XV-6 : Traps

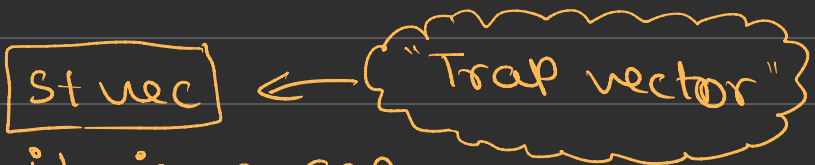
Traps Processing (Hard ware)

Traps

- Exception
  - ↳ sys call
  - ↳ Program Error.
- Interrupt.

Example:
- System call
  "ecall" instruction
- Program Error
  Illegal instruction,
  Alignment Errors
- Device Interrupt.
  interrupt can occur when we are executing in user mode or supervisor mode

After trap happens, regardless of any mode,
we begin executing some handler code running in supervisor mode.

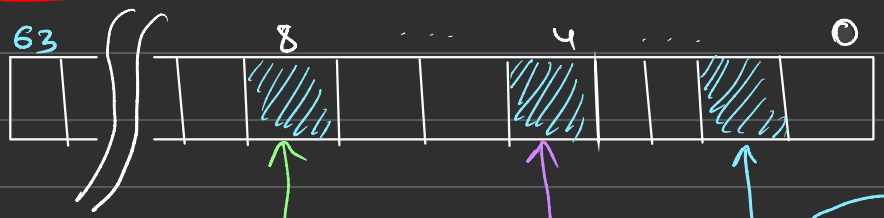


it is a CSR
contains the address of the "Handler" code.

There are two pieces of code in xv6 that handle traps.

- **kernelvec**
  Handles traps during code in kernel mode.
- **uservec**
  Handles traps that happens when running in user mode.

Sstatus



SIE

Interrupts enabled.

0 = disabled

1 = enabled

SPIE

Previous interrupt Enabled

SPP

Previous Privilege level (Previous mode)

0 → user mode

1 → supervisor mode

What happens when a Trap occurs?

Are interrupts disabled?

↳ The interrupts remains 'Pending' until interrupts are re-enabled.

↳ If it is an exception it's going to be handled immediately regardless of whether interrupts are disabled or not.

When traps occurs, The hardware does this.

sepc ← PC

PC ← stvec

scause ← ...

1: "software" Ctrl-C
8: system call
9: External Device
other?

stval ← ... additional info. ...

//hardware will save previous mode

SStatus.SPP ← Previousmode

0 = user
1 = supervisor

sstatus.spie ← sstatus.sie

save previous "interrupts enabled bit."

sstatus.sie ← ∅ ← Disable Interrupt

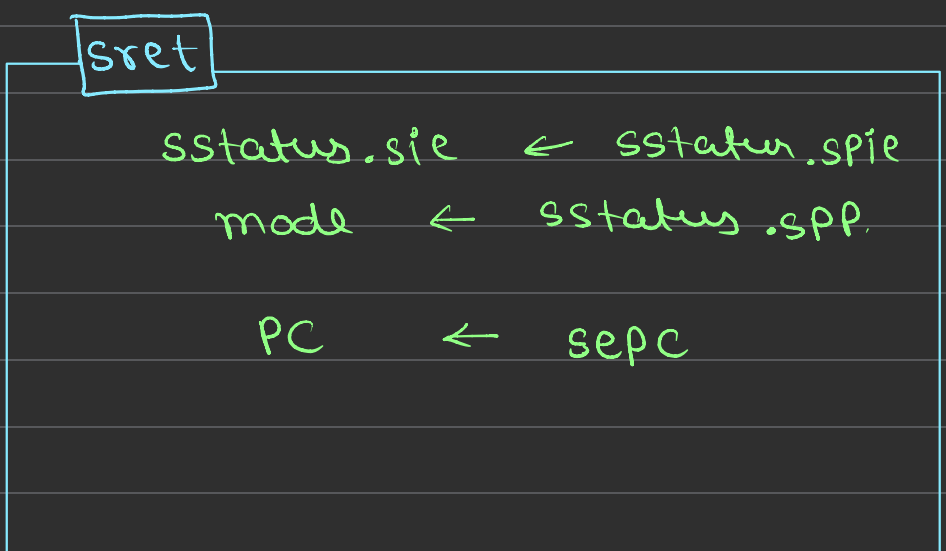
// change the mode to supervisor (if its not supervisor)

mode ← supervisor.

And then the first instruction of the trap handler executes.

Handler code do something

Later, the handler returns to the interrupted code.

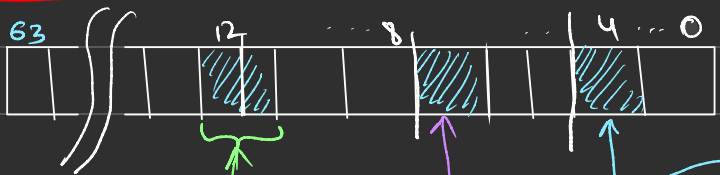


That's what happens in supervisor mode and most of the runs in the supervisor mode

But there's a little bit of the kernel that runs in machine mode.

Trap Processing in machine mode

mstatus



mie

Interrupts enabled

0 = disabled

1 = enabled

mpie

Previous interrup Enabled

mpp

Previous Privilege level (Previous mode)

00 → user mode

01 → supervisor mode

11 → machine mode

- Only Interrupt we deal with at machine mode level is Timer Interrupt.
  [All others go to ("delegated") supervisor mode.

- Interrupts are always enabled at machine mode.
  (whenever a timer interrupt occurs it will be handled, whenever any trap of other kind happens it will be automatically delegated to supervisor mode)

- On timer Interrupt a machine mode handler will run and that code:
  - force a "software" Interrupt to supervisor mode
  - Re-enable Interrupts.

- Return To ... the interrupted code ...

Trap Processing in machine mode.

Same Idea

only Cause : Timer Interrupts.

All other types are "Delegated" to supervisor mode

mtvec: CSR containing address of handler.

↳ in XV6, that code is timervec function

Hardware Actions

mepc ← PC

PC ← mtvec

mcause ← ... } ignored.

mtval ← ... }

mstatus.mPP ← Previous mode

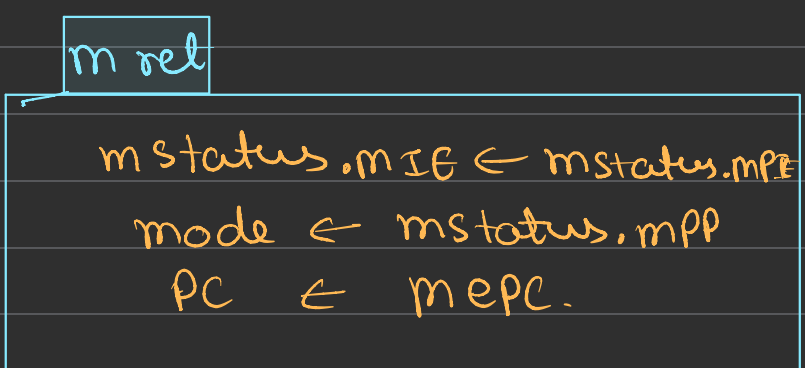
mstatus.mPIE ← mstatus.MIE

mstatus.MIE ← ϕ

mode ← Machine

The handler will handle the Interrupt.

The handler will trigger / cause an interrupt at the supervisor level.



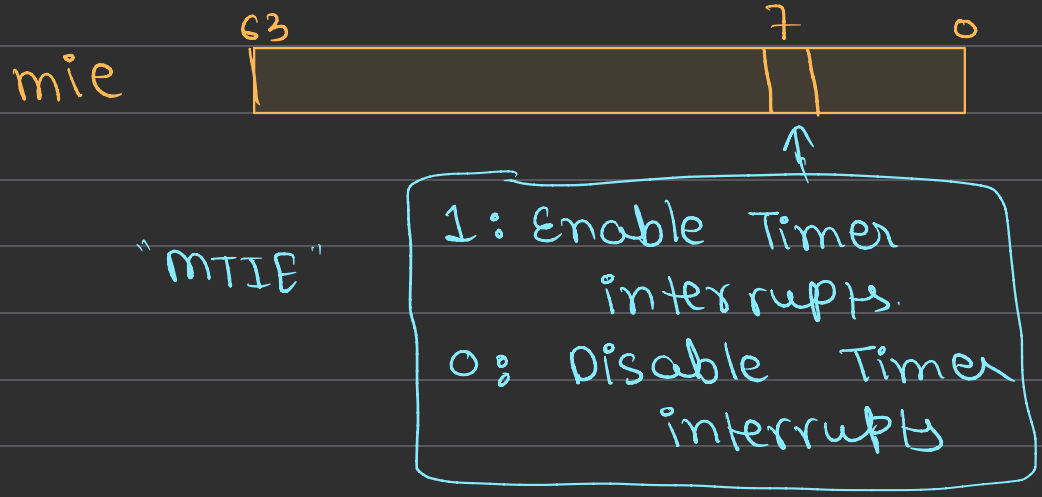
Next instruction

- Interrupts Disabled ← "Software" Interrupt Pending
- Interrupts Enabled ← A trap occurs at supervisor

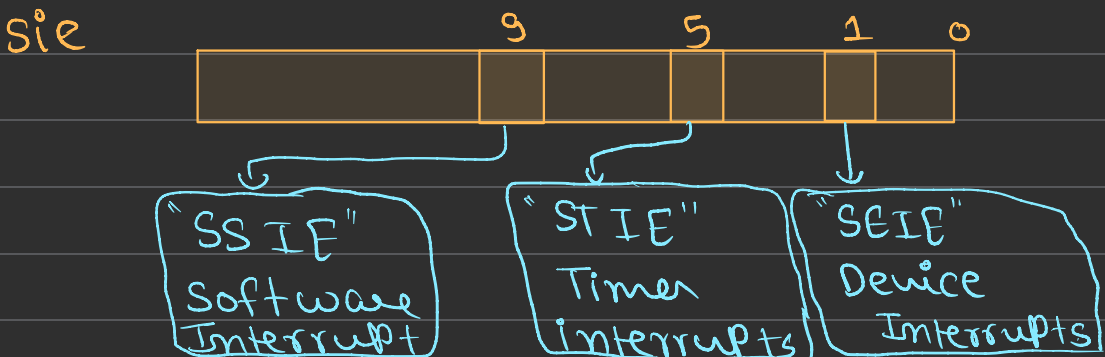
level

Global "Interrupts Enabled" bit in mstatus, sstatus CSR.

more selective control.



set during machine mode initialization



SIP: Same as above.

1: An interrupt is pending
0: No interrupt.

Trap Delegation

All traps in RISC-V architecture will go to machine mode handlers

But there is also a facility to sort of bypass the machine mode handler and go straight to handler that's executing at supervisor mode level.

XV6 delegates all traps to supervisor mode.

There are two registers:

Hardware Delegation Registers

- **medeleg** — which exceptions to delegate
- **mideleg** — which interrupts to delegate

(CSRs)

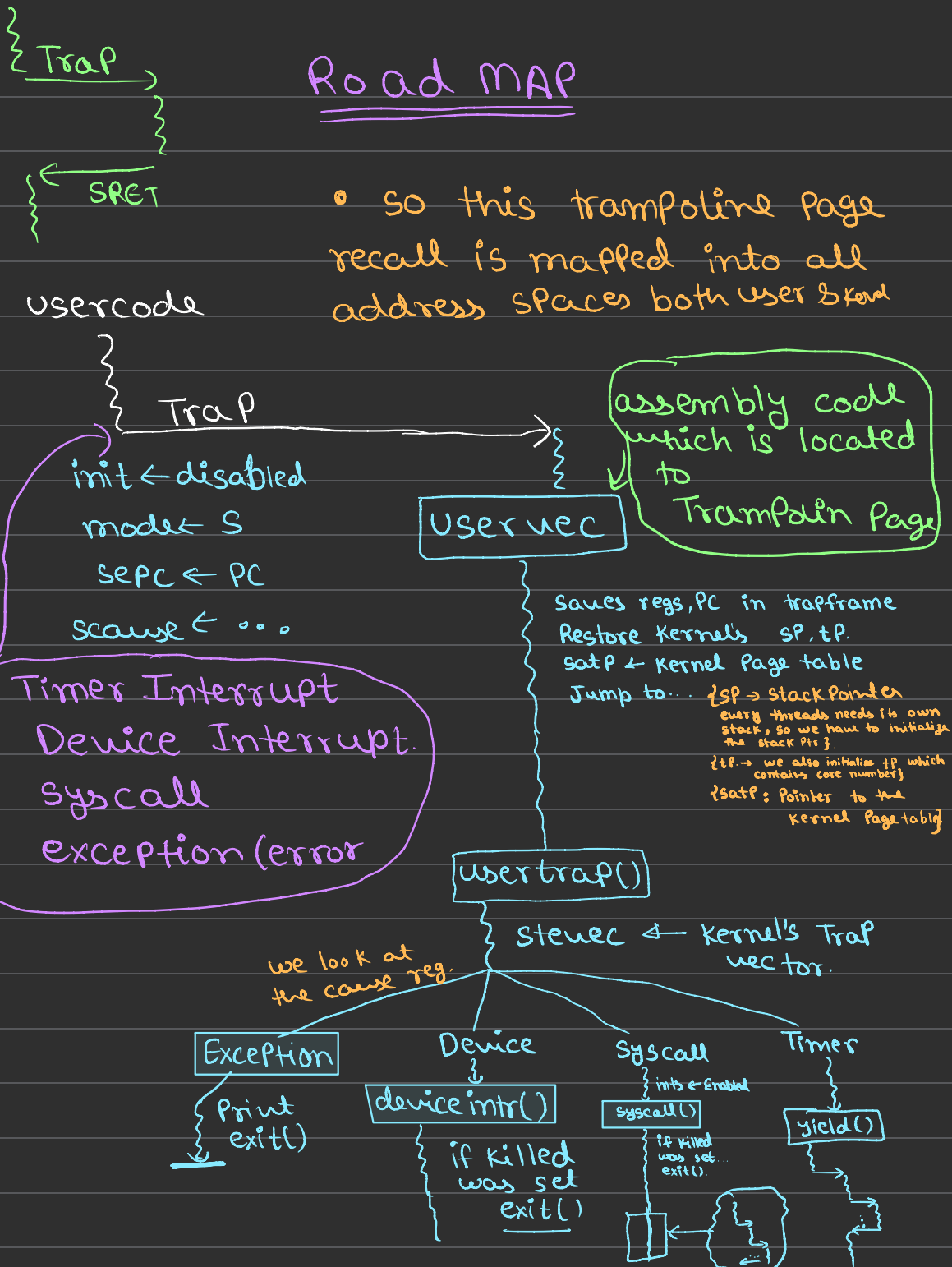
XV6 delegates all traps to supervisor mode.

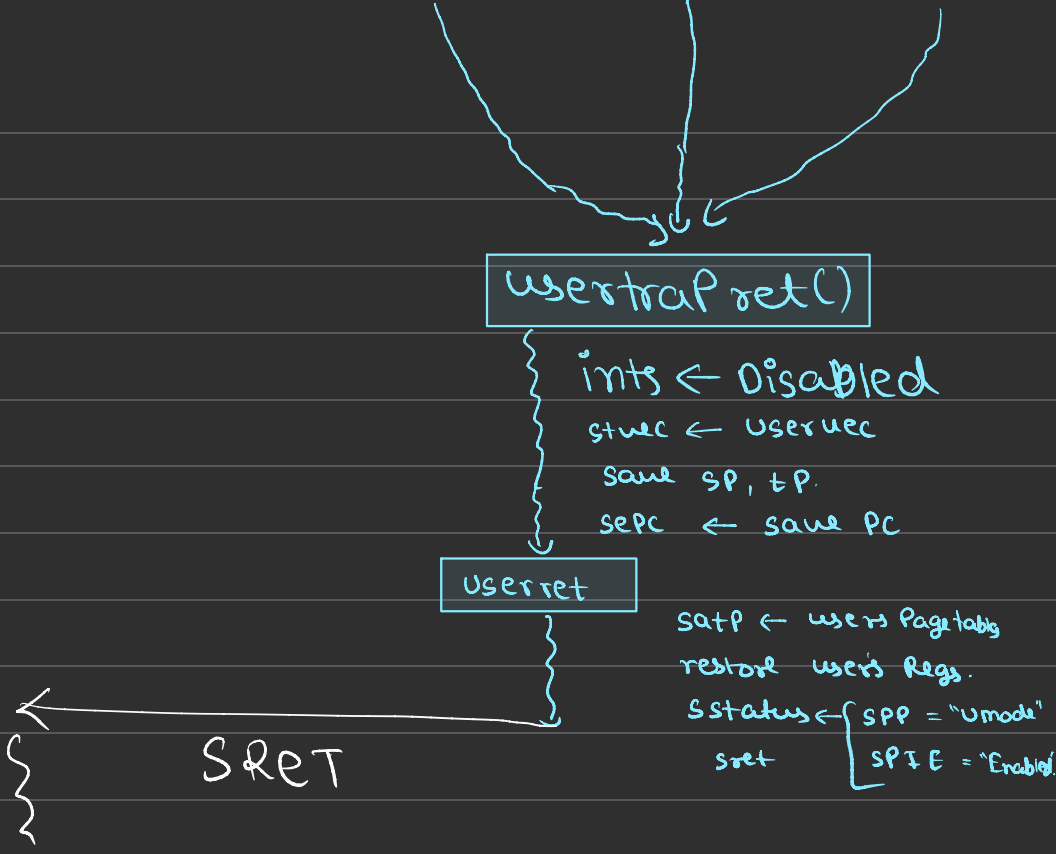
Timer Interrupt cannot be delegated. It is handled differently.

medeleg, mideleg are initialized during setup (in "start" in machine mode).

when a trap occurs

- No machine mode code.
- See TrapProcessing for supervisor mode


Trap
SRET
Road MAP
• so this trampoline page recall is mapped into all address spaces both user & Kernel
usercode
Trap
assembly code which is located to Trampoin Page
init ← disabled
mode ← S
sepc ← PC
scause ← ...
uservec
Saves regs, PC in trapframe
Restore Kernel's SP, tP.
satP ← Kernel Page table
Jump to...
{SP → Stack Pointer
every threads needs its own stack, so we have to initialize the stack Ptr.}
{tP → we also initialize tP which contains core number}
{satP: Pointer to the Kernel Page table}
• Timer Interrupt
• Device Interrupt
• Syscall
• exception (error
usertrap()
stevec ← Kernel's Trap vector.
we look at the cause reg.
Exception
Device
Syscall
Timer
Print exit()
deviceintr()
intr ← Enabled
syscall()
yield()
if killed was set exit()
if killed was set exit()


usertrapret()
ints ← Disabled
stvec ← uservec
save sp, tp
sepc ← save pc
userret
satp ← users Pagetable
restore users Regs.
Sstatus ← SPP = "Umode"
SPIE = "Enabled"
sret
SRET