ARM EXCEPTIONS AND INTERRUPTS

By: Muhammad ElZeiny

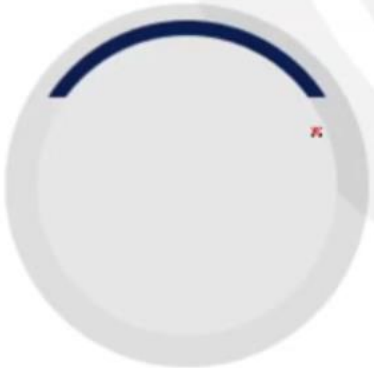
EXCEPTION STATES

INACTIVE

The exception is not active and not pending.

PENDING

The exception is waiting to be serviced by the processor.



ACTIVE

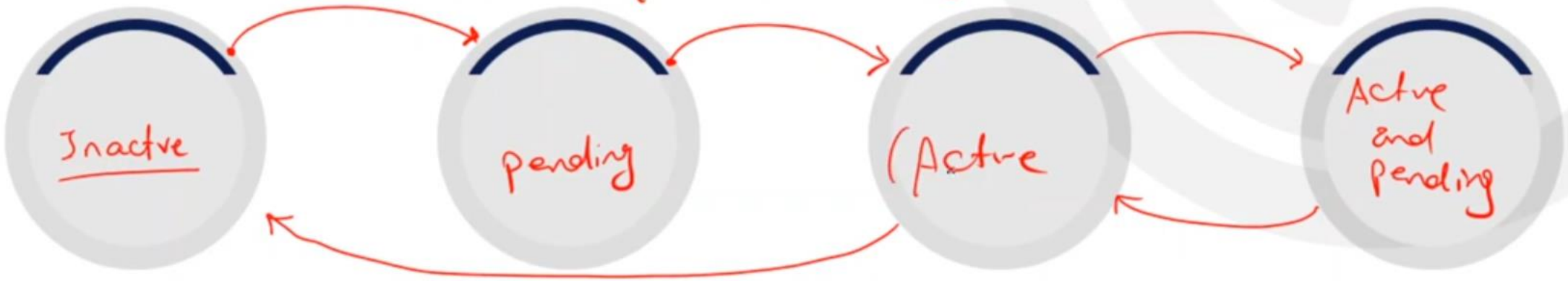
An exception that is being serviced by the processor but has not completed.

Note: An exception handler can interrupt the execution of another exception handler. In this case, both exceptions are in the active state.

ACTIVE AND PENDING

An exception that is being serviced by the processor but **has not completed**

EXCEPTION STATES



INACTIVE

The exception is not active and not pending.

PENDING

The exception is waiting to be serviced by the processor.

ACTIVE

An exception that is being serviced by the processor but has not completed.

Note: An exception handler can interrupt the execution of another exception handler. In this case, both exceptions are in the active state.

ACTIVE AND PENDING

An exception that is being serviced by the processor but **has not completed**

EXCEPTION TYPES

RESET

- Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception.
- Execution restarts as privileged execution in Thread mode.

NMI

- A non-maskable Interrupt (NMI) can be signaled using the NMI signal or triggered by software using the Interrupt Control and State (INTCTRL) register.
- This exception has the highest priority other than reset. NMI is permanently enabled and has a fixed priority of **-2**.
- NMIs **cannot be masked or prevented** from activation by any other exception or preempted by any exception other than reset

HARD FAULT

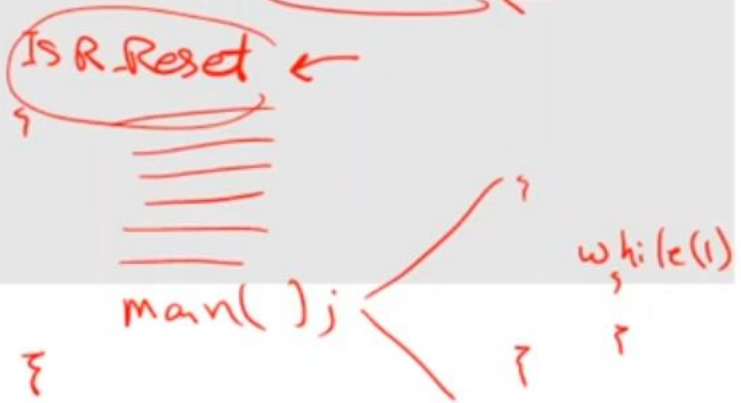
- A hard fault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism.
- Hard faults have a fixed priotity of 1- meaning they have higher priority than any exception with configurable priority.

EXCEPTION TYPES

↳ Handler Mode ← Serve ISR
Thread [ ]

RESET

- Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception.
- Execution restarts as privileged execution in Thread mode.



NMI

- A non-maskable Interrupt (NMI) can be signaled using the NMI signal or triggered by software using the Interrupt Control and State (INTCTRL) register.
- This exception has the highest priority other than reset. NMI is permanently enabled and has a fixed priority of **-2**.
- NMIs **cannot be masked or prevented** from activation by any other exception or preempted by any exception other than reset

HARD FAULT

- A hard fault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism.
- Hard faults have a fixed priotity of 1- meaning they have higher priority than any exception with configurable priority.

EXCEPTION TYPES

MVP

MEMORY MANAGEMENT FAULT

• A memory management fault is an exception that occurs because of a memory protection related fault, including access violation and no match.
• The MPU or the fixed memory protection constraints determine this fault, for both instruction and datamemory transactions.

BUS FAULT

•• A bus fault is an exception that occurs because of a memory-related fault for an instruction or data memory
transaction such as a prefetch fault or a memory access fault.

• This fault can be enabled or disabled.

USAGE FAULT

• A usage fault is an exception that occurs because of a fault related to instruction

• execution, such as:

• An undefined instruction
Invalid state on instruction execution
An error on exception return
division by zero

EXCEPTION TYPES

SVCALL

·A supervisor call (SVC) is an exception
that is triggered by the SVC instruction.

· In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.

PENDSV

· PendSV is a pendable, interrupt-driven request for system-level service.

· In an OS environment, use PendSV for context switching when no other exception is active.

· PendSV is triggered using the Interrupt Control and State (INTCTRL) register.

SYSTICK

· A SysTick exception is an exception that the system timer generates when it reaches zero when it is enabled to generate an interrupt.

· Software can also generate a SysTick exception using the Interrupt Control and State (INTCTRL) register.

· In an OS environment, the processor can use this exception as system tick.

EXCEPTION TYPES

INTERRUPTS

- An interrupt, or IRQ, is an exception signaled by a peripheral or generated by a software request and fed through the NVIC (prioritized).

- All interrupts are asynchronous to instruction execution.

- In the system, peripherals use interrupts to communicate with the processor.

DEBUG MONITOR

- This exception is caused by the debug monitor.

- This exception does not activate if it is a lower priority than the current activation.

EXCEPTION HANDLER

INTERRUPT SERVICE ROUTINES (ISRS)

1. Interrupts (IRQx)
   - are the exceptions handled by ISRs.

FAULT HANDLERS

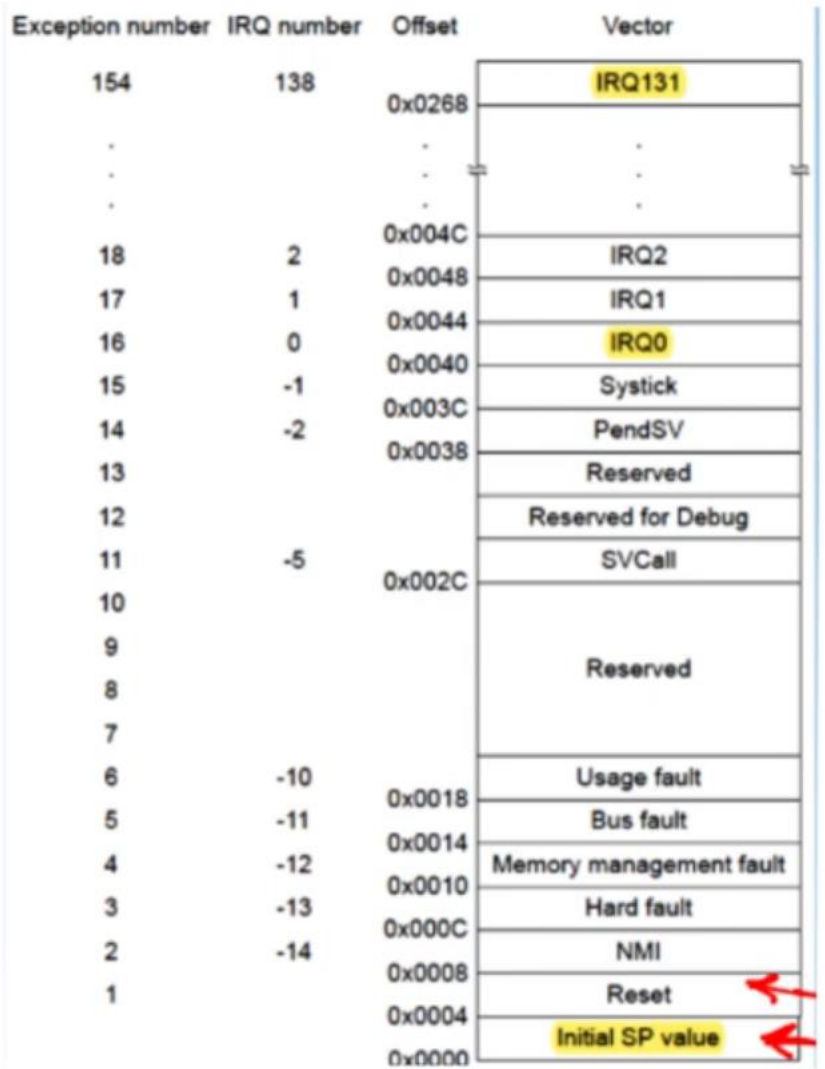
1. Hard fault
2. memory management fault
3. usage fault
4. bus fault
   - are fault exceptions handled by the fault handlers.

SYSTEM HANDLERS

1. NMI
2. PendSV
3. SVCall
4. SysTick

Vector Table

- The vector table contains the reset value of the stack pointer and the start addresses of all handlers and ISR.
- On system reset, the vector table is fixed at address 0x0000.0000.
- Privileged software can relocate the vector table start address to a different memory location



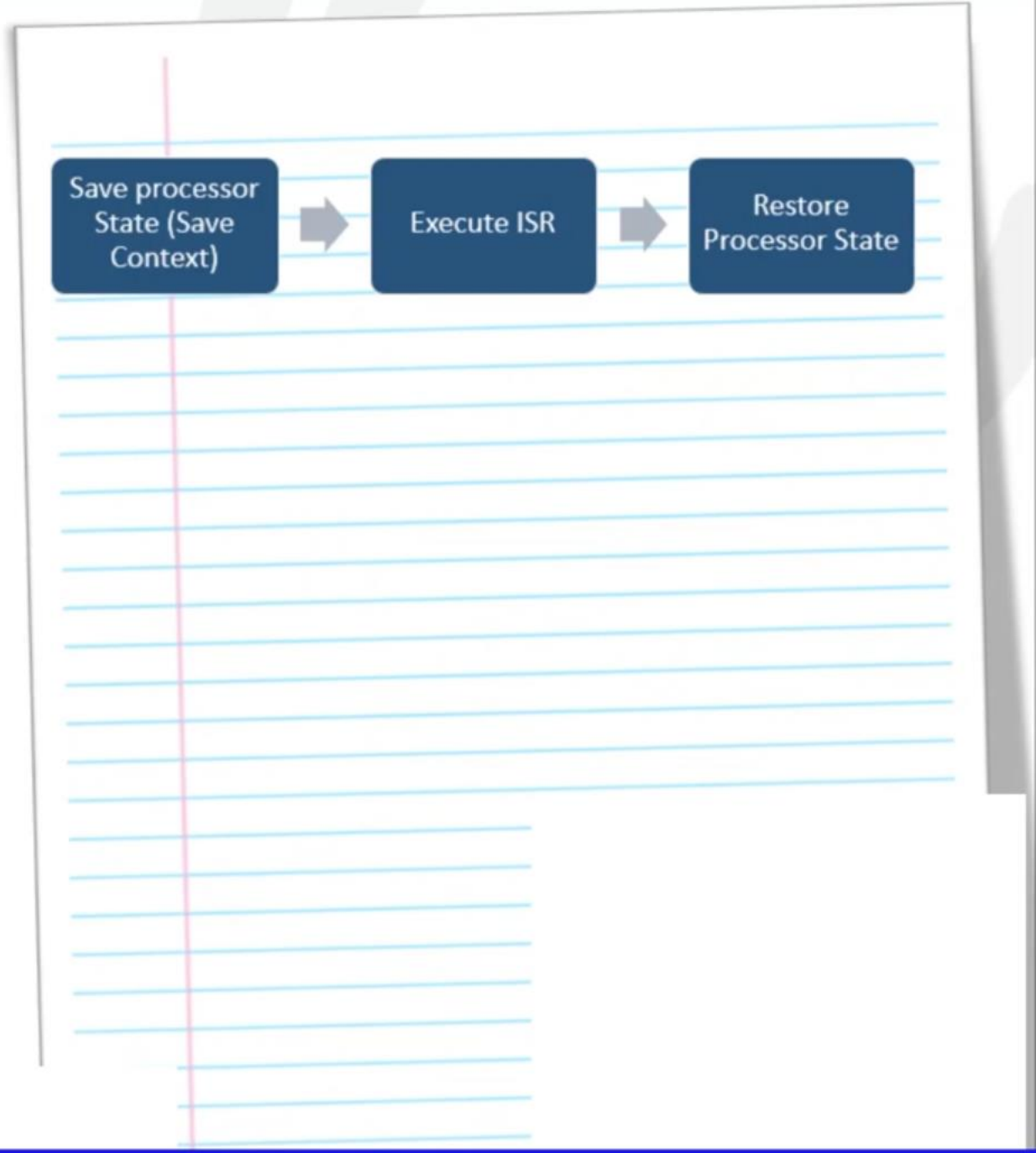
EXCEPTION RETURN AND ENTRY

Preemption

- When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled.

Return

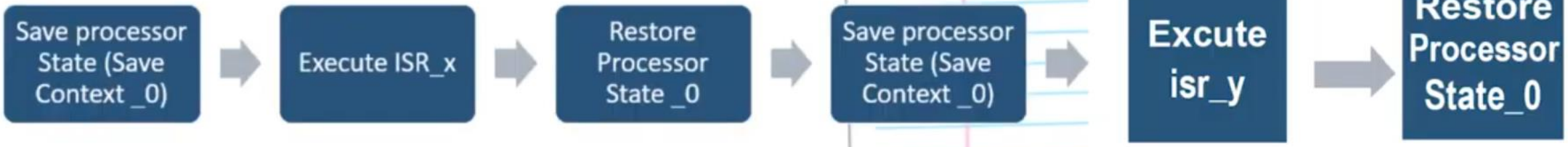
- Return occurs when the exception handler is completed, **and** there is no pending exception with sufficient priority to be serviced and the completed exception handler was not handling a late-arriving exception.
- The processor pops the stack and restores the processor state to the state it had before the interrupt occurred.



EXCEPTION RETURN AND ENTRY

Tail Chaining

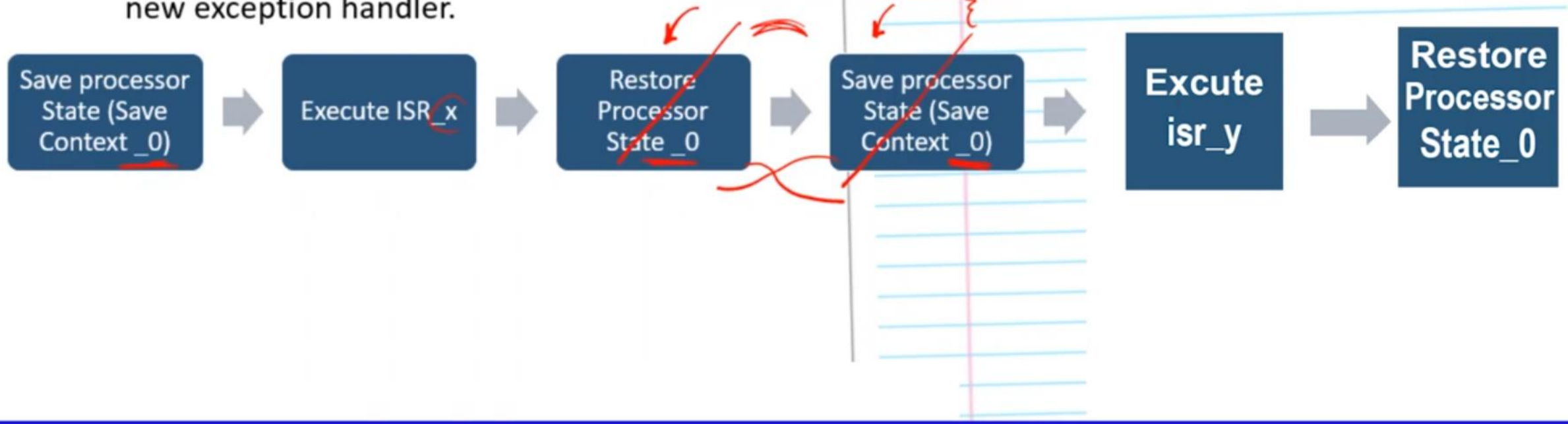
- This mechanism speeds up exception servicing.
- On completion of an exception handler, if there is a pending that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.



EXCEPTION RETURN AND ENTRY

Tail Chaining

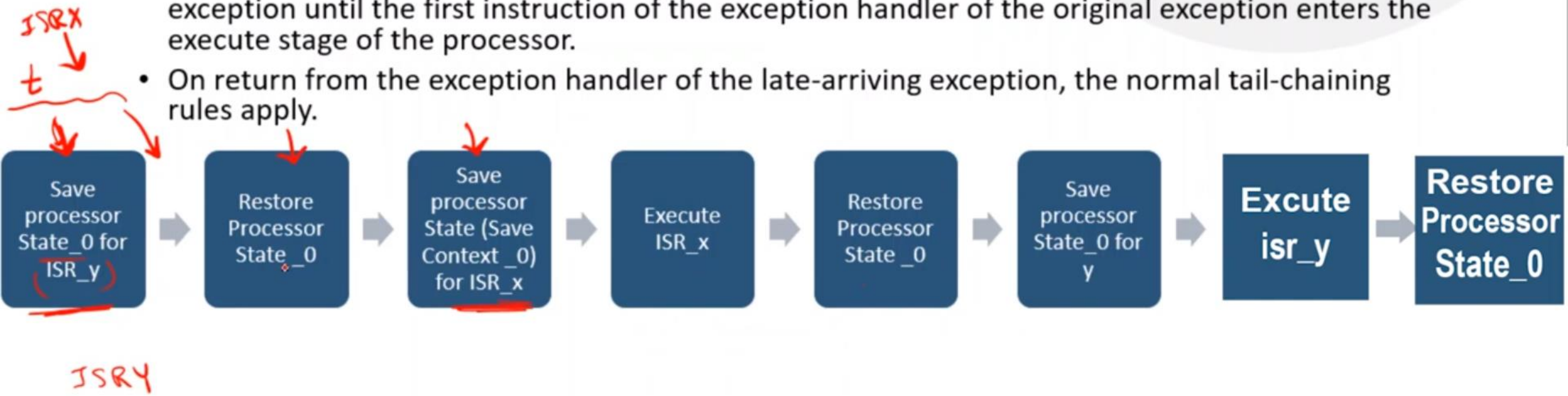
- This mechanism speeds up exception servicing.
- On completion of an exception handler, if there is a pending that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.



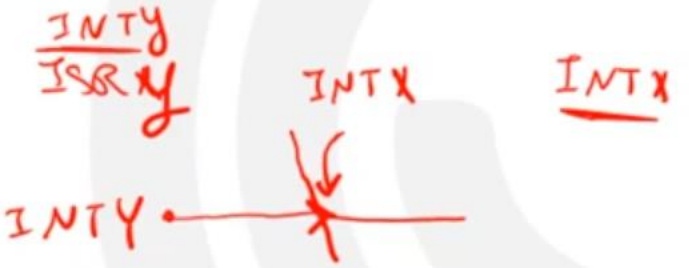
EXCEPTION RETURN AND ENTRY

Late Arriving

- This mechanism speeds up preemption.
- If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception.
- State saving is not affected by late arrival because the state saved is the same for both exceptions.
- Therefore, the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor.
- On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.

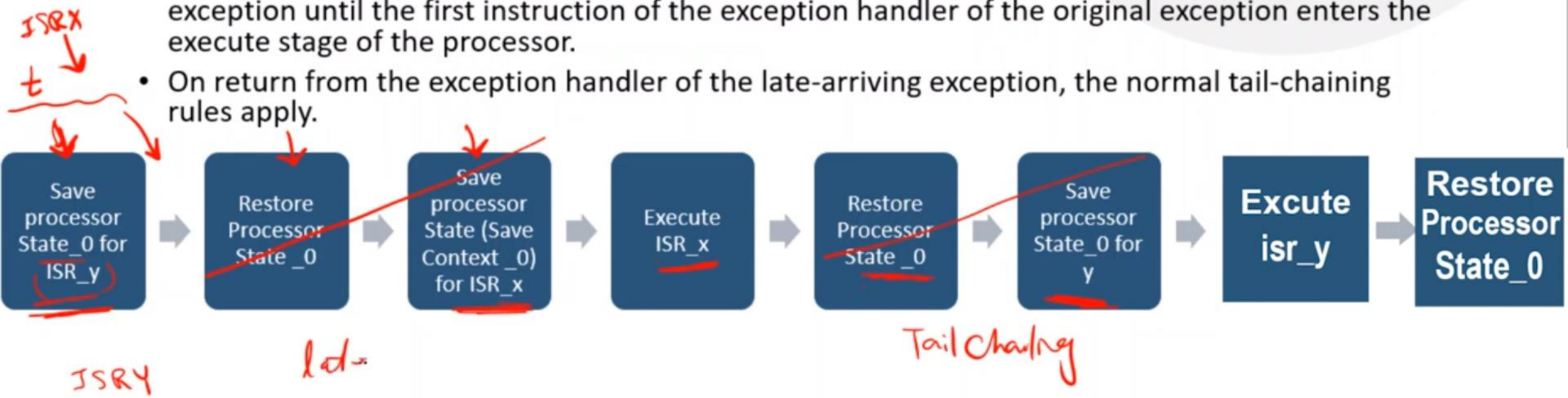


EXCEPTION RETURN AND ENTRY



Late Arriving

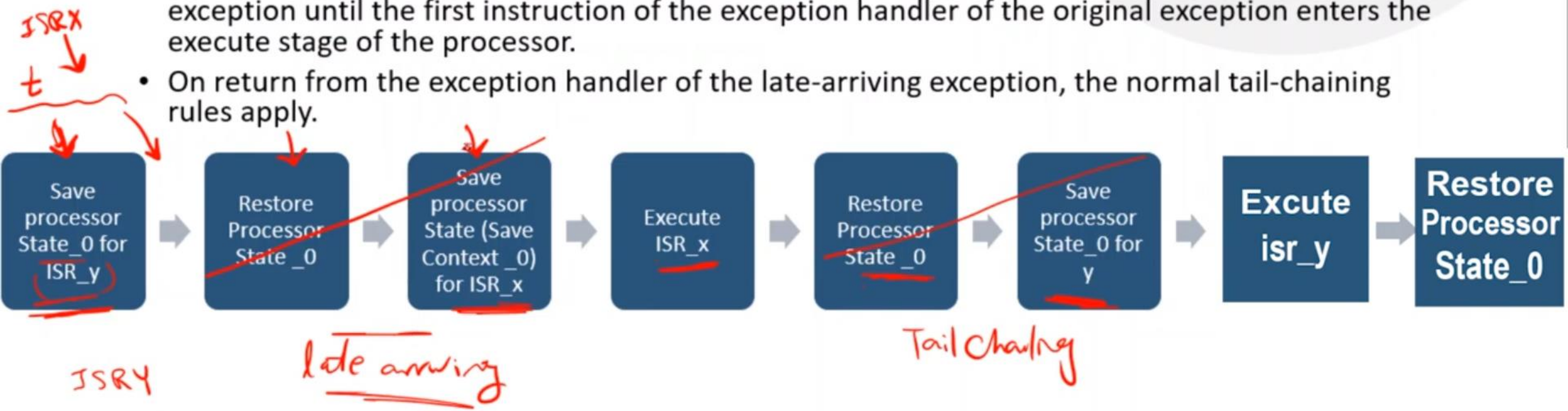
- This mechanism speeds up preemption.
- If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception.
- State saving is not affected by late arrival because the state saved is the same for both exceptions.
- Therefore, the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor.
- On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.



EXCEPTION RETURN AND ENTRY

Late Arriving

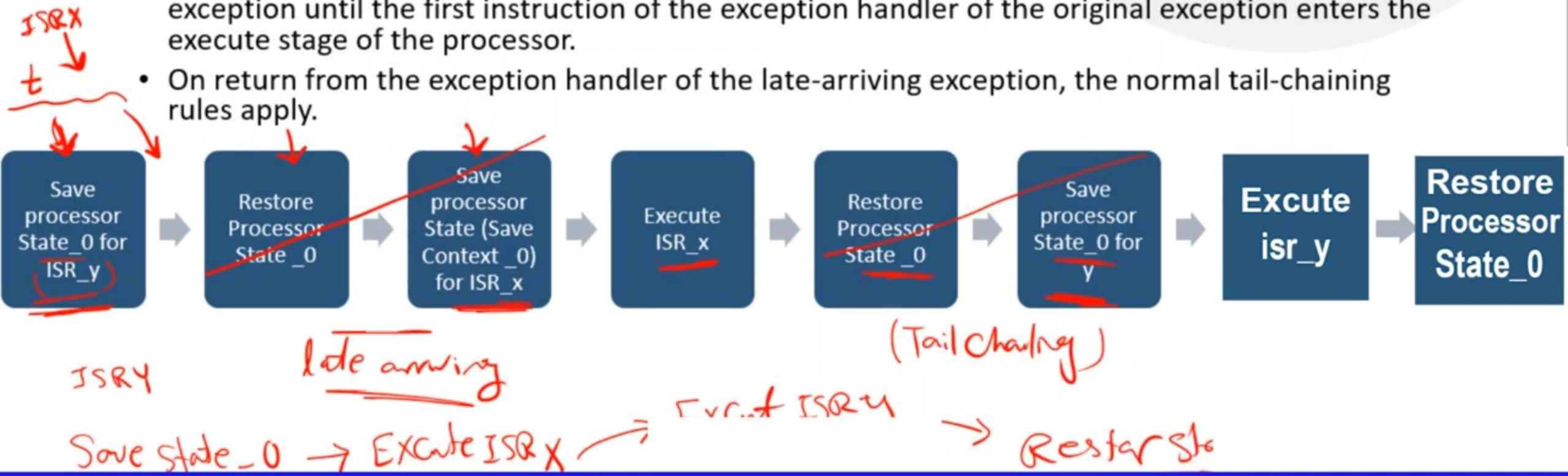
- This mechanism speeds up preemption.
- If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception.
- State saving is not affected by late arrival because the state saved is the same for both exceptions.
- Therefore, the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor.
- On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.



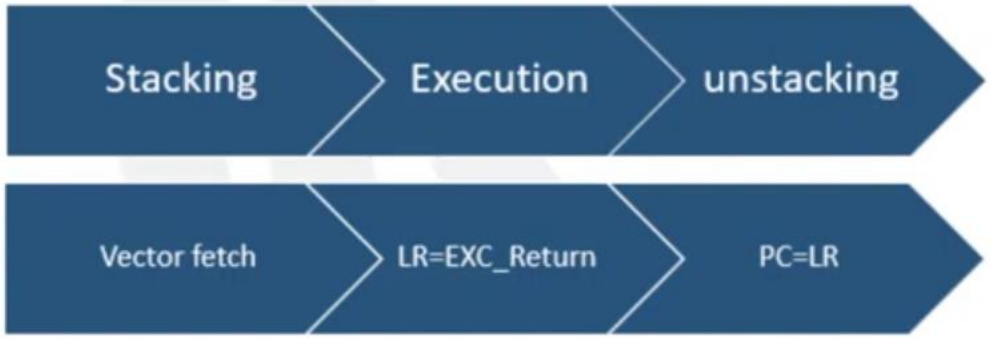
EXCEPTION RETURN AND ENTRY

Late Arriving

- This mechanism speeds up preemption.
- If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception.
- State saving is not affected by late arrival because the state saved is the same for both exceptions.
- Therefore, the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor.
- On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.

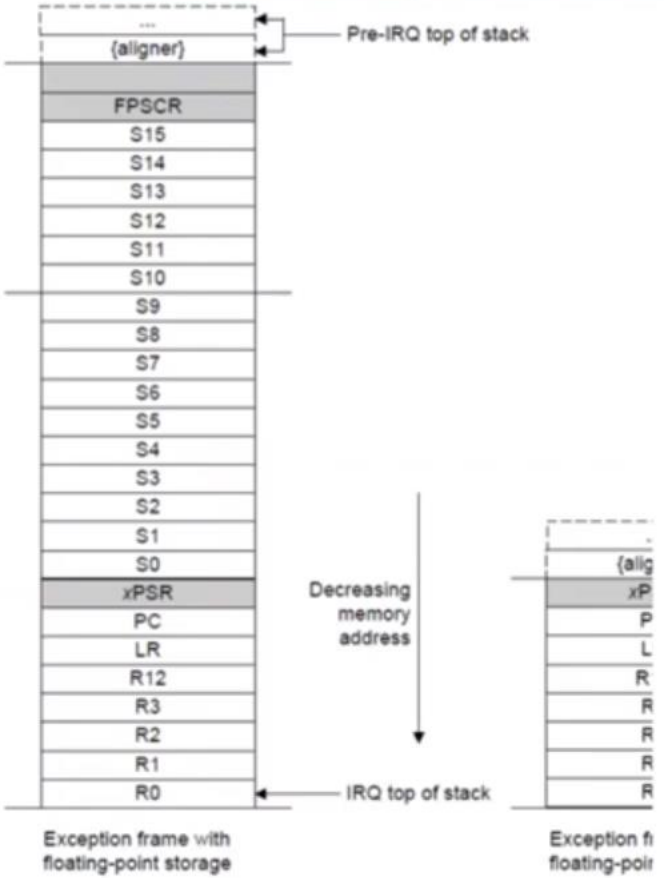


EXCEPTION ENTRY AND RETURN



- When the processor takes an exception, the processor pushes the stack frame (eight data words)
- In parallel the processor performs a vector fetch i.e. read the exception handler address.
- When stacking is complete the processor starts executing the exception handler.
- At the same time, the processor writes an EXC_RETURN value to the LR
- When the exception is complete the EXC_RETURN loaded into the PC

- The EXC_RETURN lowest five bits provides information of
  - return stack (Process\main)
  - processor mode(Handler\Thread).
  - EXC_RETURN bits 31:5 are all set.



LEVEL-SENSITIVE AND PULSE INTERRUPTS

LEVEL-SENSITIVE INTERRUPT

- Interrupt signal is held asserted until the peripheral de-asserts the interrupt signal.
- Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request.
- if the signal is not de-asserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must re-enter its ISR again.

NOTE: When the processor enters the ISR, it automatically removes the pending state from the interrupt – The state will become pending again if the clear-pending register bit is not cleared.

PULSE INTERRUPT EDGE TRIGGERED)

- Interrupt signal sampled synchronously on the rising edge of the processor clock.
- the NVIC continues to monitor the interrupt signal, and if this is pulsed the state of the interrupt changes to **pending and active**. In this case, when the processor returns from the ISR the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR.
- If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.

FAULTS TYPE

Hard fault

- Bus error on a vector read
- Fault escalated to a hard fault

Memory management fault

- MPU or default memory mismatch on instruction access
- MPU or default memory mismatch on data access
- MPU or default memory mismatch on exception stacking
- MPU or default memory mismatch on exception unstacking
- MPU or default memory mismatch during lazy floating-point state preservation

Bus fault

- Bus error during exception stacking
- Bus error during exception unstacking
- Bus error during instruction prefetch

Usage fault

- Attempt to access a coprocessor
- Undefined instruction
- Attempt to enter an invalid instruction
- Invalid EXC_RETURN value
- Illegal unaligned load or store
- Divide by 0

HARD FAULT ESCALATION

• In the following situations, a fault with configurable priority is treated as a hard fault :

1. A fault handler causes the same kind of fault as the one it is servicing.
It occurs because a fault handler cannot preempt itself because it must have the same priority as the current priority level.

2. A fault handler causes a fault with the same or lower priority as the fault it is servicing.
This situation happens because the handler for the new fault cannot preempt the currently executing fault handler.

3. An exception handler causes a fault for which the priority is the same as or lower than the currently executing exception.

4. A fault occurs and the handler for that fault is not enabled.