIC:** Platform-Level Interrupt Controller
  - Prioritizes and routes interrupts from other peripherals to external machine/supervisor interrupt inputs
  - Peripherals are accessed using memory-mapped I/O (not CSRs)
- More details in Chapter 15

Chapter 8: Privileged Operations

Reset Behavior

Reset Behavior

- On reset, typically **PC = 0x80000000** or **0x1000**
  - implementation-specific
- **`mcause` = cause** of reset (or 0 if system doesn't indicate)
- **`mstatus.MIE` = 0** (turn off global interrupts)
- **`mstatus.MPRV` = 00** (set to U-mode)
- **`mstatus.A` = `mstatus.L` = 0** (disable PMP registers)
- Other hart state undefined
  - So, system must initialize all registers and CSRs before using them

Chapter 8: Privileged Operations

Traps

Traps

- **Types of traps:**
  - **Exceptions:** caused by instructions (some of these are also called **faults**)
  - **Interrupts:** caused by external events

RISC-V exceptions are precise:

- All instructions prior to excepting instruction complete
- Neither excepting instruction nor subsequent instructions change state of system before exception is handled

What Happens when Trap Occurs

- When trap occurs to mode *x* (*m*achine or *s*upervisor)
  - ***x*cause** = cause of trap (msb = 1 for interrupt, 0 for exception)
  - ***x*epc** = PC of instruction executing when trap occurs
  - ***x*tval** = Extra information about trap (or 0)
  - Then, PC = ***x*tvec** (trap handler vector (i.e., address))

Trap Cause and Value

Trap	Cause (*xcause*)		Value (*xtval*)
	Interrupt Bit (msb)	Trap Code (lsbs)	
S-mode software interrupt	1	1	0
M-mode software interrupt	1	3	0
S-mode timer interrupt	1	5	0
M-mode timer interrupt	1	7	0
S-mode external interrupt	1	9	0
M-mode external interrupt	1	11	0
Instruction address misaligned	0	0	Target Address
Instruction access fault	0	1	PC
Illegal instruction	0	2	Instruction
Breakpoint	0	3	PC
Load address misaligned	0	4	Load Address
Load access fault	0	5	Load Address
Store/AMO address misaligned	0	6	Store Address
Store/AMO access fault	0	7	Store Address
`ecall` from U-mode	0	8	0 (a0 = arg)
`ecall` from S-mode	0	9	0 (a0 = arg)
`ecall` from M-mode	0	11	0 (a0 = arg)
Instruction page fault	0	12	PC
Load page fault	0	13	Load Address
Store/AMO page fault	0	15	Store Address

Traps

- **Horizontal trap:**
  - Traps that are handled in same privilege mode where they occur
- **Vertical trap:**
  - Traps to higher privilege level
- **Traps never go to a lower privilege level**

Traps

- **When trap to M-mode occurs:**
  - `mstatus.MPP` (machine previous privilege) = current privilege mode
  - `mstatus.MPIE` (machine previous interrupt enable) = current interrupt enable)
    - So that privilege & interrupt enable can be restored at end of trap handler
  - `mstatus.MIE` = 0 (so the trap won't be interrupted again)
  - Privilege mode = 11 (M-mode)
- **When trap to S-mode occurs:**
  - Same as M-mode, but with `mstatus.SPP`, `SPIE`, and `SIE`, and privilege mode = 01 (S-mode)

`mstatus.MIE/SIE`: Global interrupt enable (1 = enabled)
`mie/sie` registers: Per-trap enable bits

CSRs used by Traps

CSR	Description
`mstatus`	Interrupt enable (`MIE`), previous interrupt enable (`MPIE`), previous privilege mode (`MPP`) bits
`mcause`	Cause of interrupt or exception
`mtval`	Value that gives extra trap information
`mtvec`	Address (also called vector) of trap handler
`mepc`	Exception PC, holding return address from trap
`mscratch`	Typically holds the exception stack pointer
`mip`	Interrupt pending that lists which interrupts are currently pending
`mie`	Interrupt enable: lists which interrupts can be taken or should be ignored

Exception Types

- Illegal instructions
- Access faults
- Address misaligned
- `ecall` or `ebreak`
- Page faults

Exception Types, cont'd

- **Illegal instructions:**
  - Instruction doesn't exist, or
  - Instruction isn't available in current privilege mode
  - **Examples:**
    - `mret` in lower privilege levels
    - Accessing a privileged CSR from lower privilege level
    - Etc.
- **Access faults:**
  - When hart tries to access inaccessible memory
  - **Examples:**
    - Non-existent memory
    - Writing to a ROM
    - Word access to peripheral that only accesses bytes
  - PMA (physical memory attributes) unit: define accessible regions
  - PMP (physical memory protection) unit: checks for allowed accesses

Exception Types, cont'd

- **Address misaligned:**
  - **Example:** `lw` to 0xB2
  - **Handler may:** kill program or issue two aligned loads and shift, mask, and merge to deliver misaligned result
- **`ecall` or `ebreak`:**
  - `ecall`: OS (system) calls
  - `ebreak`: invoke debugger
- **Page faults**
  - Attempt to access virtual page that is not in physical memory (see Chapter 8)

Interrupt Types

- **External**
  - **Example:** peripherals indicating event has occurred (e.g., data ready from a serial port)
- **Timer**
  - **Example:** timer exceeds preconfigured value (in `mtimecmp`)
  - **Example:** timer exceeds preconfigured value (in `stimecmp`) – included in Sstc extension
- **Software**
  - Produced to communicate between harts
  - **Example:** writing a memory-mapped bit in the CLINT

Interrupt Registers

Interrupt registers:

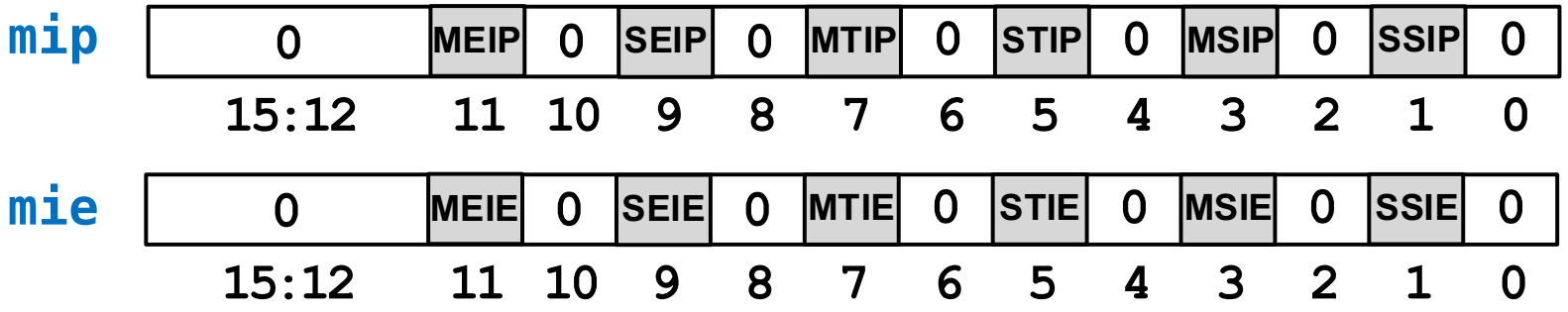
- **`mip`** (machine interrupt pending)
- **`mie`** (machine interrupt enable)

M-mode interrupts (bits of `mip` and `mie`):

- **`MEIP/MEIE`**: M-mode **external** interrupt pending/enable
- **`MTIP/MTIE`**: M-mode **timer** interrupt pending/enable
- **`MSIP/MSIE`**: M-mode **software** interrupt pending/enable

S-mode interrupts (bits of `mip` and `mie`):

- **`SEIP/SEIE`**: S-mode **external** interrupt pending/enable
- **`STIP/STIE`**: S-mode **timer** interrupt pending/enable
- **`SSIP/SSIE`**: S-mode **software** interrupt pending/enable



Chapter 8: Privileged Operations

Trap Delegation

Trap Delegation

- By default, **traps elevate privilege to M-mode** (and are handled in M-mode)
- Traps can be **delegated to S-mode** using registers:
  - `medeleg`: machine exception delegation register
  - `mideleg`: machine interrupt delegation register
  - Setting a bit in these registers delegates the trap to S-mode

- By setting corresponding bit, trap occurring in S-mode is handled by S-mode (not M-mode) handler
- But traps never reduce privilege levels.
  - **Example:** if code running in M-mode encounters illegal instruction, M-mode trap occurs (even if corresponding delegation bit, `medeleg` bit 2, is set)

Trap Delegation

- `medeleg` and `mideleg` bits correspond to number of cause:

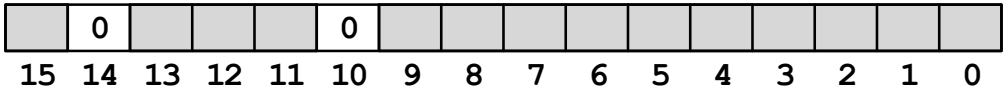
Exception	Cause
Instruction address misaligned	0
Instruction access fault	1
Illegal instruction	2
Breakpoint	3
Load address misaligned	4
Load access fault	5
Store/AMO address misaligned	6
Store/AMO access fault	7
ecall from U-mode	8
ecall from S-mode	9
ecall from M-mode	11
Instruction page fault	12
Load page fault	13
Store/AMO page fault	15

Interrupt (abbreviation)	Cause
S-mode software interrupt (SSI)	1
M-mode software interrupt (MSI)	3
S-mode timer interrupt (STI)	5
M-mode timer interrupt (MTI)	7
S-mode external interrupt (SEI)	9
M-mode external interrupt (MEI	11

`mideleg`

0	MEI	0	SEI	0	MTI	0	STI	0	MSI	0	SSI	0
15:12	11	10	9	8	7	6	5	4	3	2	1	0

`medeleg`



Examples of Trap Delegation

- **Examples** of delegating traps to S-mode (so that the OS can handle them):
  - Load page fault (exception: cause 13)
  - U-mode OS call (ecall) (exception: cause 8)
  - Breakpoint (exception: cause 3)

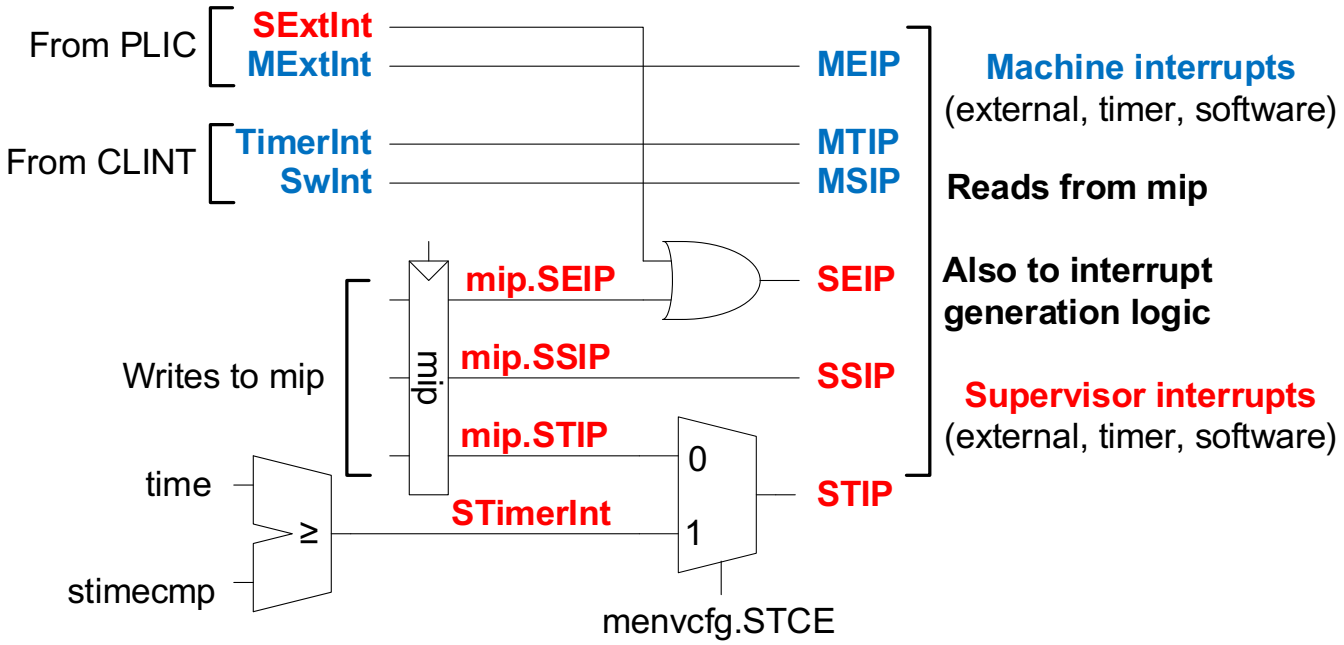
- If these traps **weren't delegated to S-mode**:
  - the **M-mode trap handler** would have to pass them along to the OS.
  - So, delegation potentially **decreases execution time**.

Chapter 8: Privileged Operations

Trap Hardware

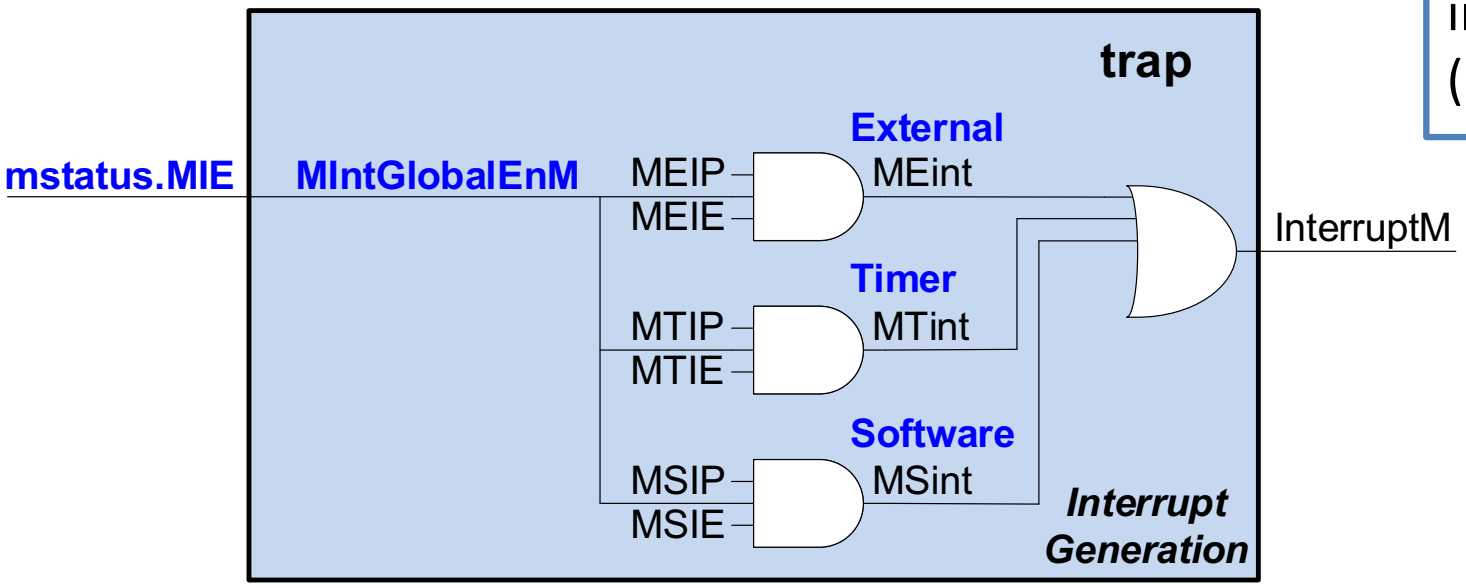
Interrupt Sources

- **Interrupts triggered by:**
  - Peripherals (external, software, or timer)
  - Writing bits into in `mip`
- **M-mode interrupts** triggered by **MExtInt** (from PLIC) or **TimerInt**, or **SwInt** (from CLINT)
- **S-mode interrupts** triggered by **SExtInt** or **writes to mip**



Interrupt Generation: M-Mode

Interrupts when only **M-mode** available:



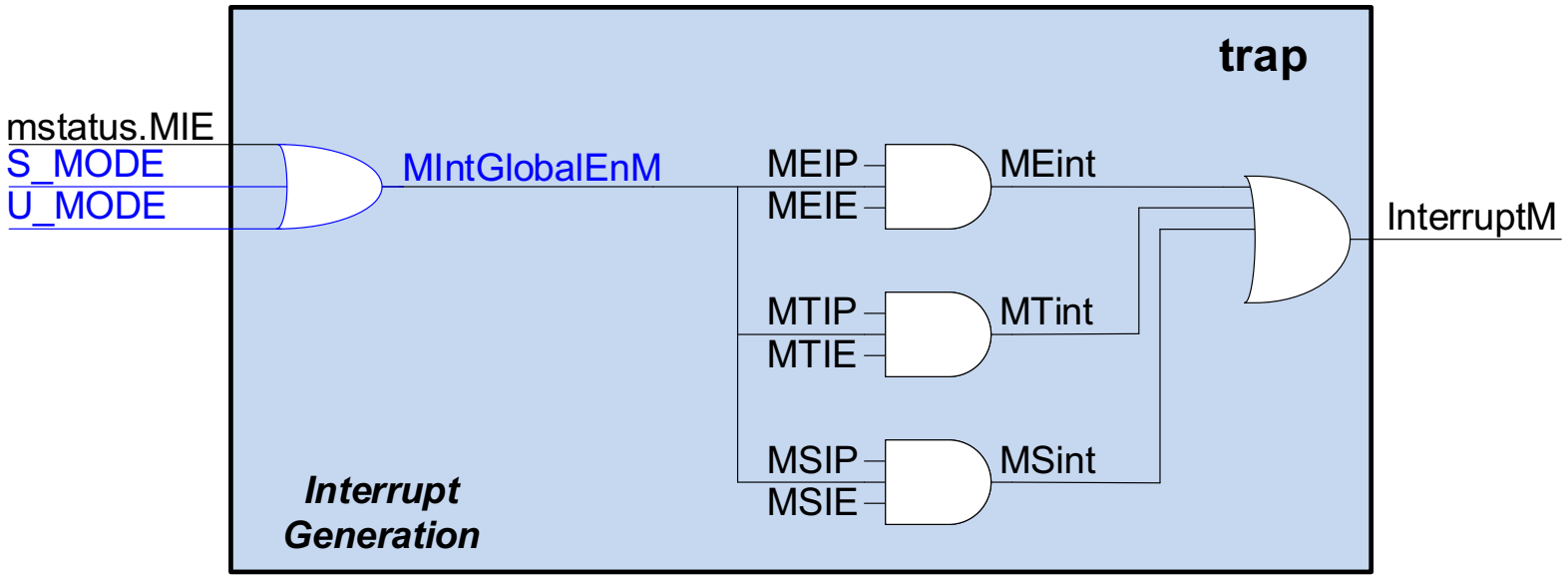
Interrupts occur in **Memory stage** (postfix M).

Interrupt occurs when:

- Global interrupts enabled: (`MIE`) AND
- Specific interrupt pending: (`M?IP`) AND
- Specific interrupt enabled: (`M?IE`)

Interrupt Generation: M/S/U-Mode

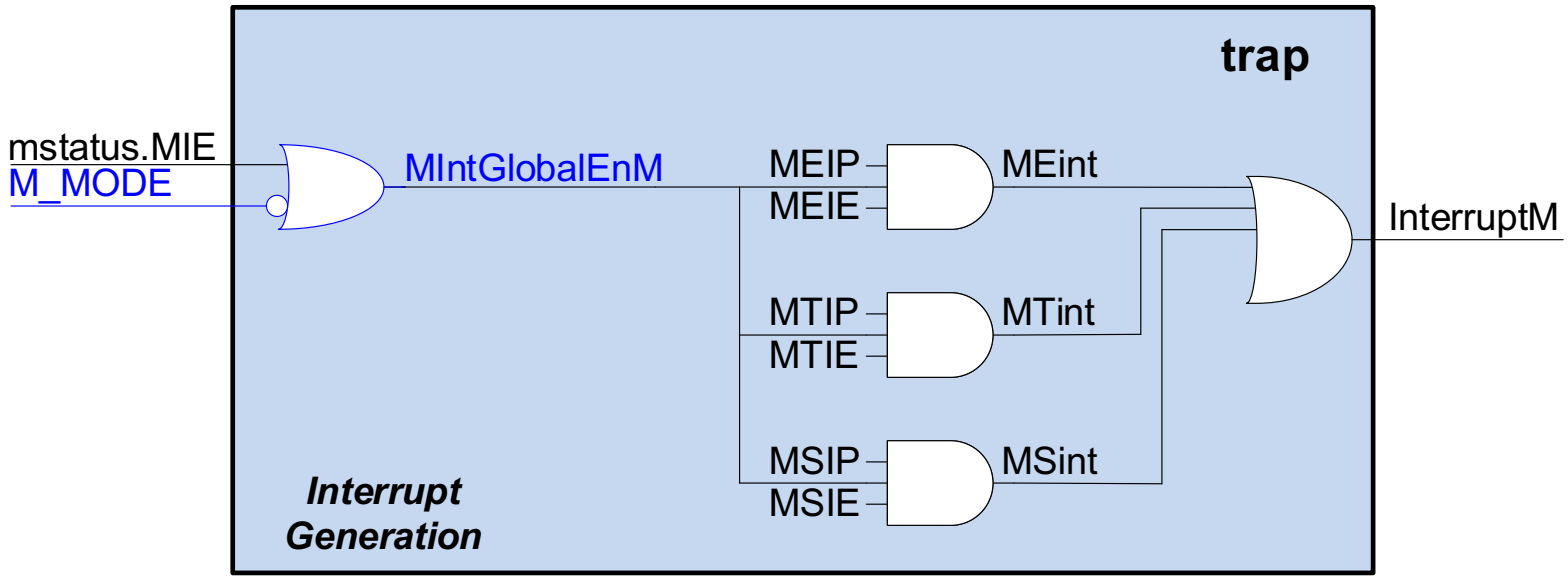
Interrupts when **M/S/U-modes** available (but no delegation):



- Global interrupts enabled: (`MIE | S_MODE | U_MODE`) AND
- Specific interrupt pending: (`M?IP`) AND
- Specific interrupt enabled: (`M?IE`)

Interrupt Generation: M/S/U-Mode

Interrupts when **M/S/U-modes** available (but no delegation):

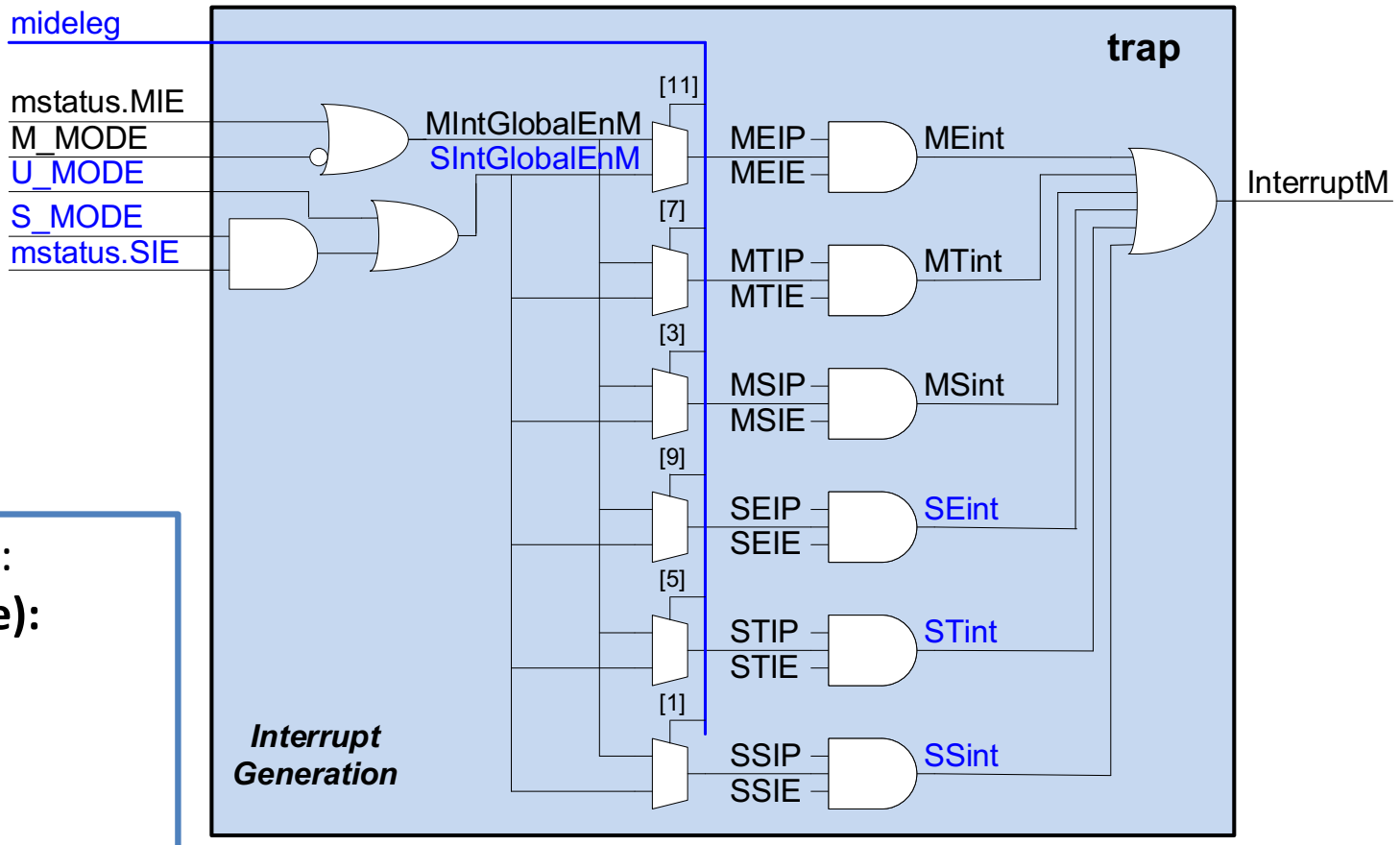


- Global interrupts enabled: (`MIE | ~M_MODE`) AND
- Specific interrupt pending: (`M?IP`) AND
- Specific interrupt enabled: (`M?IE`)

`S_MODE | U_MODE = ~M_MODE`

Interrupt Generation: Delegation

Interrupts when **M/S/U-modes** available **and delegation**:



Global interrupts enabled:

- **No delegation (as before):**

```
MIntGlobalEnM =
(MIE | ~M_MODE)
```

- **Delegation (bit set in `mideleg`):**

```
SIntGlobalEnM =
(SIE & S_MODE | U_MODE)
```

Unvectored vs. Vectored Interrupts

Unvectored Interrupts (*x*tvec[1:0] = 00):

- **`mtvec`:** machine trap vector (address)
  - Where processor jumps to on M-mode trap
- **`stvec`:** supervisor trap vector (address)
  - Where processor jumps to on S-mode trap
- Jumps to these addresses independent of trap type
  - Handler looks at ***`x`*`cause`** and ***`x`*`tval`** to figure out cause of trap
- ***`x`*`tvec`** must be multiple of 4, so bottom 2 bits ignored

Vectored Interrupts (*x*tvec[1:0] = 01):

- Faster - **jumps to different address** depending on trap type
- Exceptions still jump to trap vector (bottom 2 bits ignored)
- **Interrupts** jump to **trap vector + (4 x cause)**
  - **Example:** machine software interrupt (MSI) `mcause` = 3
  - So, jumps to `mtvec` + (4 x 3) = `mtvec` + 12
- ***`x`*`tvec`** must be multiple of 64, so vector addition becomes concatenation

Only room for one instruction at each trap vector, so it is a **jump**.

Chapter 8: Privileged Operations

Trap Handler

Example Trap Handler: Operation

Example trap handler – procedure:

- **Handles two types of traps:**
  - timer interrupt:
    - Adds more time to timer compare register (`mtimecmp`) and returns (`mret`)
  - `ecall` (OS call):
    - if `a0` = 0-3, changes to that privilege mode (0 = user, 1 = supervisor, 3 = machine) by making `mstatus.MPP` = `a0`, then issuing `mret`
    - If `a0` = 4, terminates program

- **Writes to signature memory:**
  - For testing purposes, writes `mcause` and `mtval` to signature each time trap occurs.

Example Trap Handler Code (setup)

```
.EQU MTIME,    0x200bff8
.EQU MTIMECMP, 0x2004000

.global rvtest_entry_point

rvtest_entry_point:
    la sp, topofstack                   # Initialize stack pointer (not used)
    la s6, begin_signature              # s6 points to signature memory

    # Set up timer
    jal set_timecmp

    # Set up interrupts
    la t0, trap_handler
    csrw mtvec, t0                      # Initialize mtvec to trap_handler
    csrw mideleg, zero                  # Don't delegate interrupts
    csrw medeleg, zero                  # Don't delegate exceptions
    li t0, 0x080                        # mie bit 7 = machine timer interrupt
    csrw mie, t0                        # Enable machine timer interrupt
    la t0, topoftrapstack
    csrw mscratch, t0                   # mscratch holds trap stack pointer
    csrsi mstatus, 0x8                  # Turn on mstatus.MIE global interrupt enable
```

Example Trap Handler Code (main)

```
main:
    # Change to user mode
    li a0, 0                            # a0 = 0: argument to enter user mode
    ecall                               # System call (trap) to enter user mode

    # Wait for timer interrupts
    li t0, 0x1000                       # loop counter start value
loop:
    addi t0, t0, -1                     # Decrement counter
    bne t0, zero, loop                  # And repeat until zero

done:
    li a0, 4                            # a0 = 4: argument to terminate program
    ecall                               # system call to terminate program
    j self_loop                         # wait forever (not executed)
```

Example Trap Handler Code

```
set_timecmp:                        # Set timer compare to 800 ticks later
    la t0, MTIME
    la t1, MTIMECMP
    ld t0, 0(t0)                    # Read current timer
    addi t0, t0, 0x60               # Increment timer
    sd t0, 0(t1)                    # Set MTIMECMP = MTIME + 0x800

.align 2                            # Trap handlers must be aligned to
                                    # multiple of 2^2 = 4
trap_handler:                       # Load trap handler stack pointer, tp
    csrrw tp, mscratch, tp          # Swap mscratch and tp
    sd t0, 0(tp)                    # Save t0 and t1 on the stack
    sd t1, -8(tp)
    csrr t0, mcause                 # Check the cause
    csrr t1, mtval                  # And the trap value
    sd t0, 0(s6)                    # Write mcause and mtval to the signature
    sd t1, 8(s6)
    addi s6, s6, 16
    bgez t0, exception              # If msb of mcause = 0, it is an exception
interrupt:                          # Assume it is a timer interrupt
    jal set_timecmp                 # Incr. compare for next timer interrupt
    j trap_return                   # Clean up and return
exception:
    csrr t1, mepc                   # Add 4 to mepc so exception returns to
    addi t1, t1, 4                  # 1 instr. after the excepting instruction
    csrw mepc, t1
    li t1, 8                        # Is it an ecall trap?
    andi t0, t0, 0xFC               # If mcause = 8, 9, or 11, it is an ecall
    bne t0, t1, trap_return         # Ignore other exceptions
```

Example Trap Handler Code

```
ecall:
    li t0, 4
    beq a0, t0, write_tohost          # a0 = 4: terminate program
    bltu a0, t0, changeprivilege      # a0 = 0-3: change privilege level
    j trap_return                     # Ignore other ecalls

changeprivilege:
    li t0, 0x00001800                 # Mask off mstatus.MPP in bits 11-12
    csrc mstatus, t0
    andi a0, a0, 0x003                # Only keep bottom two bits of argument
    slli a0, a0, 11                   # Move into mstatus.MPP position
    csrs mstatus, a0                  # Set mstatus.MPP with desired privilege

trap_return:                          # Return from trap handler
    ld t1, -8(tp)                     # Restore t1 and t0
    ld t0, 0(tp)
    csrrw tp, mscratch, tp            # Restore tp
    mret                              # Return from trap

write_tohost:                         # Terminate program
    la t1, tohost
    li t0, 1                          # 1 for success (3 is for failure)
    sd t0, 0(t1)                      # Send success code

self_loop:
    j self_loop                       # Wait - infinite loop
```

Example Trap Handler Code (finish)

```
.section .tohost
tohost:                                 # Write to HTIF (host target interface) to
    .dword 0                            # terminate program
fromhost:
    .dword 0

.EQU XLEN, 64
begin_signature:
    .fill 6*(XLEN/32),4,0xdeadbeef
end_signature:

# Initialize stack with room for 512 bytes each for program and trap handler
.bss
    .space 512
topofstack:
# And another stack for the trap handler
.bss
    .space 512
topoftrapstack:
```

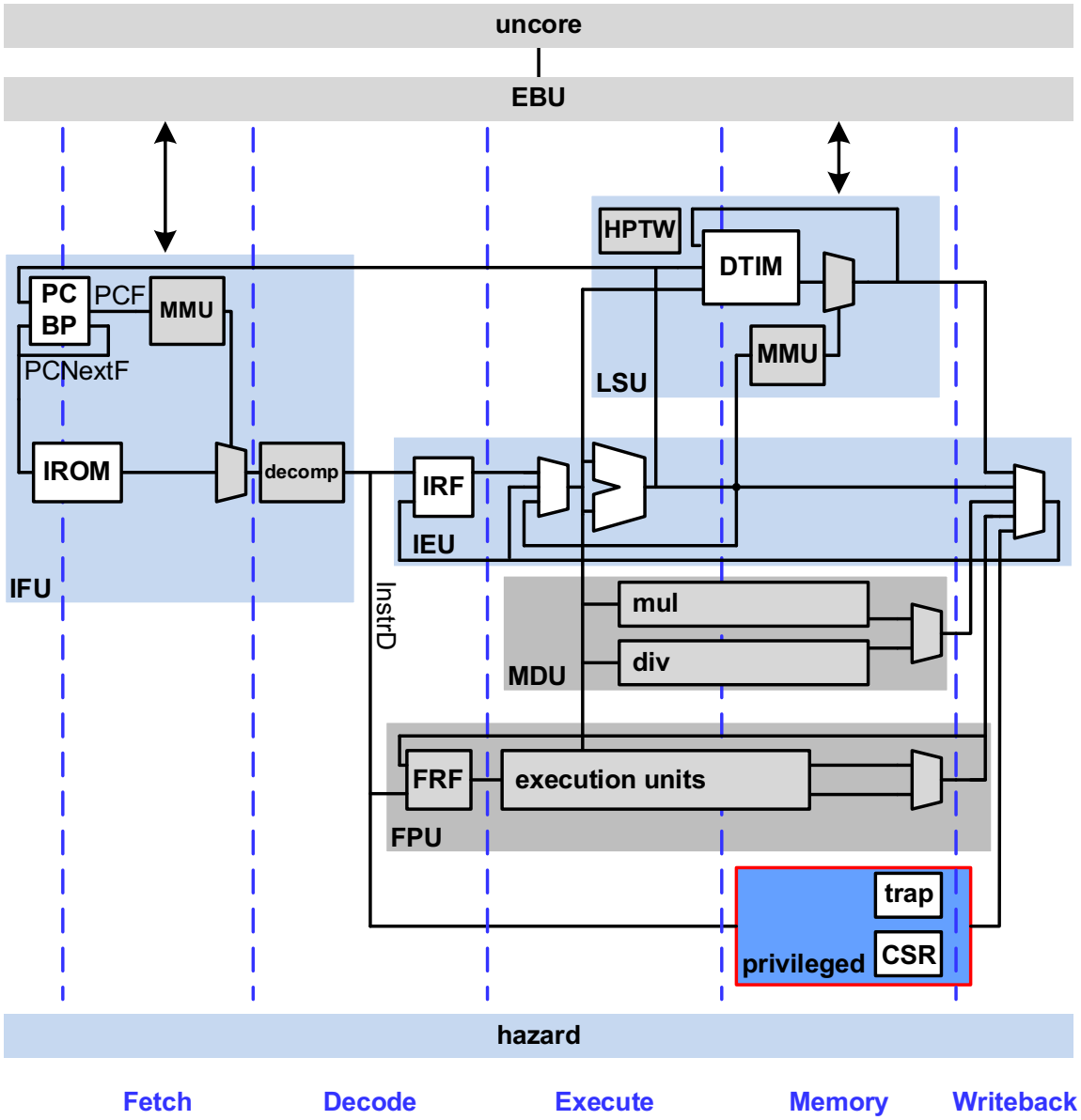
Chapter 8: Privileged Operations

Wally Implementation

Chapter 8: Privileged Operations

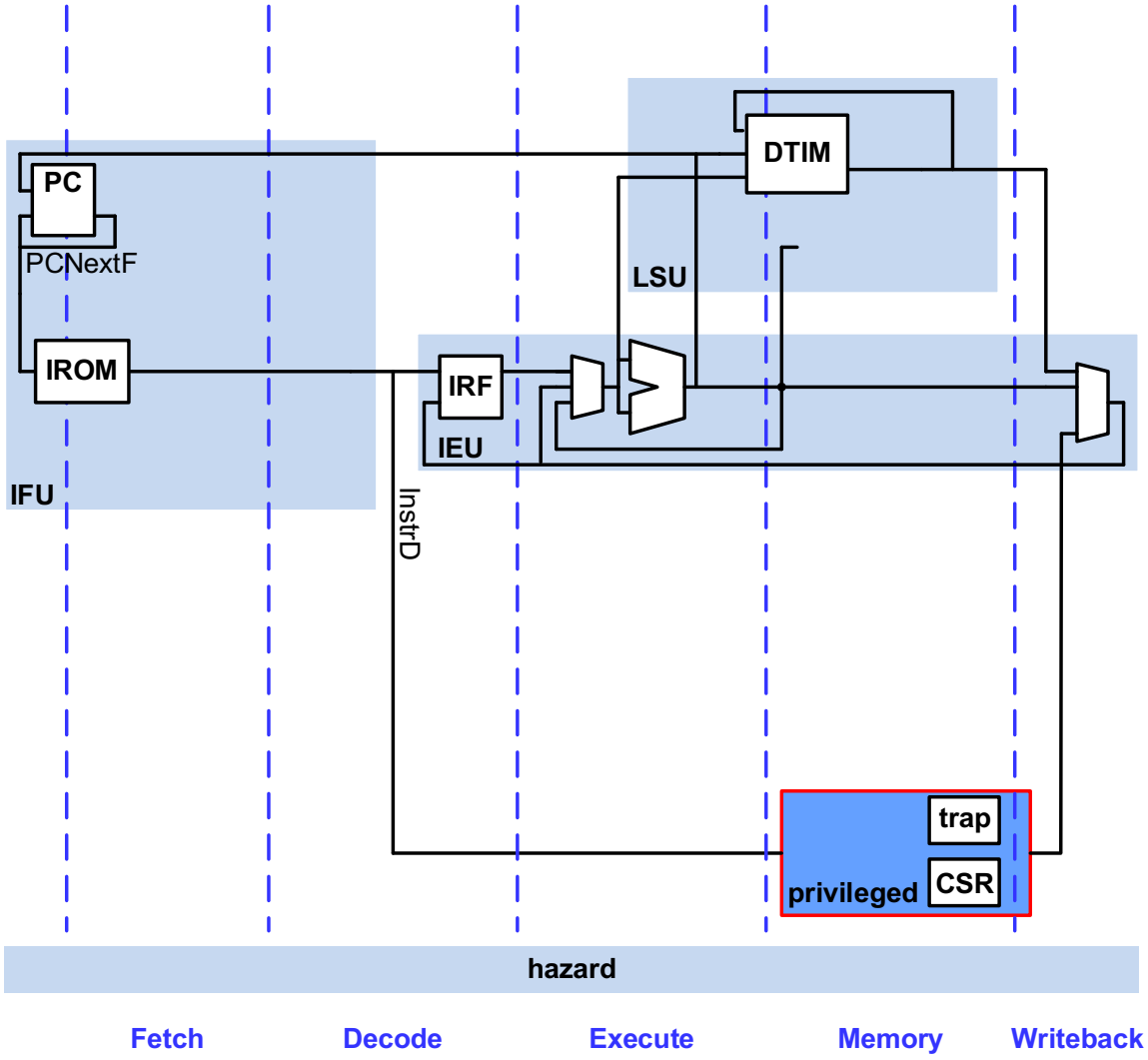
Privileged Unit

Pipelined Core with Privileged Unit

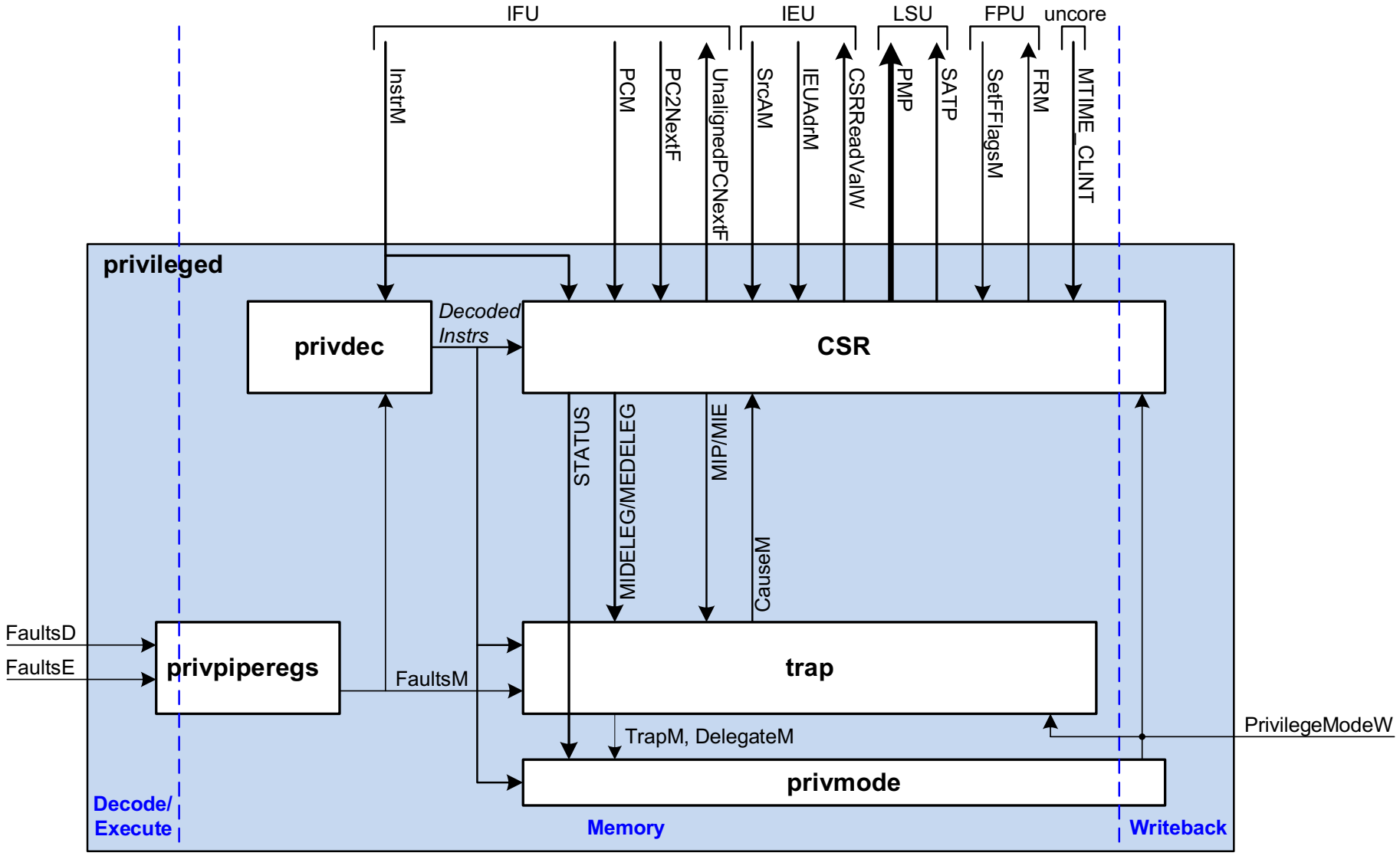


Simplified Pipelined Core with PRIV

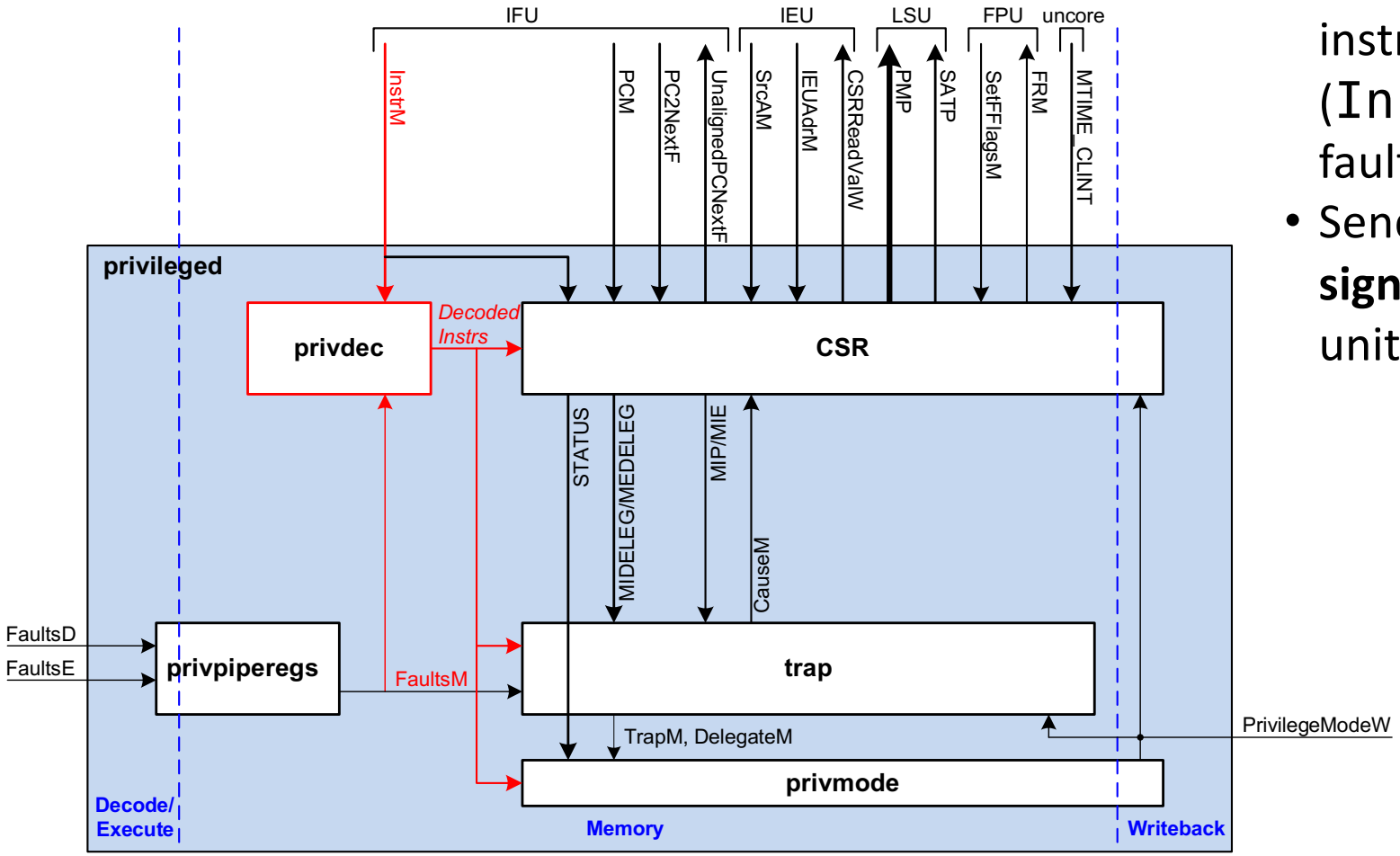
Privileged unit includes: **CSRs** and **trap** handler hardware.



Privileged Unit (`privileged`)



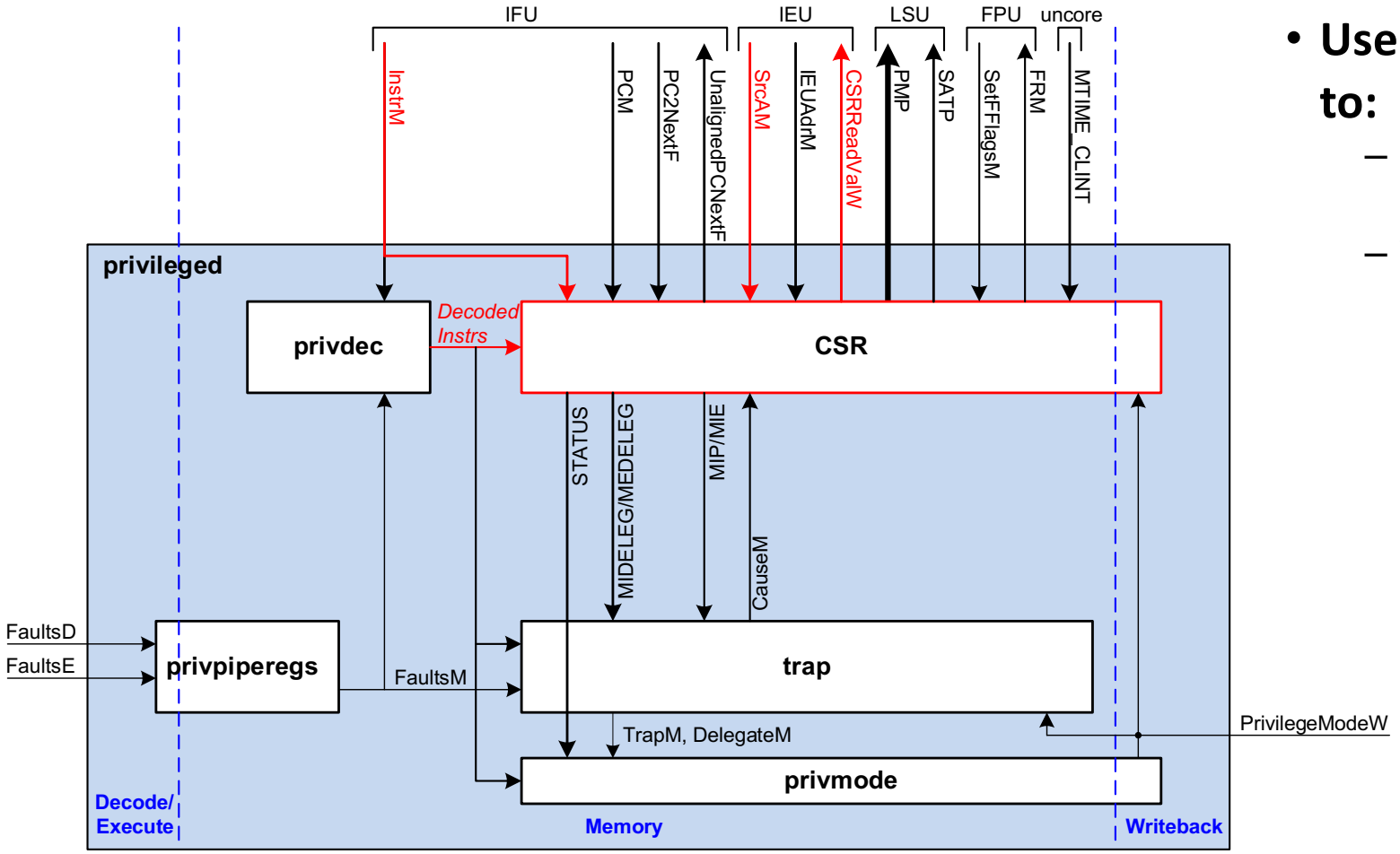
Privileged Decoder (privdec)



privdec:

- **Decodes** instruction (`InstrM`) and faults (`FaultsM`)
- Sends **control signals** to other units

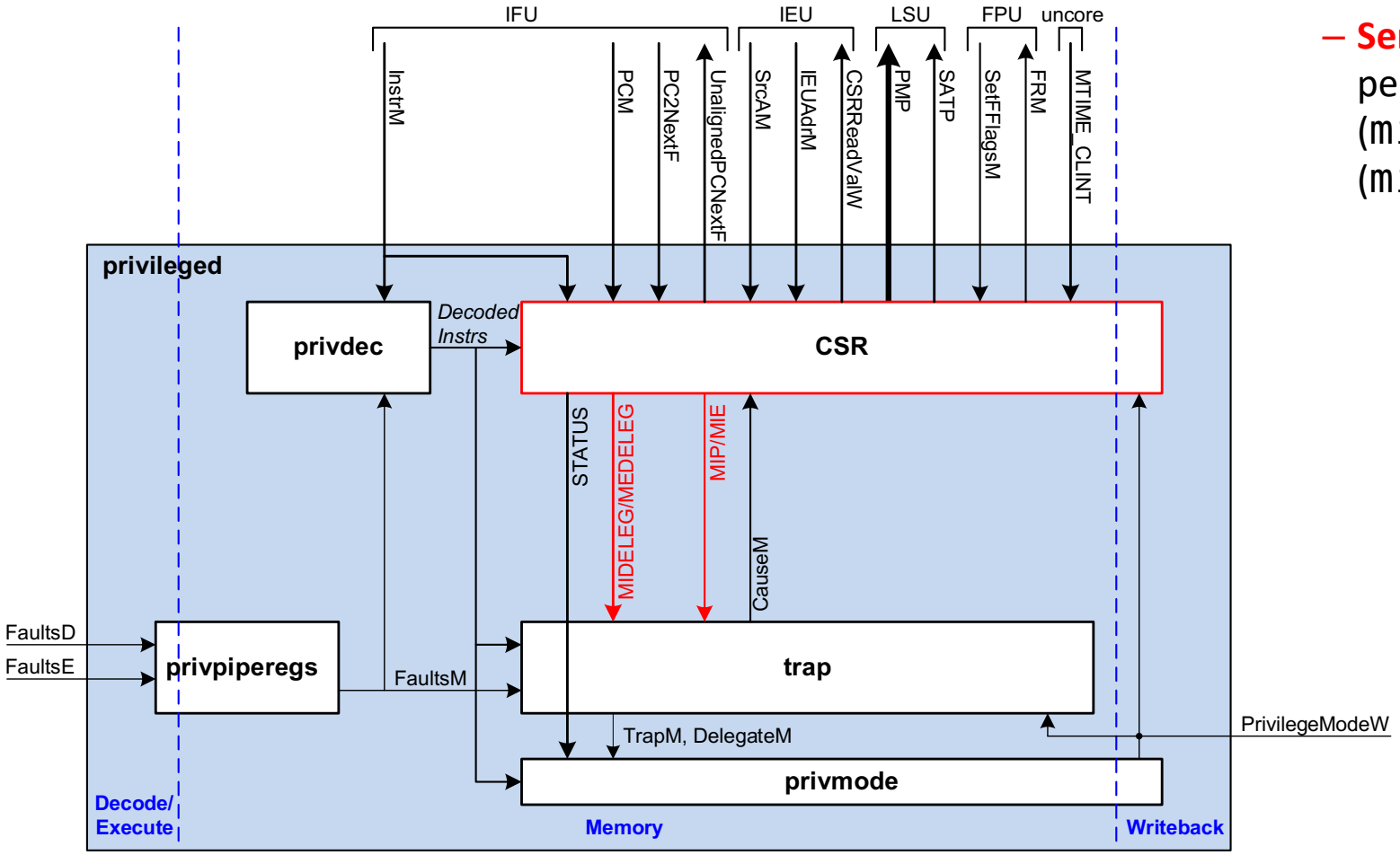
CSR Unit (csr)



csr:

- **Contains CSRs**
- **Uses instruction to:**
  - **Read CSRs:** `CSRReadValW`
  - **Write CSRs:** `SrcAM`

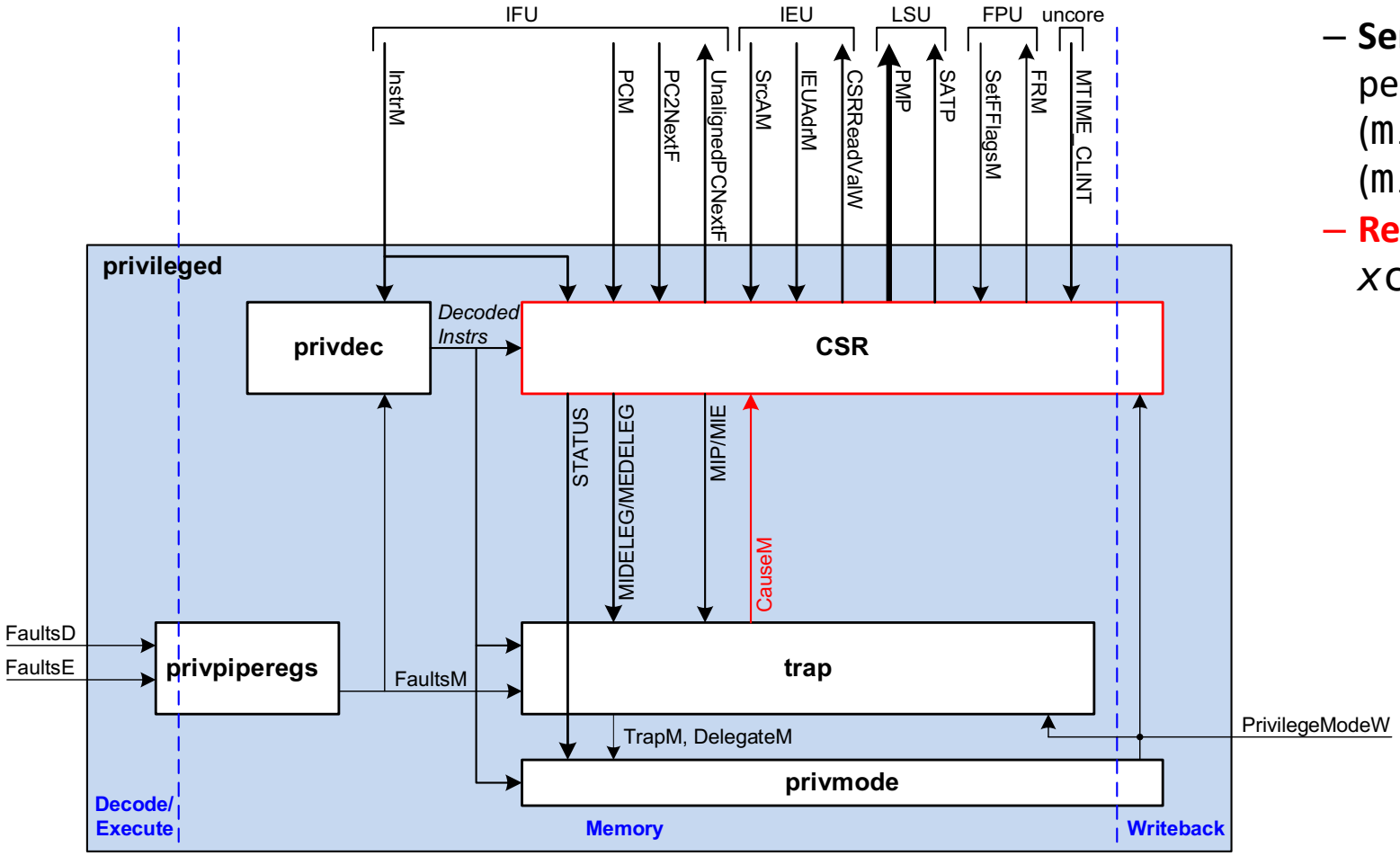
CSR Unit (csr)



csr:

- **During traps:**
  - **Sends to `trap`** pending/enable info (`mip`/`mie`) & deleg. info (`mideleg`/`medeleg`)

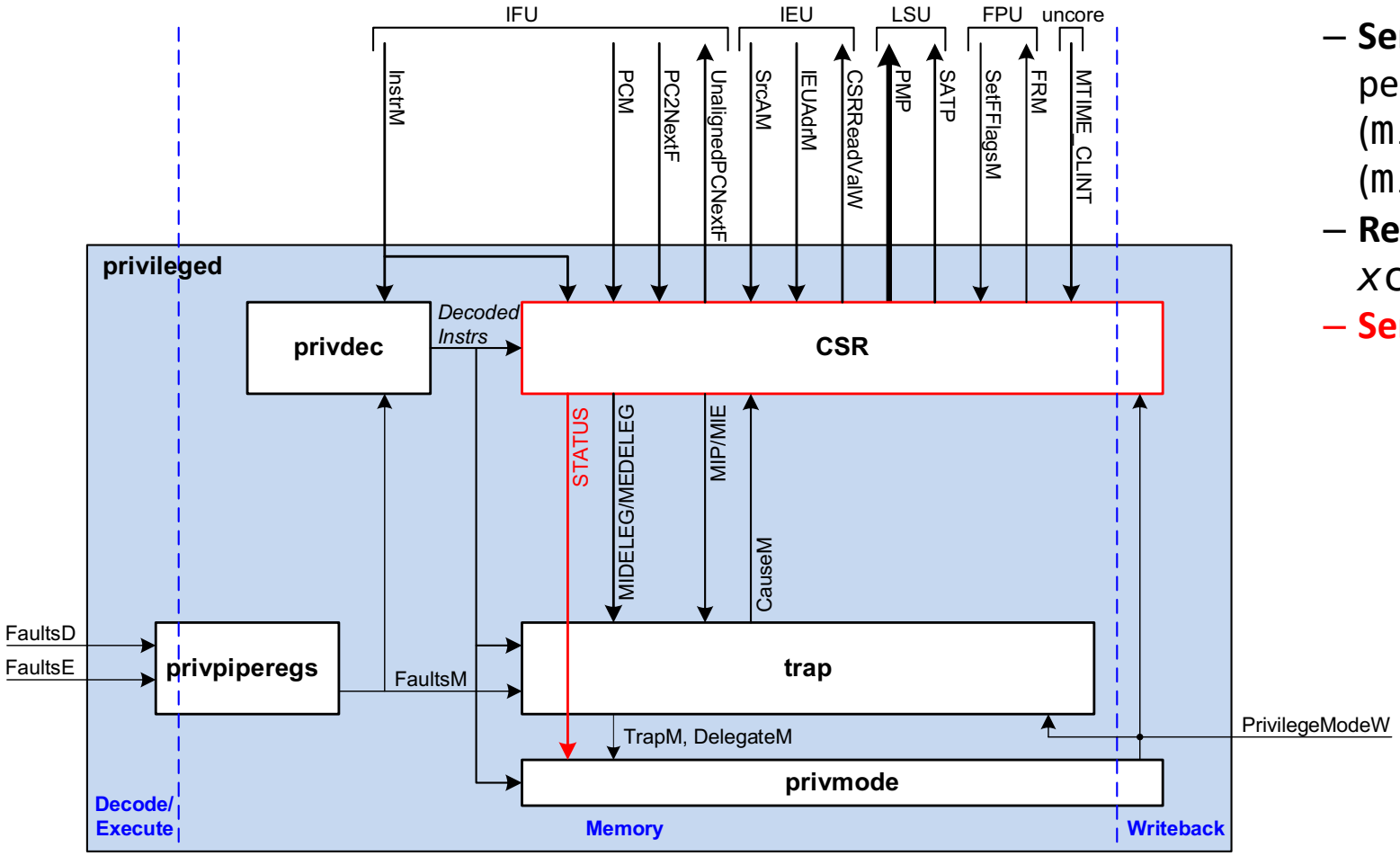
CSR Unit (csr)



csr:

- **During traps:**
  - **Sends to `trap`** pending/enable info (`mip`/`mie`) & deleg. info (`mideleg`/`medeleg`)
  - **Reads from `trap`:** *x*`cause` (`CauseM`)

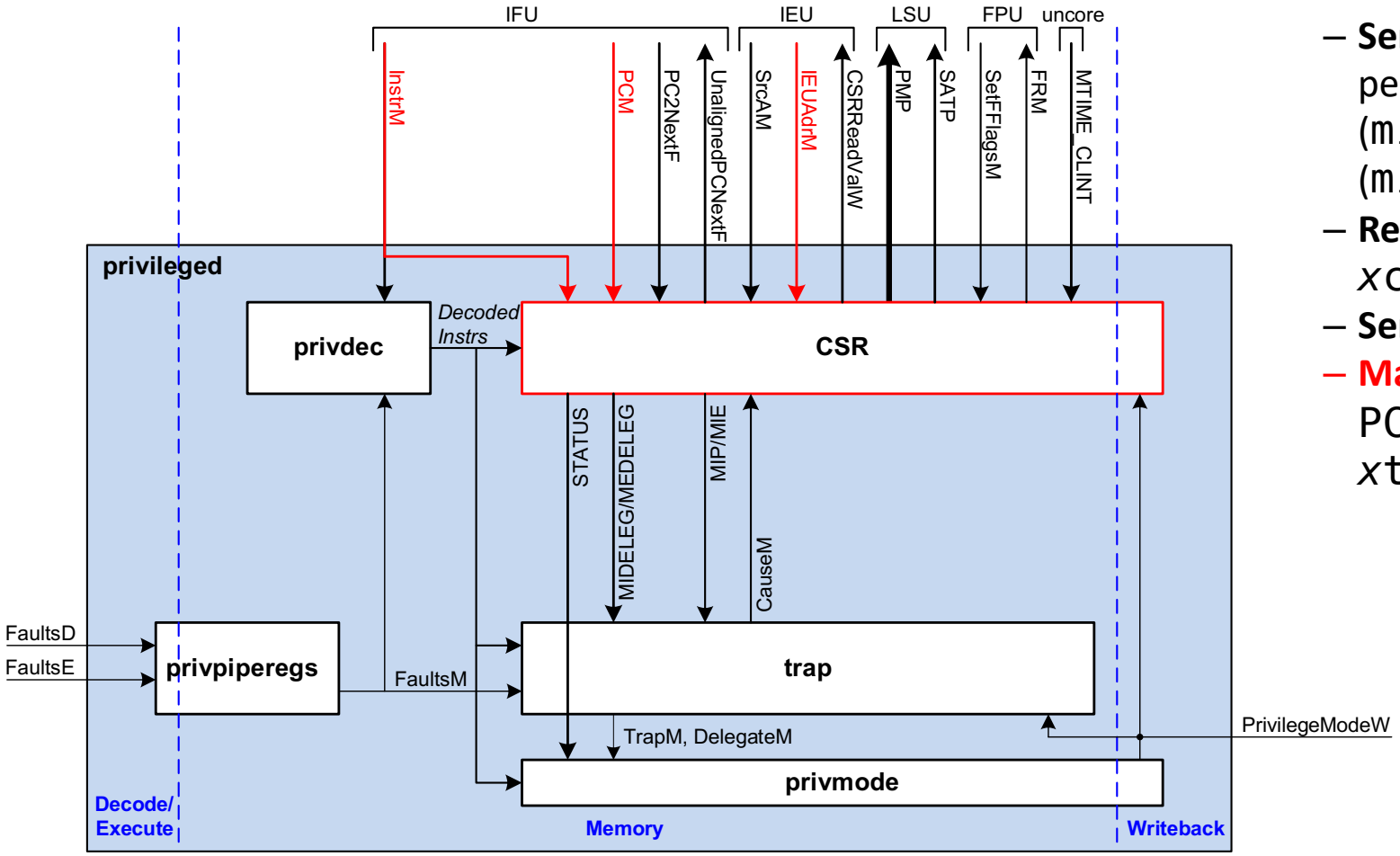
CSR Unit (csr)



csr:

- **During traps:**
  - **Sends to `trap`** pending/enable info (`mip`/`mie`) & deleg. info (`mideleg`/`medeleg`)
  - **Reads from `trap`:** *x*`cause` (CauseM)
  - **Sends** *x*`status`

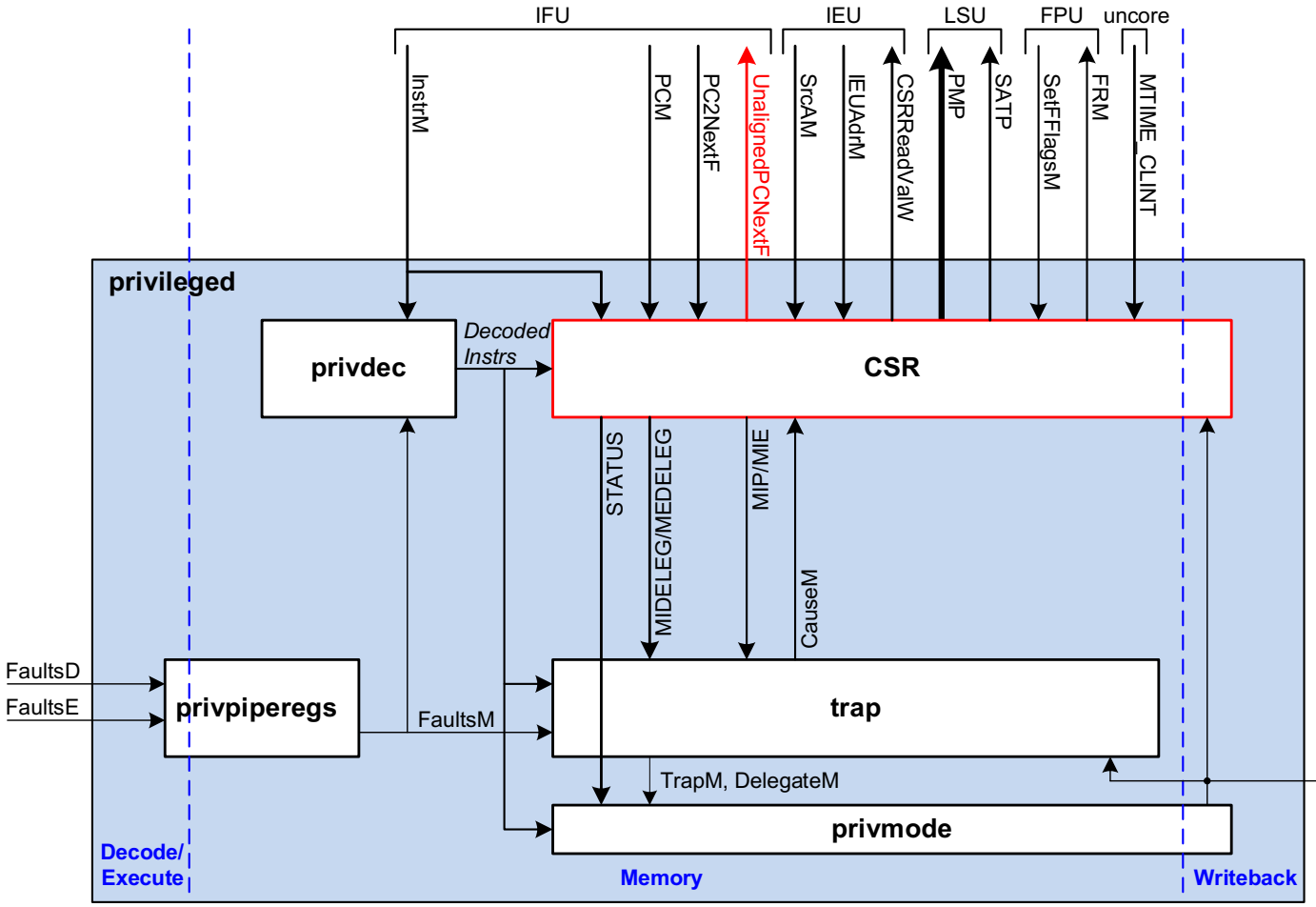
CSR Unit (csr)



csr:

- **During traps:**
  - **Sends to `trap`** pending/enable info (`mip`/`mie`) & deleg. info (`mideleg`/`medeleg`)
  - **Reads from `trap`:** *x*`cause` (`CauseM`)
  - **Sends** *x*`status`
  - **May record** `InstrM`, `PCM`, `IEUAdrM` (in *x*`tval`, `mepc`)

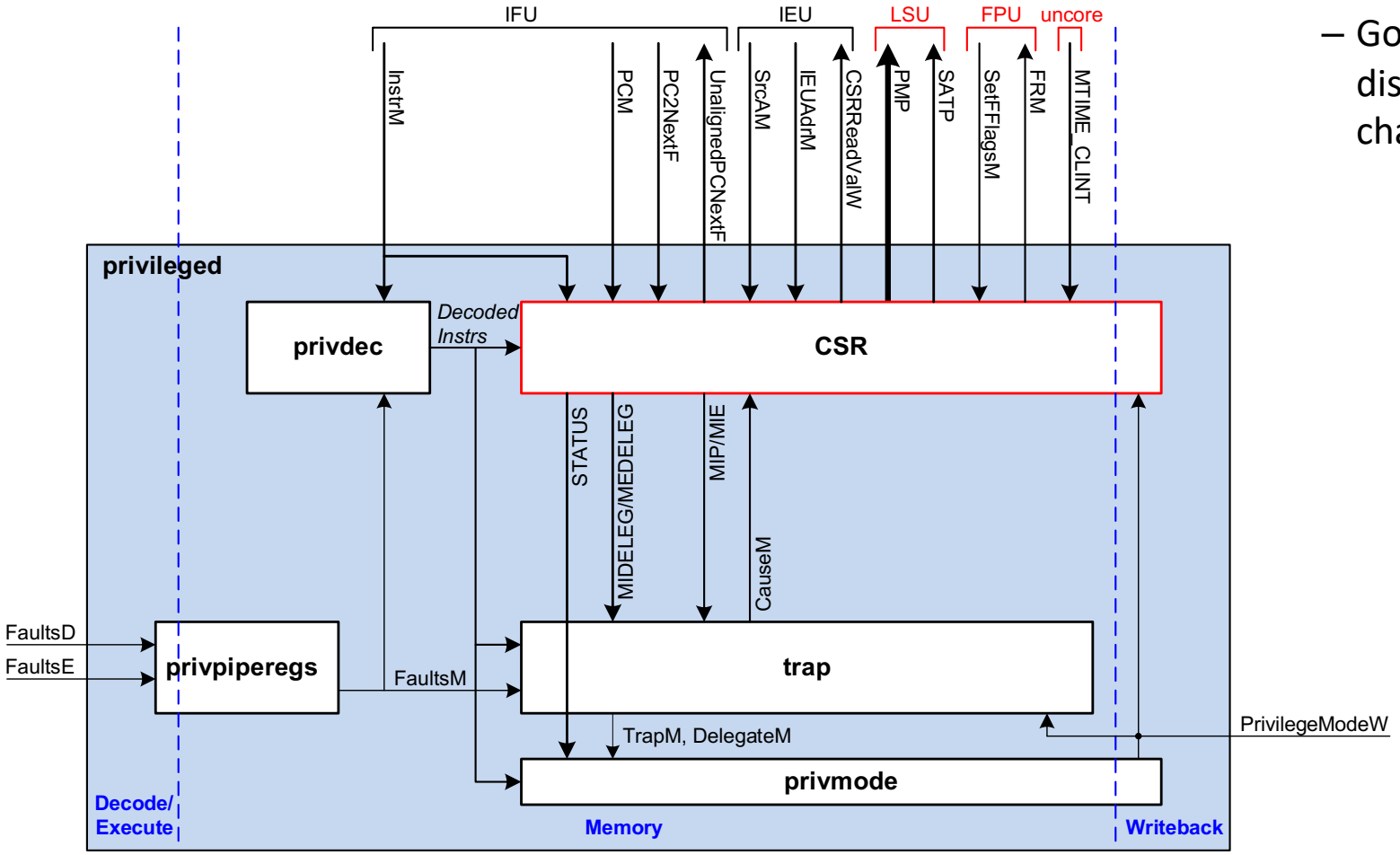
CSR Unit (csr)



csr:

- **During traps:**
  - **Sends to `trap`** pending/enable info (`mip`/`mie`) & deleg. info (`mideleg`/`medeleg`)
  - **Reads from `trap`:** *x*`cause` (`CauseM`)
  - **Sends** *x*`status`
  - **May record** `InstrM`, `PCM`, `IEUAdrM` (in *x*`tval`,`mepc`)
  - Determines **trap destination:** `UnalignedPCNextF` (using *x*`tvec`)

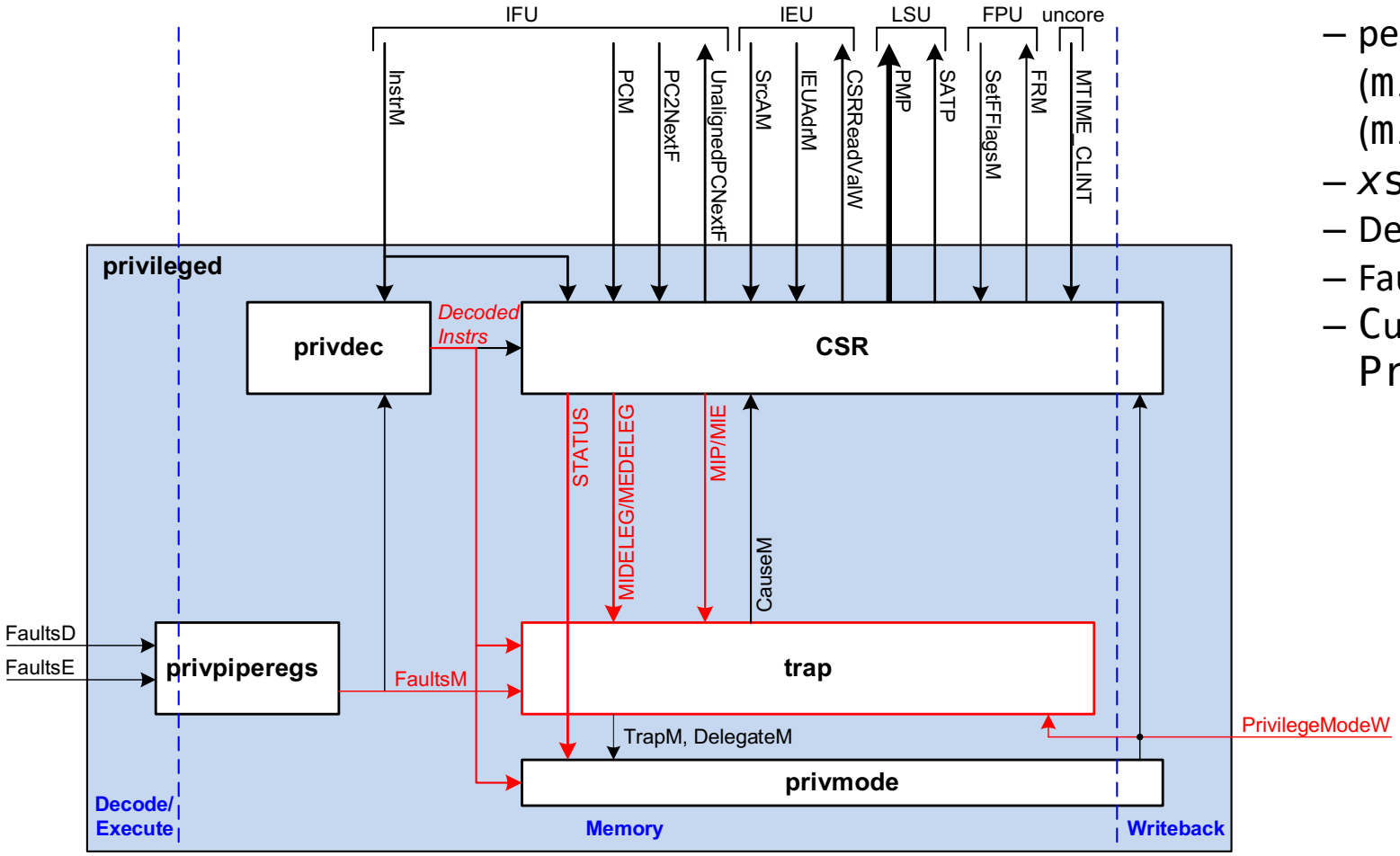
CSR Unit (csr)



csr:

- **Other signals:**
  - Go to LSU, FPU, Uncore: discussed in later chapters

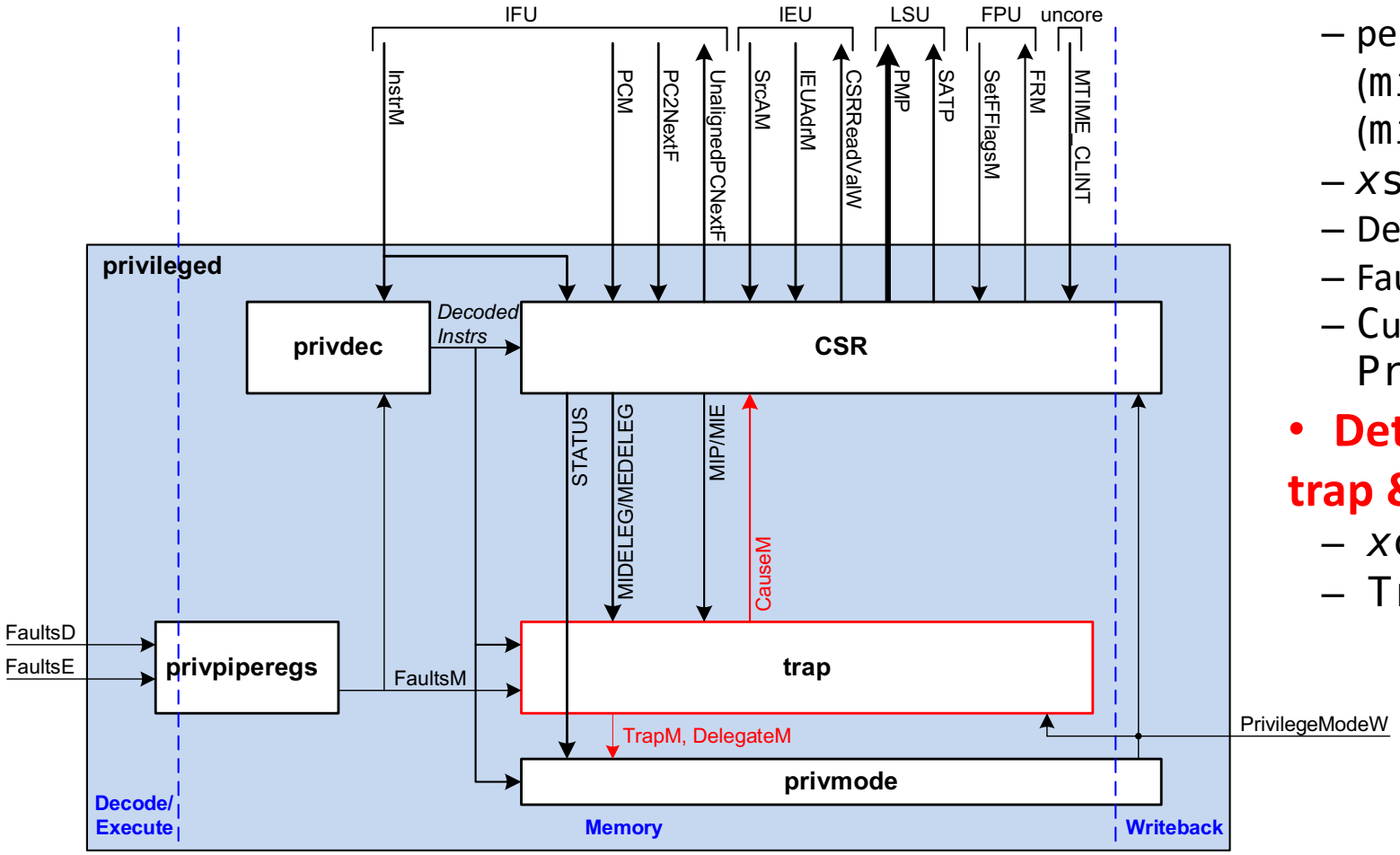
Trap Unit (trap)



trap:

- **Receives trap info:**
  - pending/enable info (mip/mie) & deleg. info (mideleg/medeleg)
  - *x*status
  - Decoded instruction
  - Fault info (FaultsM)
  - Current privilege mode, PrivilegeModeW)

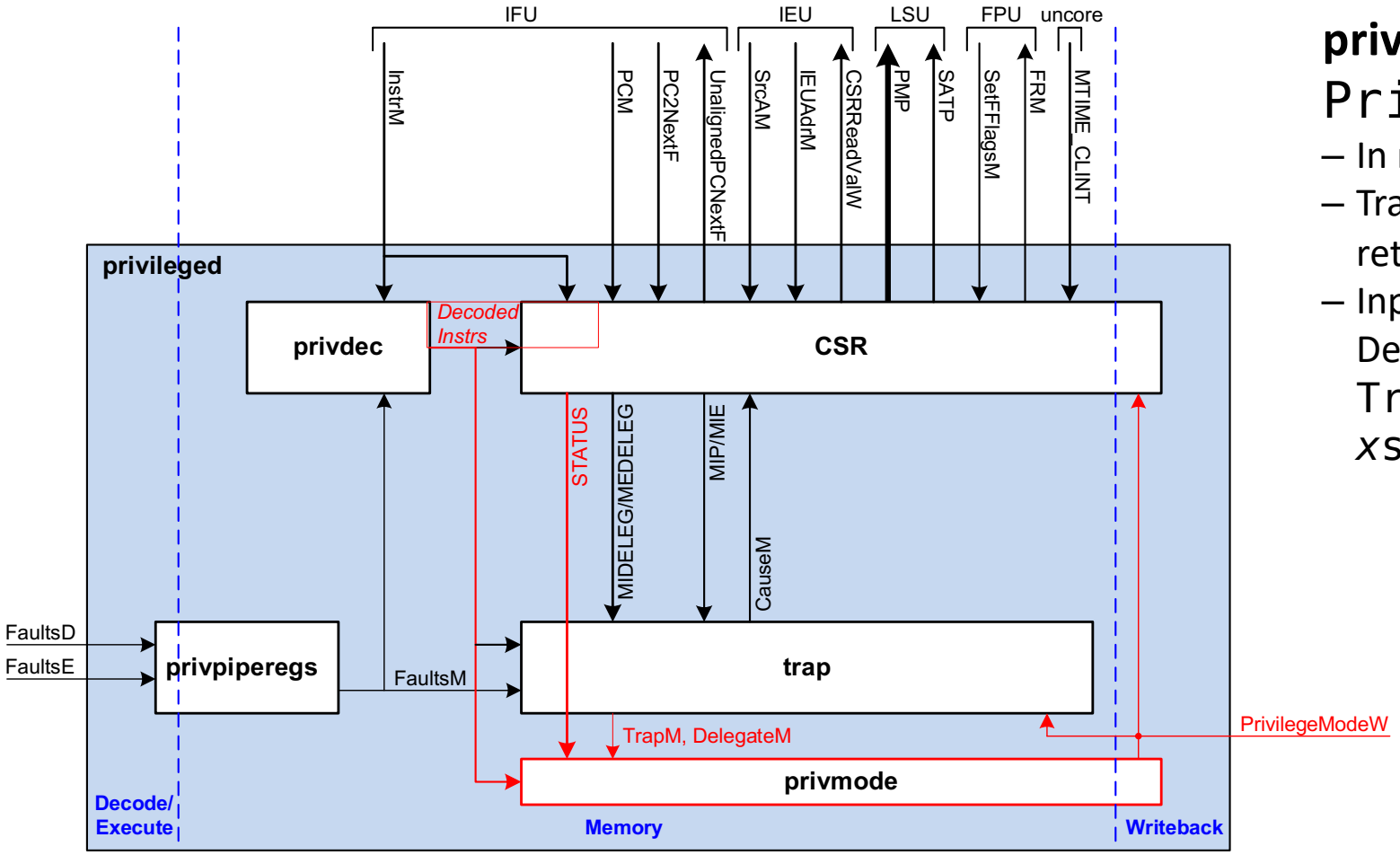
Trap Unit (trap)



trap:

- **Receives trap info:**
  - pending/enable info (mip/mie) & deleg. info (mideleg/medeleg)
  - *x*status
  - Decoded instruction
  - Fault info (FaultsM)
  - Current privilege mode, PrivilegeModeW)
- **Determines if takes trap & sends response:**
  - *x*cause (CauseM)
  - TrapM, DelegateM

Privilege Mode Unit (privmode)



privmode:

- **Stores & changes privilege mode,** `PrivilegeModeW`:
  - In response to:
  - Traps & exception returns (`mret`/`sret`)
  - Inputs/registers: Decoded instructions, `TrapM`, `DelegateM`, *x*`status` (STATUS)

Chapter 8: Privileged Operations

Privileged Configuration

Privileged Configuration

Support privileged operations:

- **`ZICSR_SUPPORTED` = `1`** (in `wally-config.vh`)

Don't support privileged operations:

- **`ZICSR_SUPPORTED` = `0`** (in `wally-config.vh`)
- Then all privileged signals and architectural state tied to 0
- So, all privileged hardware and signals optimized away by logic synthesis

Other Privileged Features

Configurations to support other privileged features

Flag (in `wally-config.vh`)	Feature Supported
S_SUPPORTED	S-Mode
ZICOUNTERS_SUPPORTED	Performance Counters
VECTORED_INTERRUPTS_SUPPORTED	Vectored Interrupts
BIGENDIAN_SUPPOPRTED	Big-Endian loads and stores
WFI_TIMEOUT_BIT	WFI times out after $2^{\text{WFI\_TIMEOUT\_BIT}}$ cycles
VIRTMEM_SUPPORTED	Virtual Memory (see Chapter 8)
PMP_ENTRIES	Physical Memory Protection (0, 16, or 64) (see Ch.8)

Privileged instructions and CSR accesses that are unsupported or not permitted in current privilege mode cause an **Illegal Instruction fault**.

Chapter 8: Privileged Operations

Privileged Instructions

Privileged Instruction Fields/Signals

Privileged instructions:

- **Instruction Fields:**
  - **`op = 1110011`**
  - **`funct3`**
  - Upper 12 bits of instruction
- **Signals:**
  - IEU asserts `PrivilegedM` (based on `op`)
  - The Privileged Decoder (`privdec`) looks at other fields to determine specific privileged instruction

Privileged Instructions

Privileged Instruction	Effect
x`ret` (`mret` / `sret`)	Return from trap: PC = *x*`epc` privileged mode = `status.MPP/SPP` `status.MIE/SIE` = `status.MPIE/SPIE`
`ecall` / `ebreak`	OS (system) call / debugger breakpoint
`wfi`	Wait for interrupt: stalls Wally in Memory stage until interrupt or timeout occurs. (The timeout counter is in the Privileged unit.)
CSR reads / writes	Reads, writes, or sets/clears bits in CSRs
`sfence.vma`	Flushes translation lookaside buffer (TLB) (see Chapter 8)

Chapter 8: Privileged Operations

CSRs

CSR Module (`csr`)

Contains:

- CSRs
- Logic to read, write, set, clear bits in CSRs
- Logic to compute mtval and PC during traps

`csr` submodules:

Submodule	Name	Registers
`csrsr`	Status registers	*x*`status` (`mstatus` / `sstatus`)
`csri`	Interrupt registers	Interrupt pending / enable registers *x*`ip` (`mip`, `sip`) / *x*`ie` (`mie`, `sie`)
`csrc`	Counters	Performance counters
`csrm`	M-mode CSRs	Other M-mode CSRs
`csrs`	S-mode CSRs	Other S-mode CSRs
`csru`	U-mode CSRs	U-mode CSRs (including floating-point registers)

Performance Counters

Configuration settings:

ZICSR_SUPPORTED = 1

ZICOUNTERS_SUPPORTED = 1

- COUNTERS = # performance counters (up to 32)
- 64-bit counters

Performance Counters

Performance Counters

CSR	Count
B00	Cycles elapsed
B01	unused
B02	Instructions retired
B03	Load stalls
B04	Branch prediction wrong direction
B05	Total branches
B06	Branch/jump prediction wrong target address
B07	Total jumps
B08	Return address stack wrong address
B09	Total returns
B0A	Instruction class predictor wrong
B0B	Data cache accesses
B0C	Data cache misses
B0D	Instruction cache accesses
B0E	Instruction cache misses
B0B	Data cache accesses
B0C	Data cache misses
B0D	Instruction cache accesses
B0E	Instruction cache misses

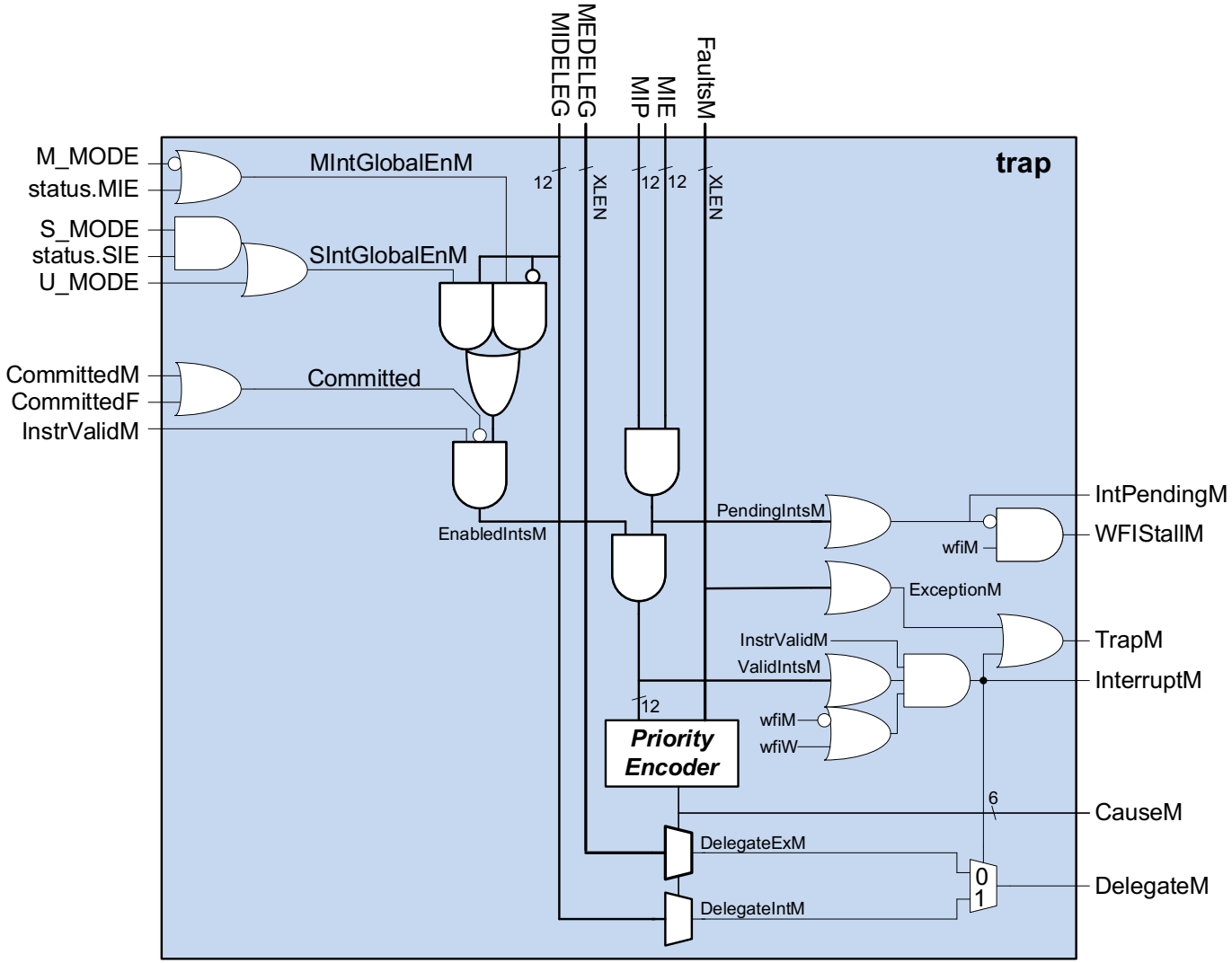
Chapter 8: Privileged Operations

Traps

Trap Unit (trap) Outputs

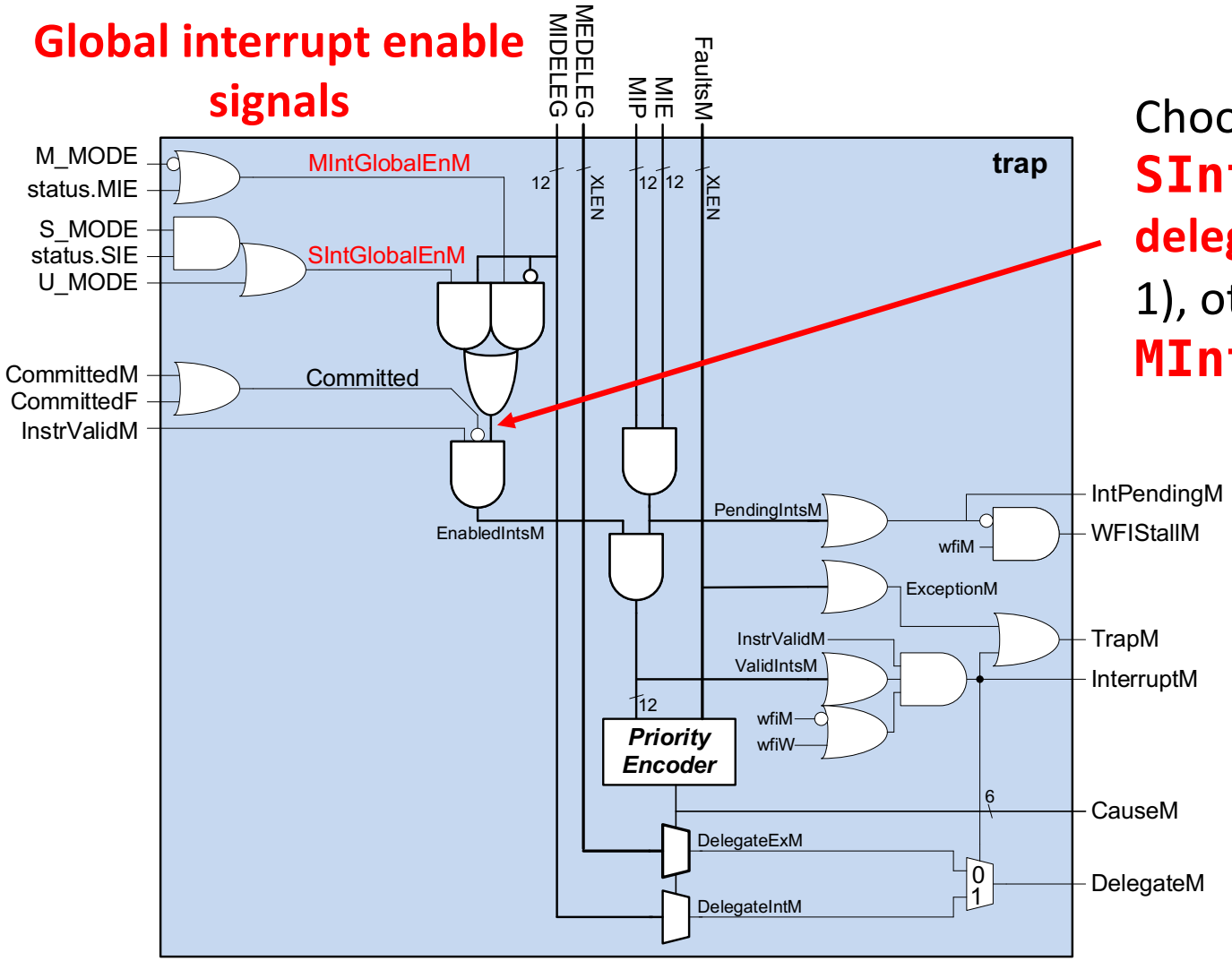
Signal	Description
`TrapM`	1 when trap should occur
`CauseM`	Cause of highest-priority trap
`DelegateM`	1 when trap should be delegated to S-mode

Trap Unit (trap)



Trap Unit (trap): Global Int. Enable

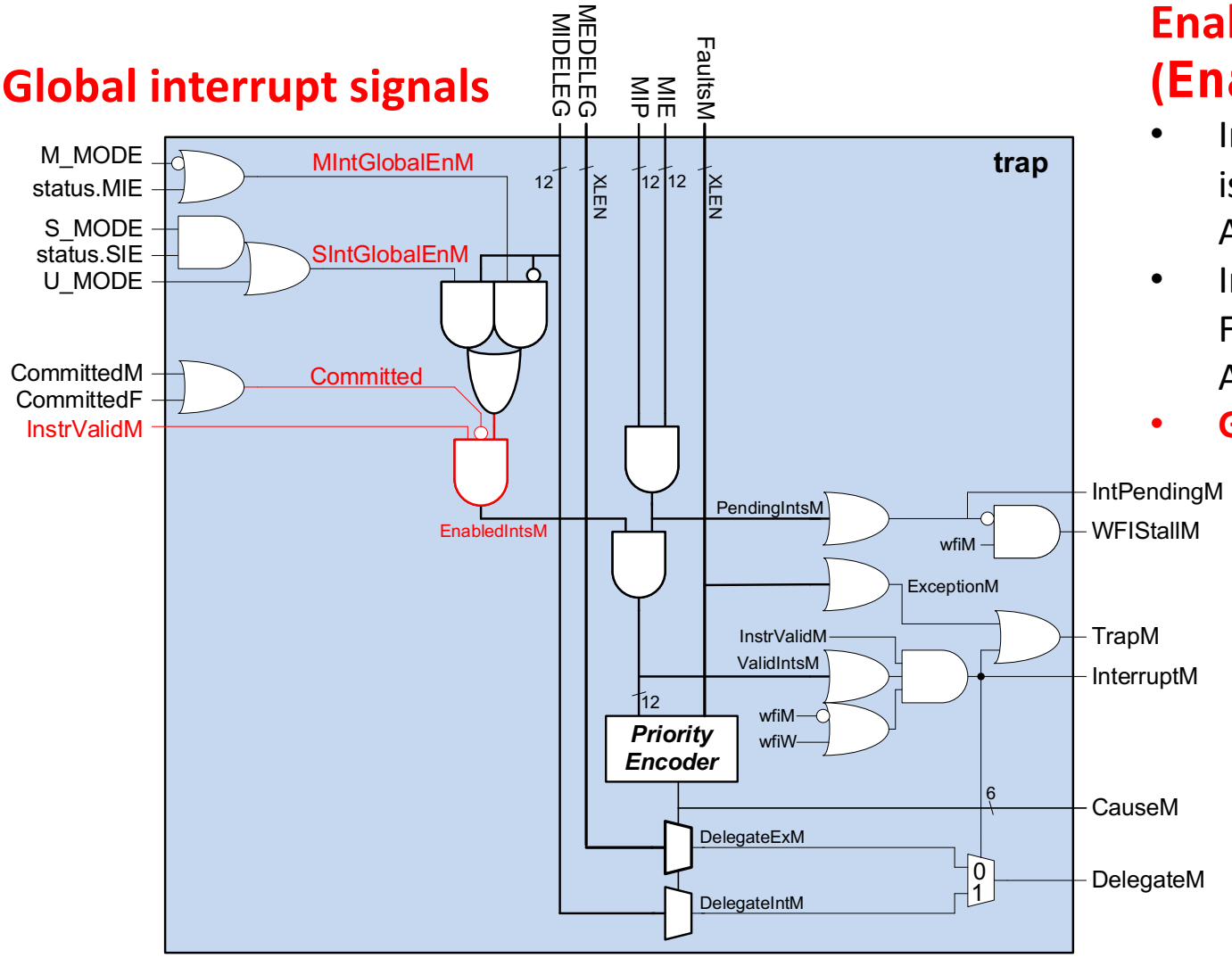
Global interrupt enable signals



Choose **SIntGlobalEnM if delegated** (MIDELEG bit is 1), otherwise choose **MIntGlobalEnM**

Trap Unit (`trap`): Enable Interrupts

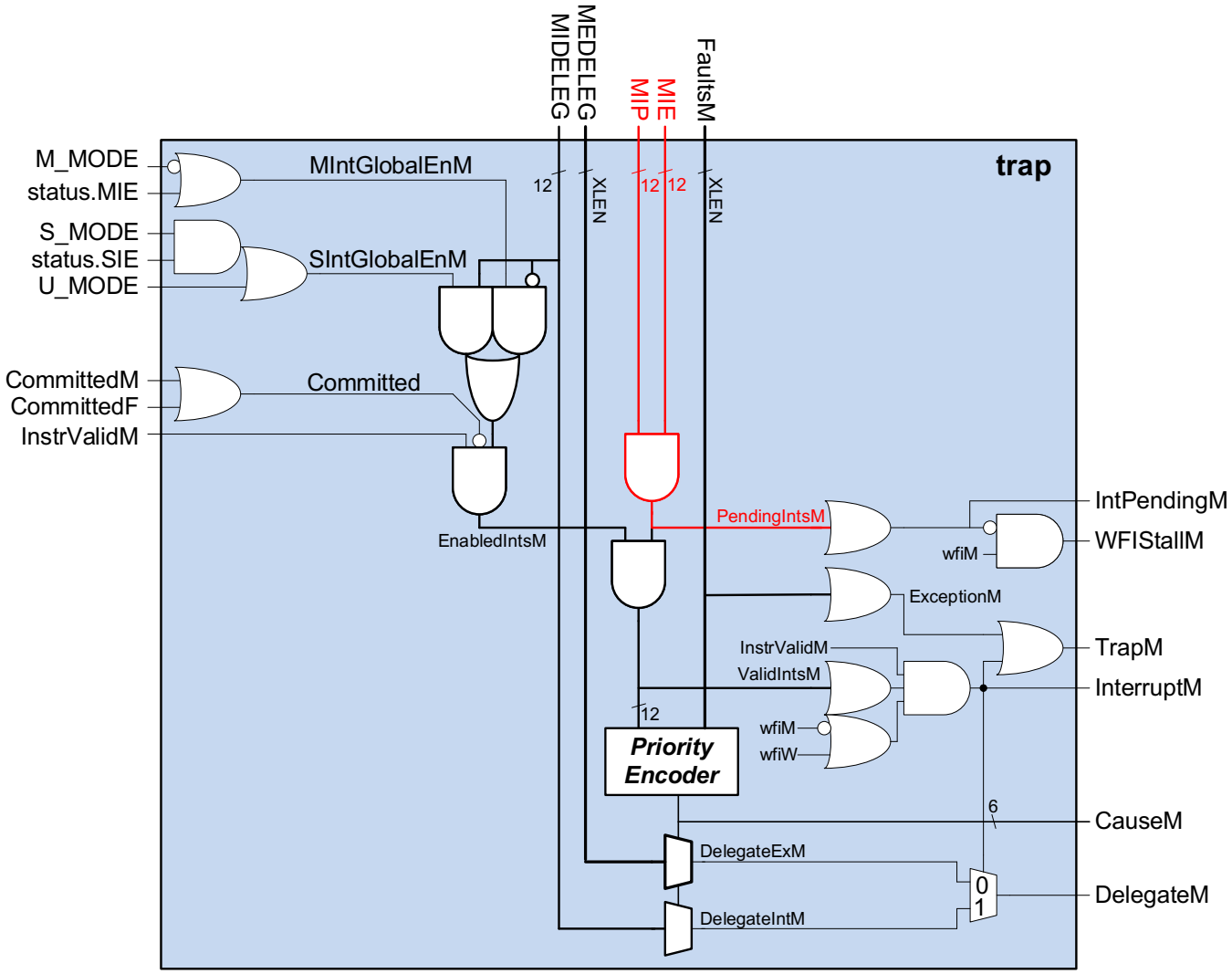
Global interrupt signals



Enable interrupts (`EnabledIntsM`) if:

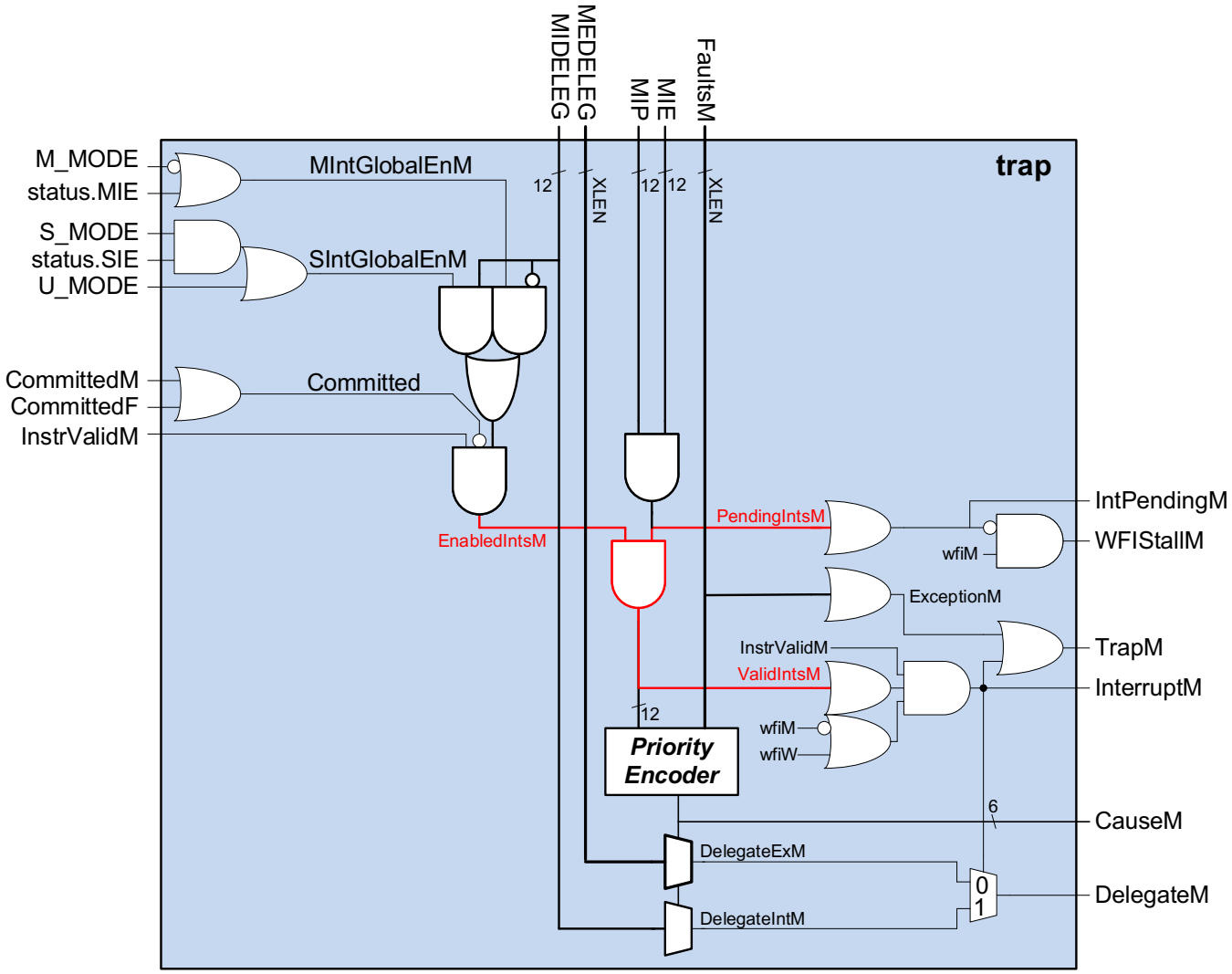
- Instruction in Memory stage is valid (`InstrValidM`) AND
- Instruction in Memory or Fetch stage **isn't committed** AND
- **Global interrupts** are enabled

Trap Unit (trap): Pending Interrupts



PendingIntsM:
Specific interrupt is **pending (MIP)** AND **enabled (MIE)**

Trap Unit (trap): Valid Interrupts

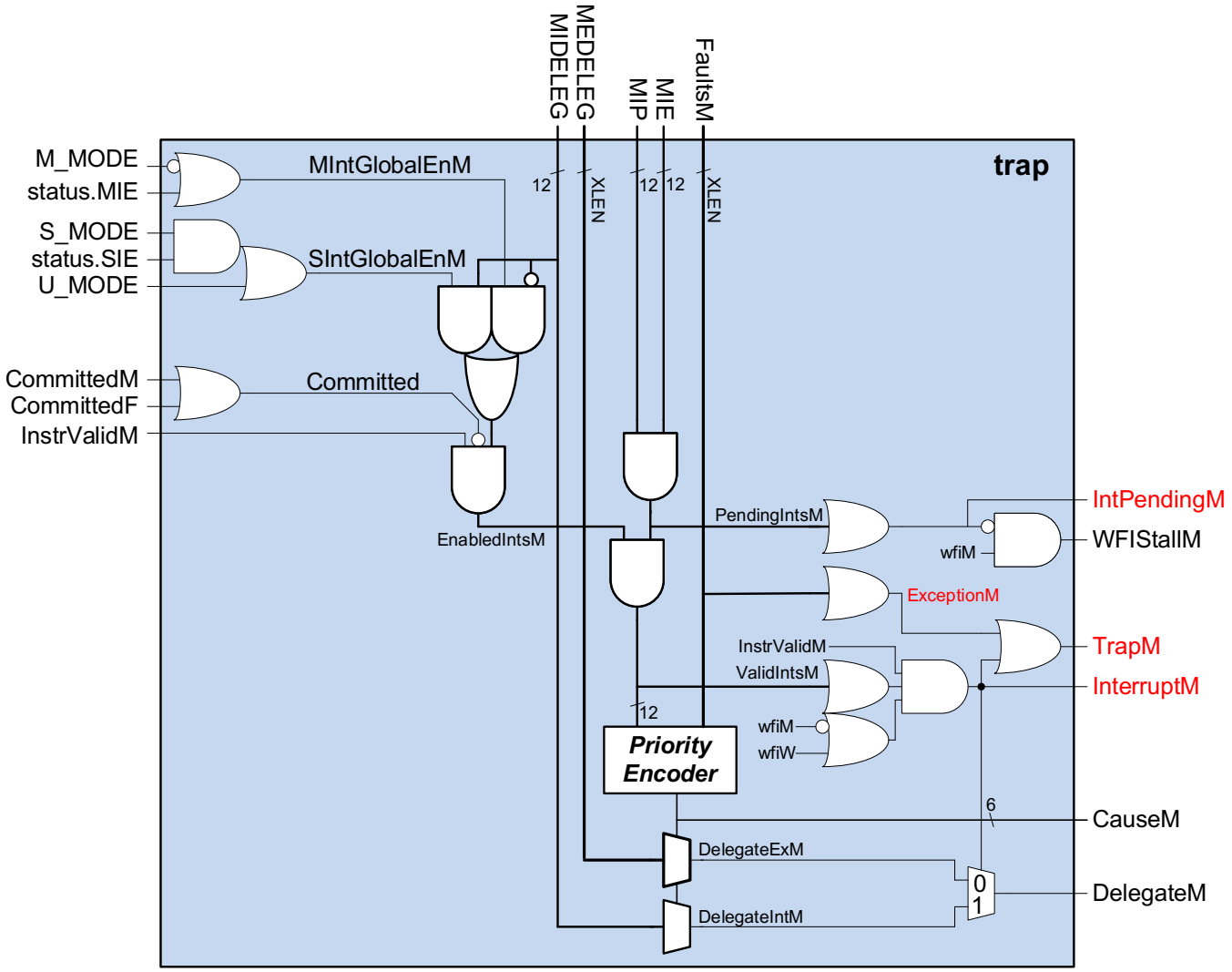


ValidIntsM:

Specific interrupts are:

- **Pending AND**
- **Enabled**

Trap Unit (trap): Global Signals



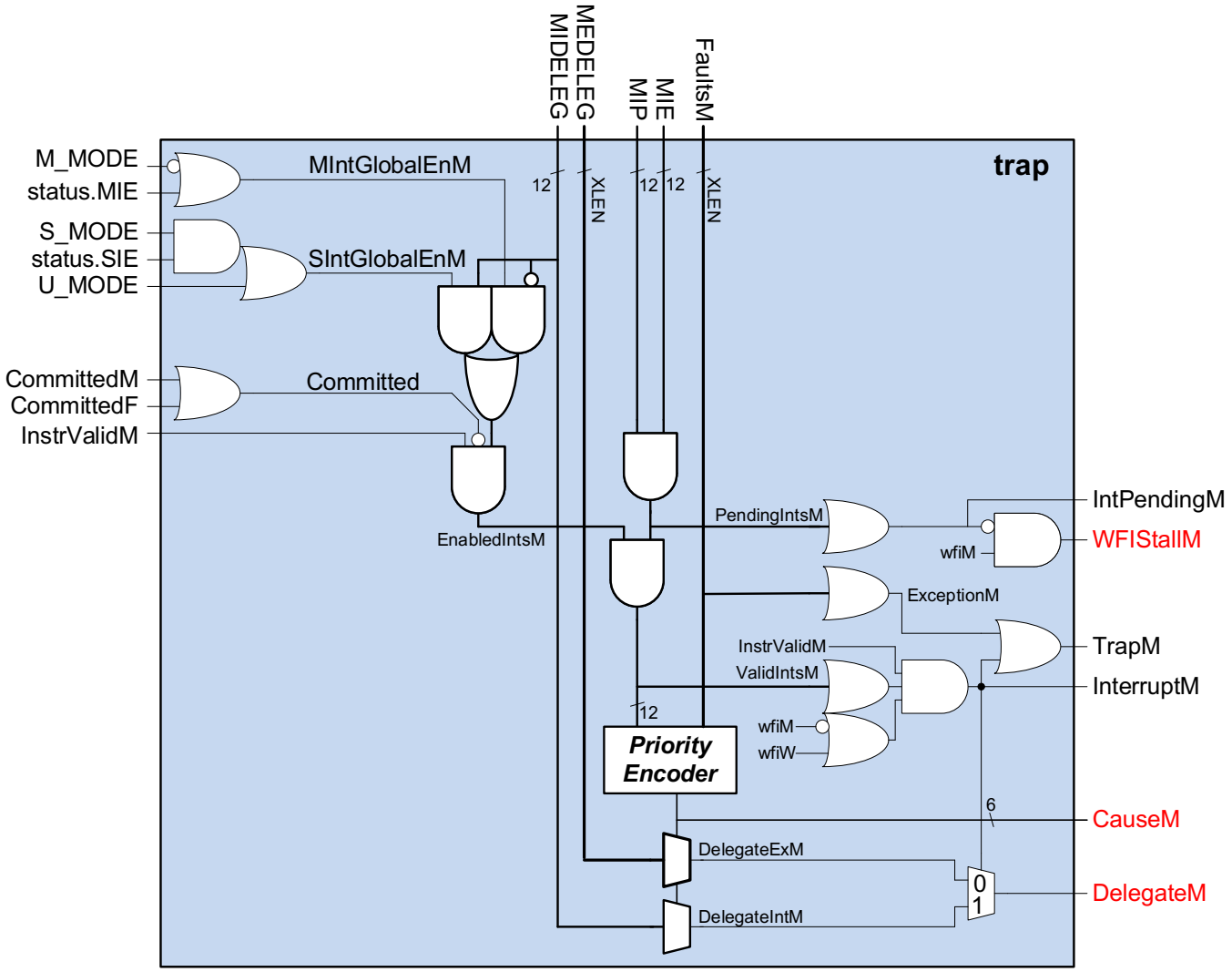
IntPendingM:
Any interrupt pending

ExceptionM:
Any exception

TrapM:
Any trap (interrupt or exception)

InterruptM:
Any interrupt

Trap Unit (trap): Global Signals

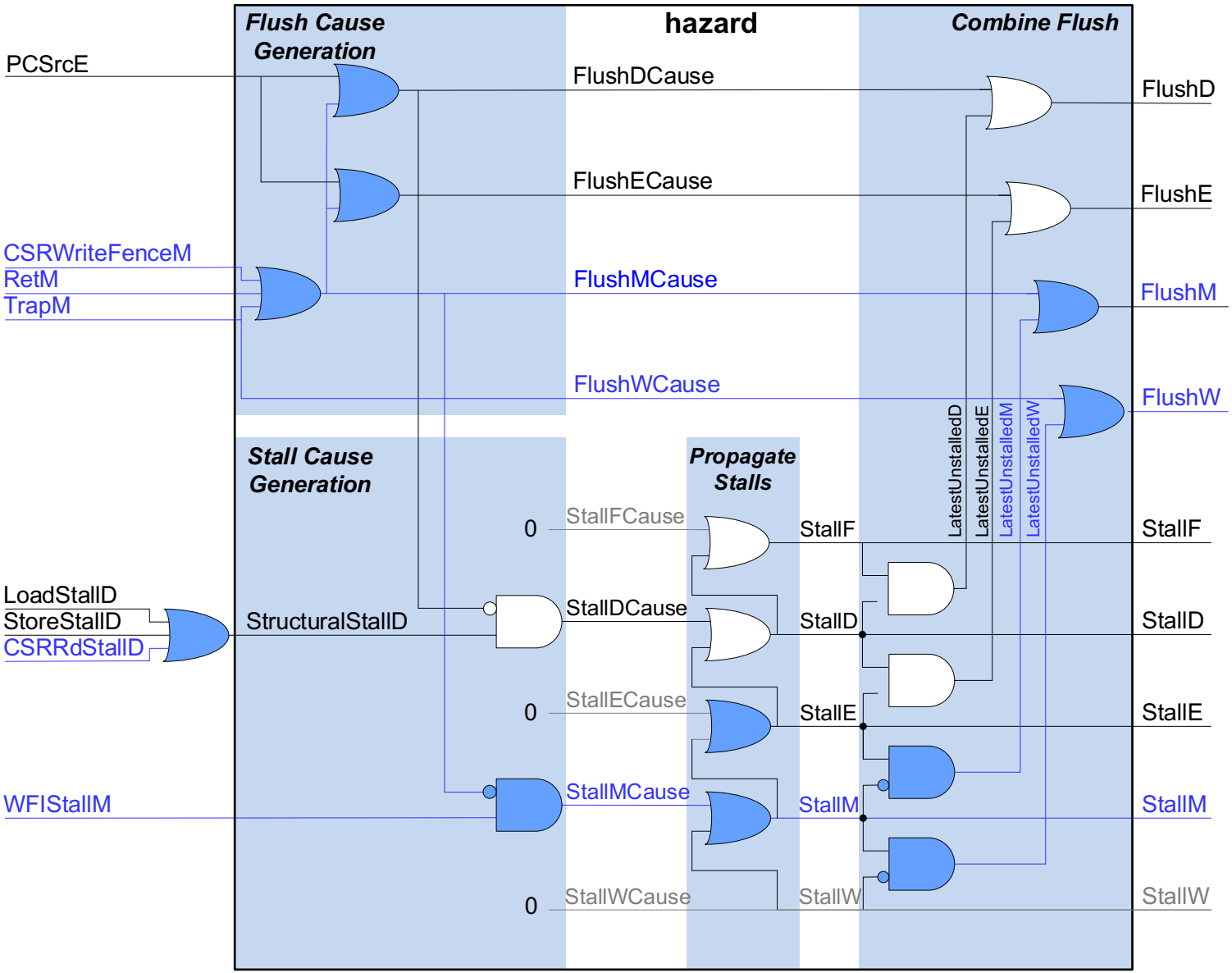


WFIStallM:
`wfi` instruction in Memory stage and no interrupts pending

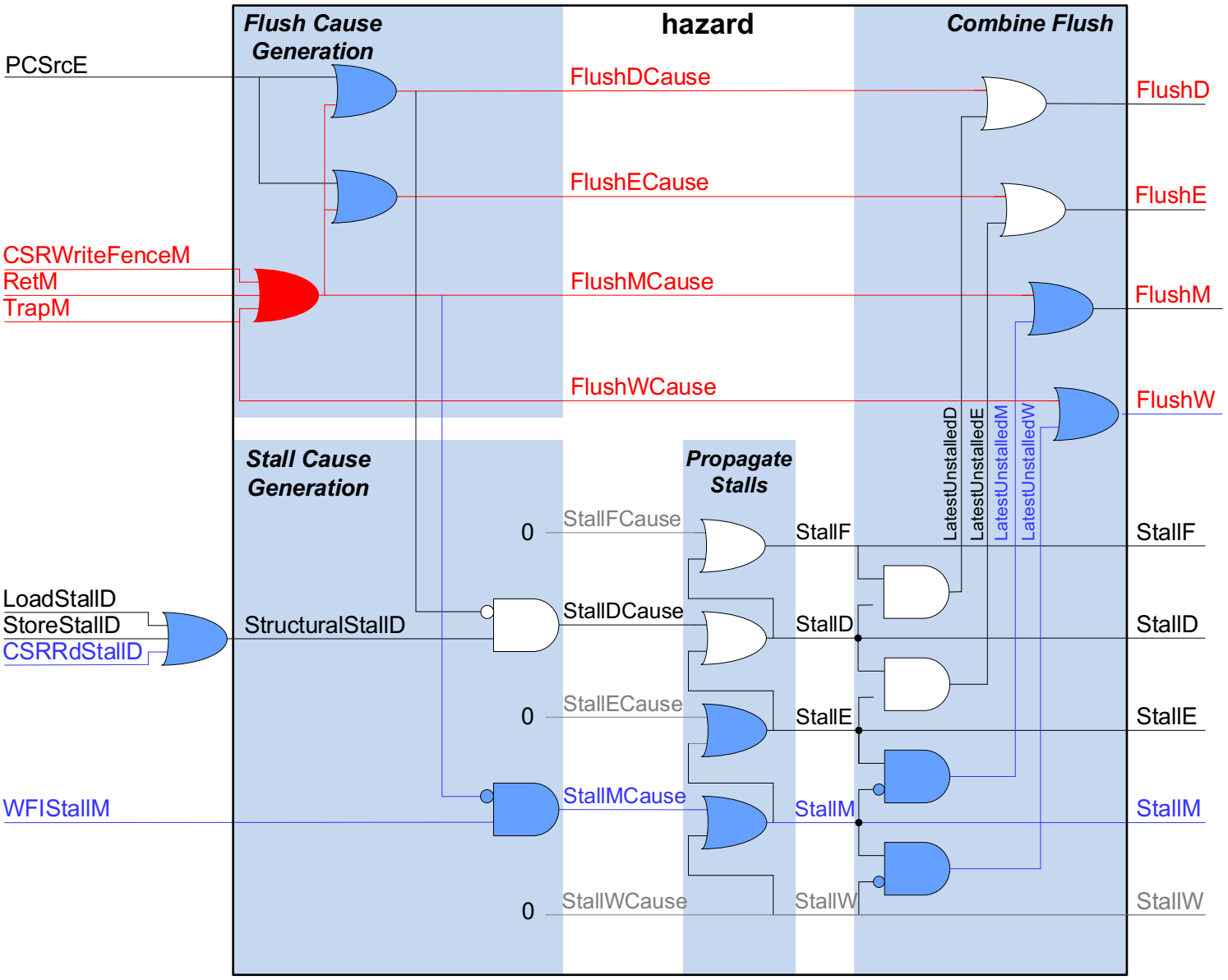
CauseM:
Cause of highest priority trap

DelegateM:
Delegate trap to S-mode: If trap is an interrupt, use interrupt delegation info, otherwise use exception delegation

Hazard Unit with Privileged Support



Hazard Unit: Flushing Pipeline



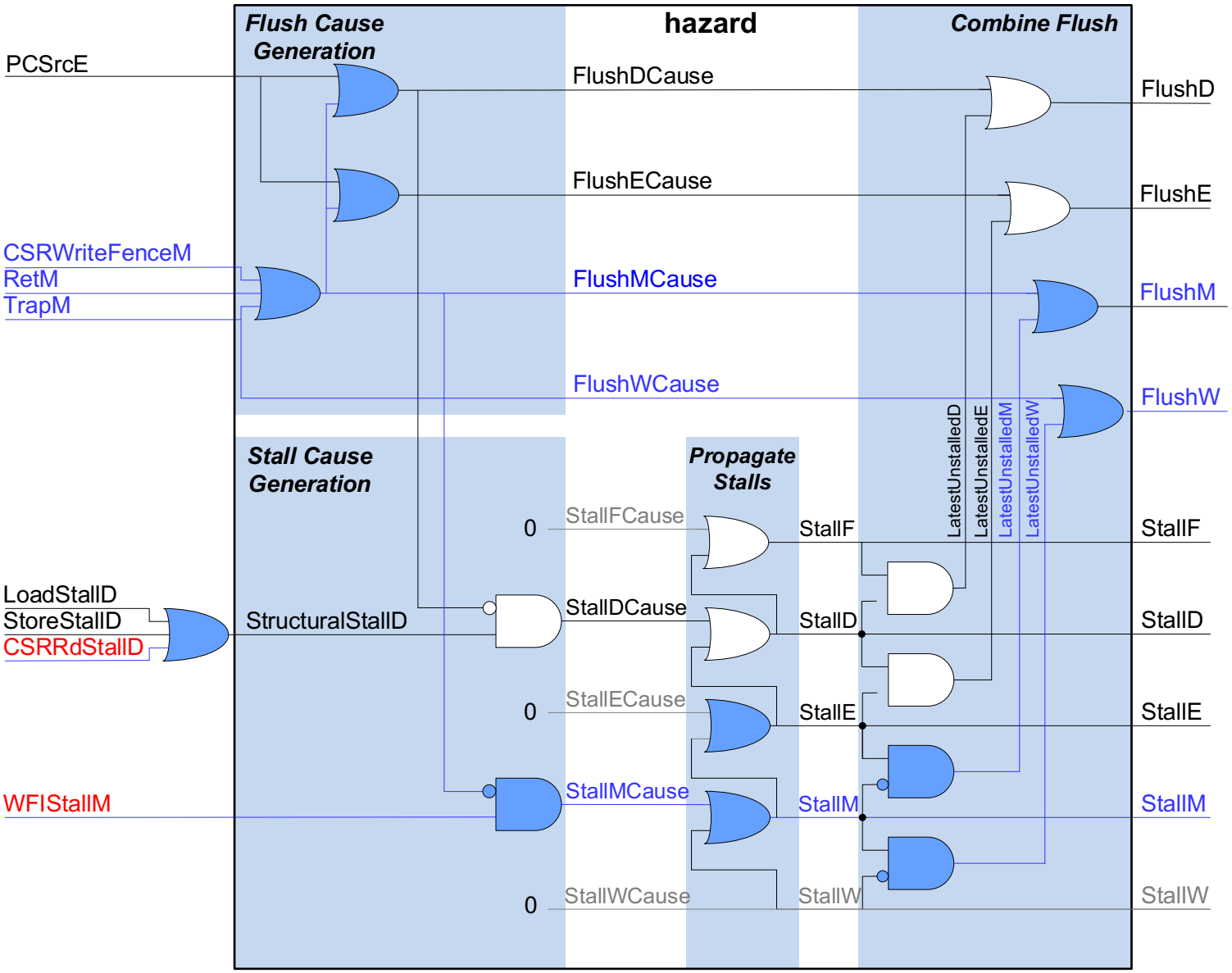
Flush Decode, Execute, and Memory stages upon:

- CSR write
- Exception return

Flush full pipeline upon:

- Trap

Hazard Unit with Privileged Support



If not being flushed…

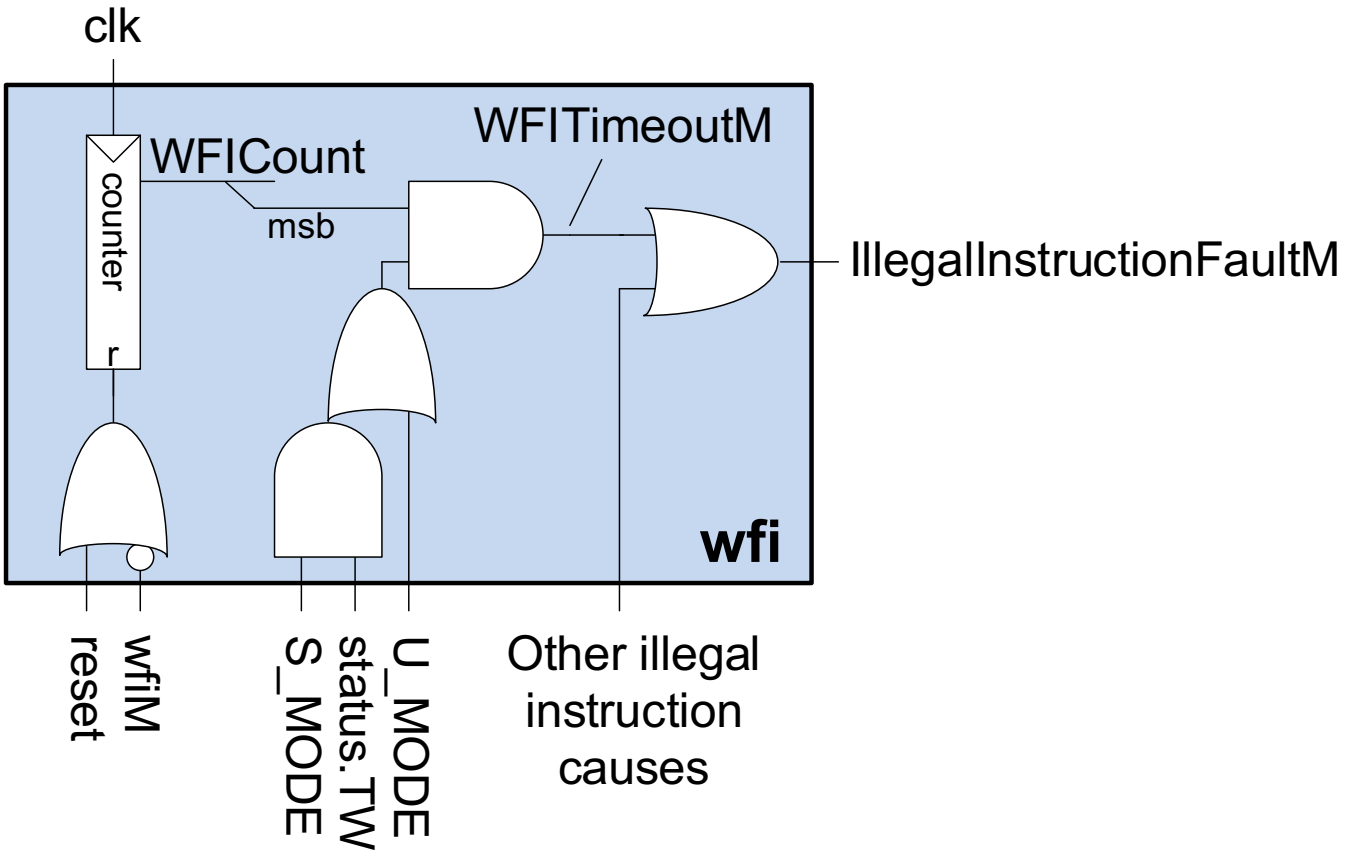
Stall Fetch and Decode stages upon:

- CSR read stall (2-cycle latency like loads/stores)

Stall F, D, E, and M stages upon:

- `wfi` (wait for interrupt instruction) stall in Memory stage

Timeout Logic for `wfi`



Counter resets upon:

- Reset, or
- No `wfi` instruction

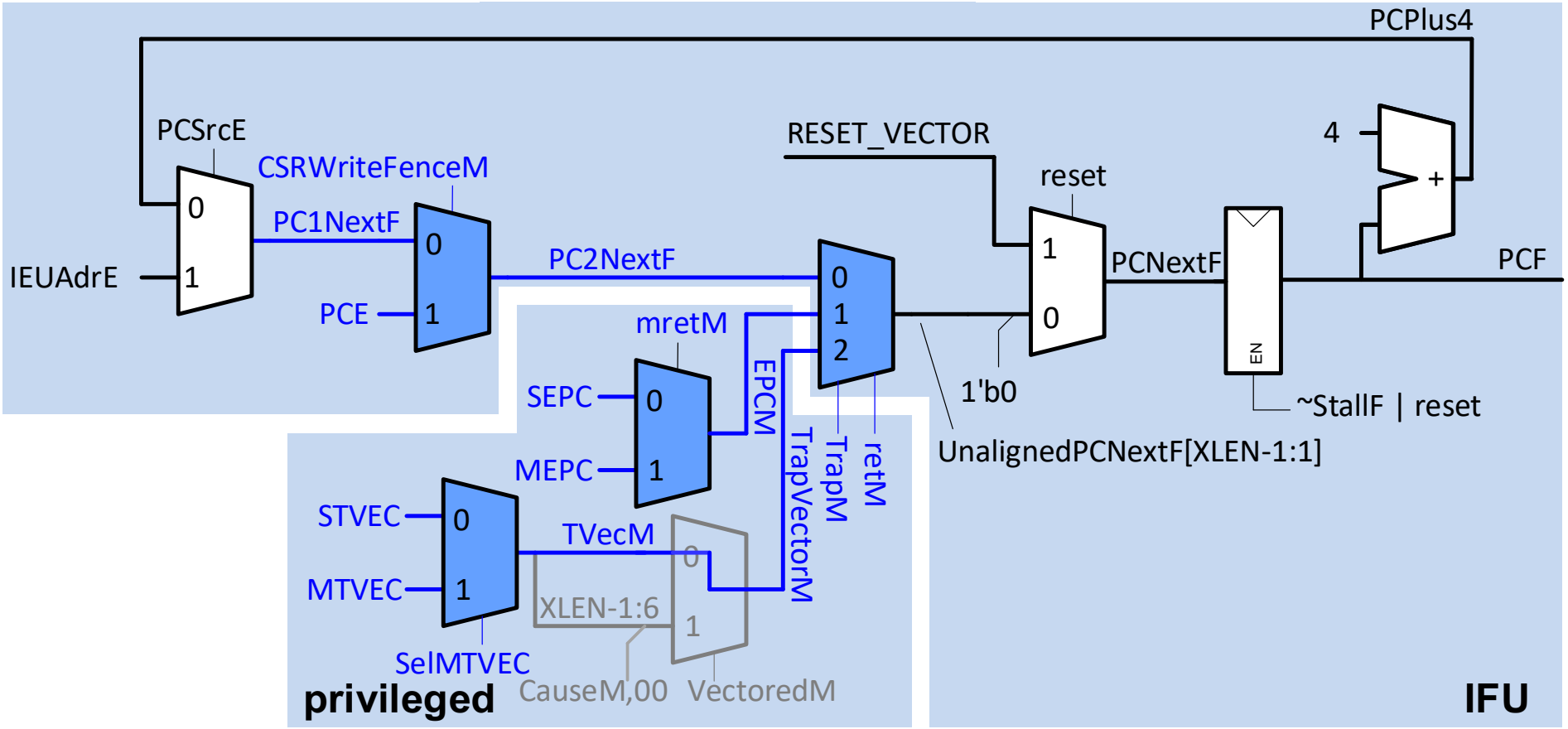
`wfi` times out (triggers an illegal instruction fault) if:

- Counter msb = 1 AND
- In U-mode OR
- In S-mode and TW (timeout wait) bit set in status register

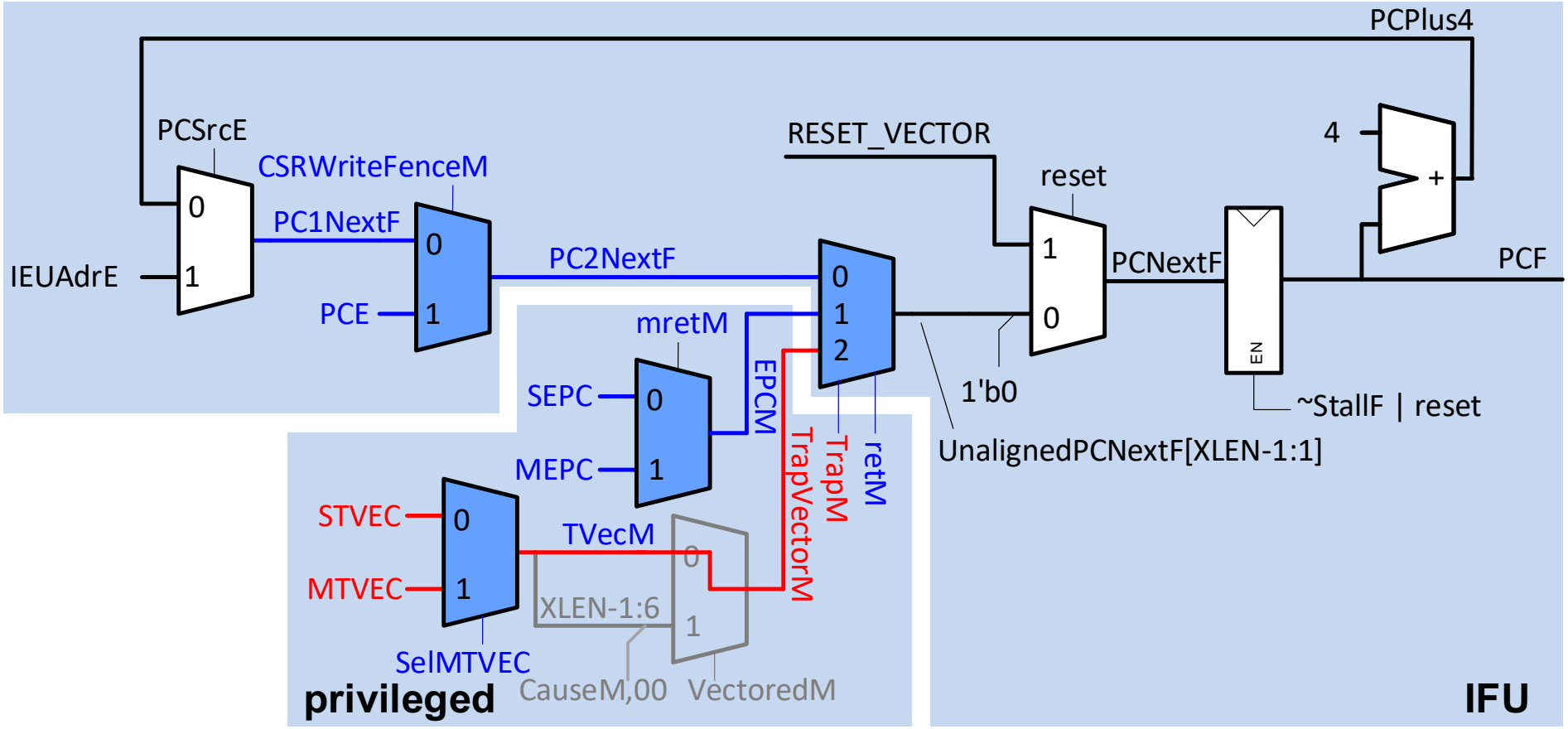
Chapter 8: Privileged Operations

IFU / Privileged Changes

PC Logic with Privileged Changes



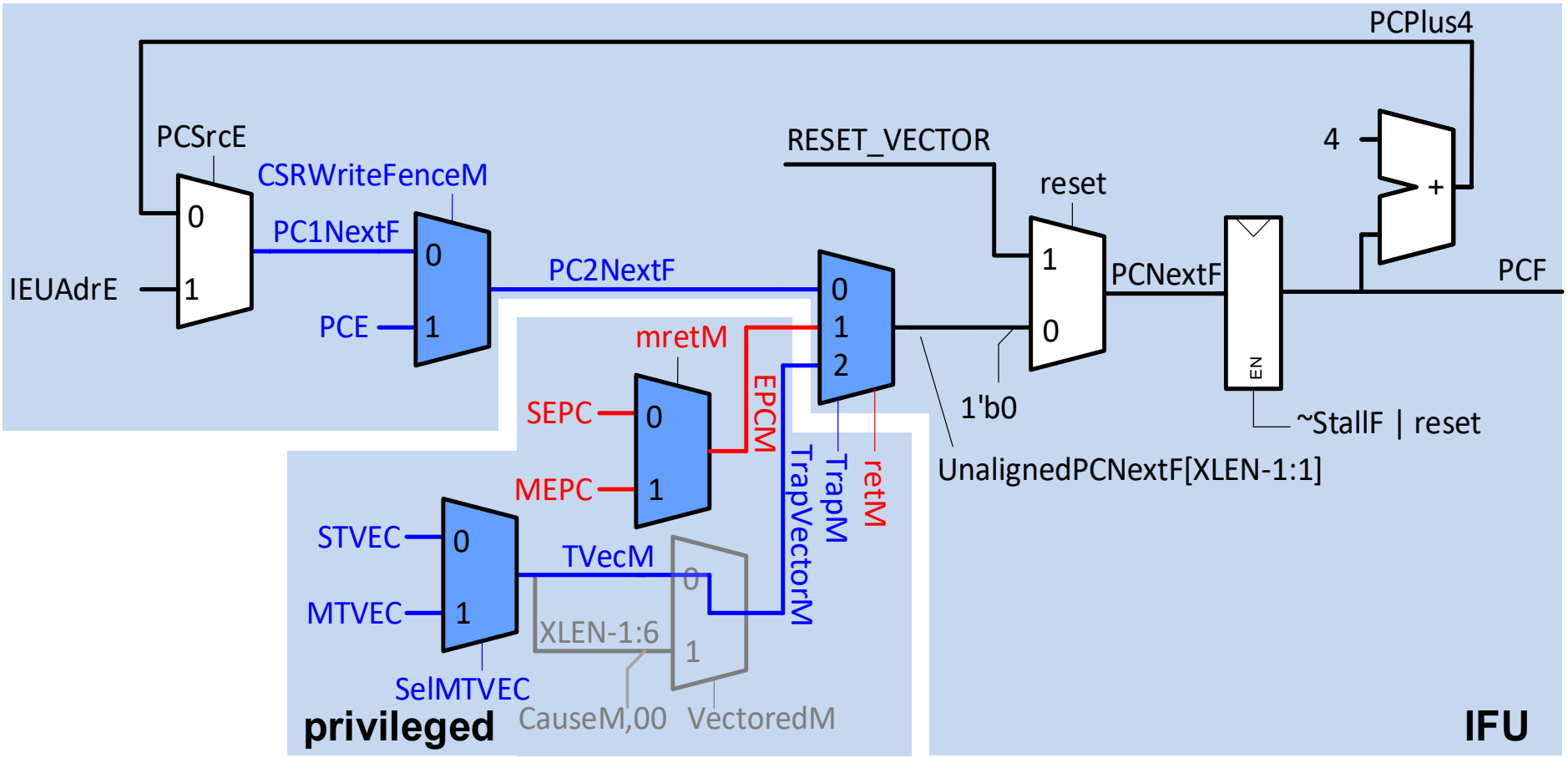
PC Logic with Privileged Changes



Jump to trap handler:

- **Nonvectored:** *xtvec* (`stvec` or `mtvec`)

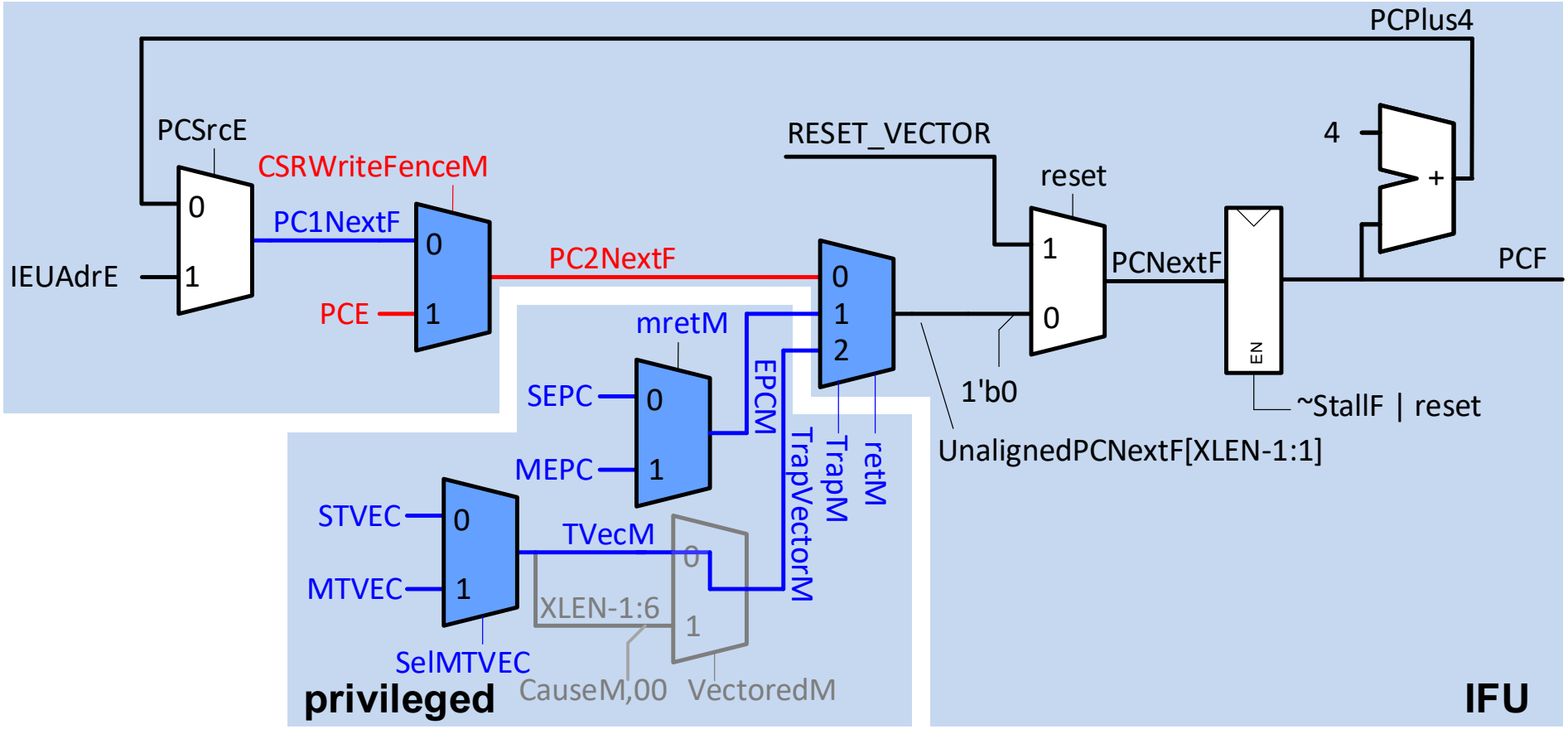
PC Logic with Privileged Changes



Trap return:

- *x*epc (sepc or mepc)

PC Logic: Upon Memory Stage Flush



If fences or CSRs supported:

- These flush the pipeline in the Memory stage
- So, the next instruction is in the Execute stage (PCE)

Chapter 5: Privileged Operations

Endian Swapper

Endian Example

What are the contents of memory at end of program for little-endian mode? Big-endian mode?

```
// RISC-V program
li  x1, 0xFEDCBA98_76543210
sd  x1, 1000(x0)
ld  x2, 1000(x0)
lwu x3, 1000(x0)
lwu x4, 1004(x0)
lbu x5, 1001(x0)
```

Endian Example

What are the contents of memory at end of program for little-endian mode? Big-endian mode?

```
// RISC-V program
li  x1, 0xFEDCBA98_76543210
sd  x1, 1000(x0)
ld  x2, 1000(x0)
lwu x3, 1000(x0)
lwu x4, 1004(x0)
lbu x5, 1001(x0)
```

Little-Endian Memory Contents	Byte Address	Big-Endian Memory Contents
FE	1007	10
DC	1006	32
BA	1005	54
98	1004	76
76	1003	98
54	1002	BA
32	1001	DC
10	1000	FE

Endian Example

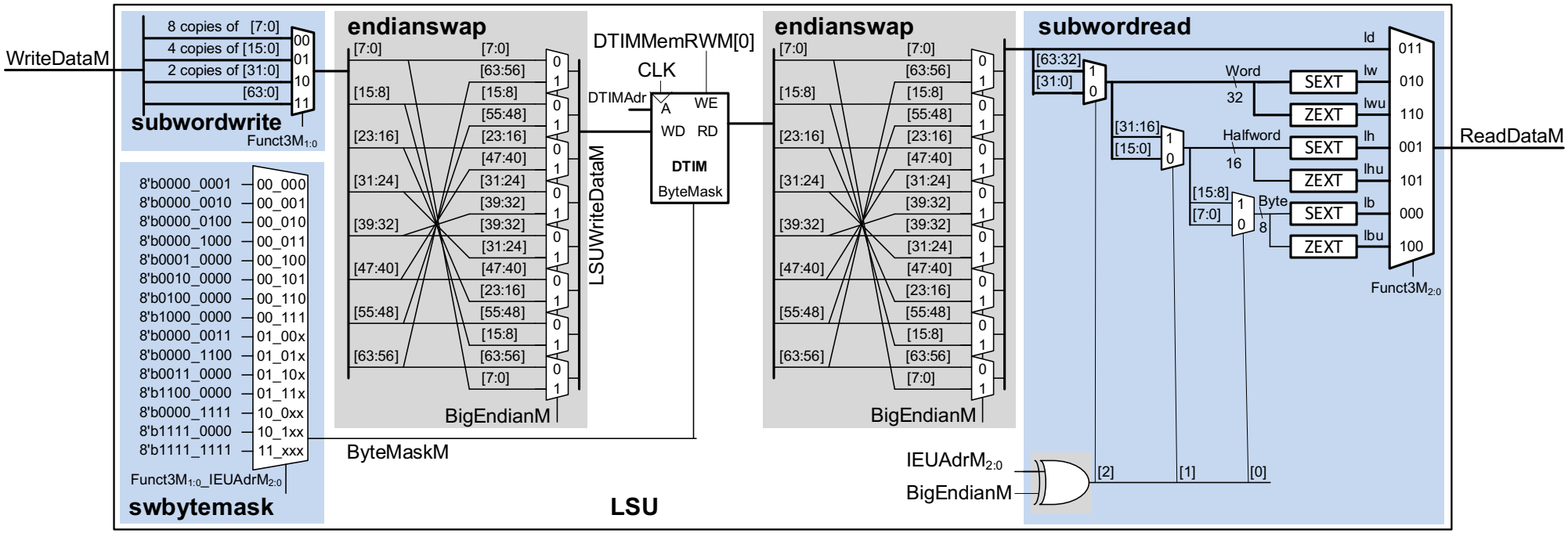
What are the contents of memory at end of program for little-endian mode? Big-endian mode?

```
// RISC-V program
li  x1, 0xFEDCBA98_76543210
sd  x1, 1000(x0)
ld  x2, 1000(x0)
lwu x3, 1000(x0)
lwu x4, 1004(x0)
lbu x5, 1001(x0)
```

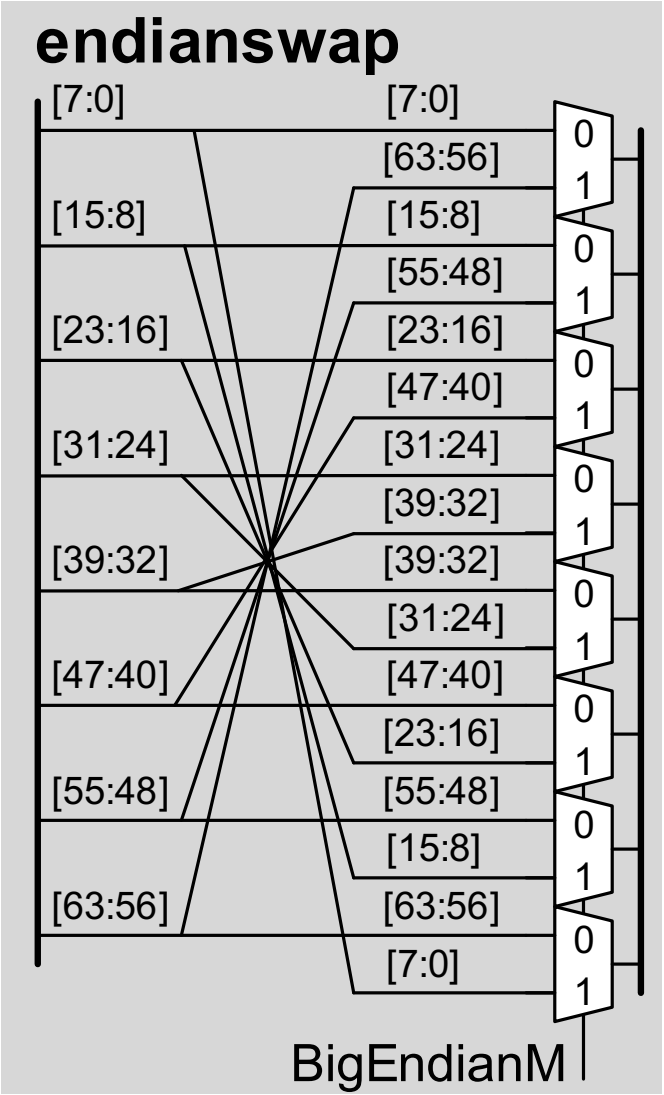
Little-Endian Memory Contents	Byte Address	Big-Endian Memory Contents
FE	1007	10
DC	1006	32
BA	1005	54
98	1004	76
76	1003	98
54	1002	BA
32	1001	DC
10	1000	FE

Load	Register	Little Endian	Big Endian
`ld x2, 1000(x0)`	x2	FEDCBA98_76543210	FEDCBA98_76543210
`lwu x3, 1000(x0)`	x3	00000000_76543210	00000000_FEDCBA98
`lwu x4, 1004(x0)`	x4	00000000_FEDCBA98	00000000_76543210
`lbu x5, 1001(x0)`	x5	00000000_00000032	00000000_000000DC

LSU with Endian Swap Unit



Endian Swap Unit (`endianswap`)



About these Notes

RISC-V System-on-Chip Design Lecture Notes

Trap Example

Trap Example: Context Switching

- **Practical Example: Multiple User Processes**
  - S-mode OS time-multiplexes one physical hart to give the illusion of many independent user-mode harts, each running an individual process.
  - These user-mode harts can't corrupt each other or interact except through OS calls.
  - Hence, a defective program with an infinite loop or read to a bad memory location won't crash other programs or steal their secret information
  - The operating system periodically suspends the current user-mode hart, loads in the state of the next hart, and starts running its process.
  - A supervisor-mode timer interrupt every 0.25-10 ms or so controls this time multiplexing
    - Often enough that the user perceives all processes running concurrently
    - Infrequent enough that context switch overhead is minor
  - Core-Local Interruptor (CLINT) peripheral is programmed to generate a supervisor-mode interrupt on the STimerInt pin at the desired rate
  - *** NEEDS SSTC EXTENSION

Supervisor Timer Interrupt

- `MIDELEG.STI` (bit 5) = 1 to delegate supervisor timer interrupts to the S-mode OS
- When `STimerInt` = 1