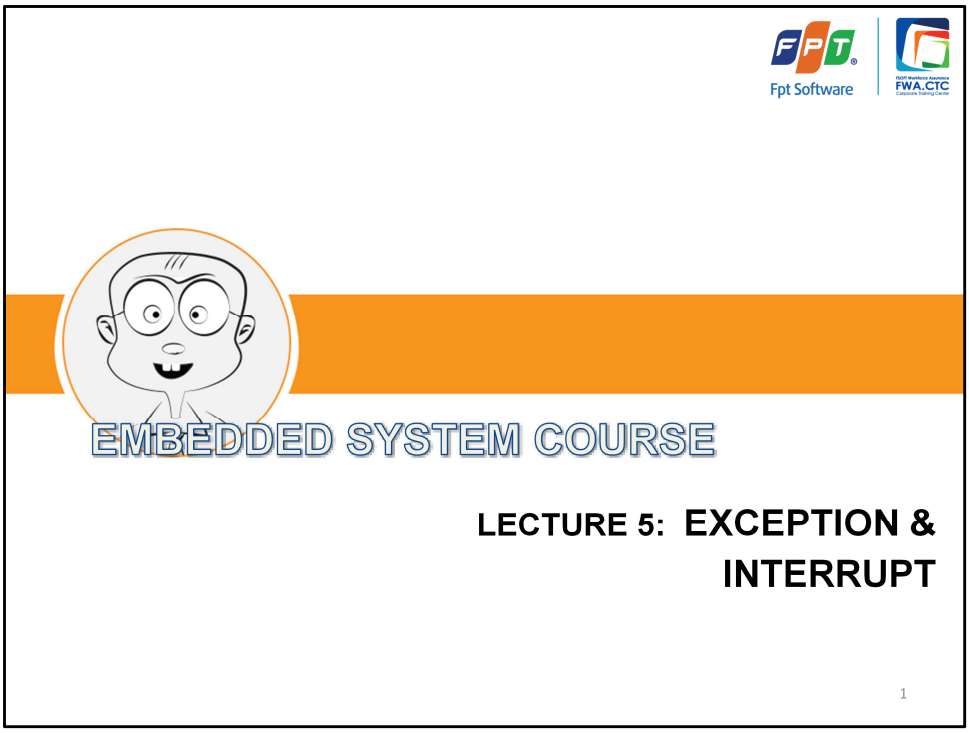


Questions:

- PC register align theoe word/halfword/byte? Tại sao ?
- What is the different between Thumb & Thumb-2.
- What is the different between Handler mode & Thread mode.

Introduce some keyword:

- **What is interrupt/exception ?**

Interrupts are events typically generated by hardware (e.g., peripherals or external input pins) that cause changes in program flow control outside a normal programmed sequence (e.g., to provide service to a peripheral).
In ARM terminology, an interrupt is one type of exception. Other exception kinds are: fault exception// sysTick Timer exception // system exeption support OS (SVC instruction); etc.

- **What is the exception handler?** The pieces of program code that handle exceptions. They are part of the compiled program image.

➔ Cortex-M processors provide a **Nested Vectored Interrupt Controller (NVIC)** for interrupt handling.

➔ The lecture focuses on NVIC implementation & exception management.

Learning Goals

- Describe detail about the Cortex-M interrupt handling (to be more specific: NVIC controller).
- Describe about the exception entry/return sequence and how to optimize the interrupt latency.

2

Cortex-M provides a Nested Vectored Interrupt Controller (NVIC) for interrupt handling.

Exception entry sequence: Staking, Update registers, Fetch vector number & instruction of exception handler.

Exception return sequence: Unstacking, Fetch & execute from the restored return address

Exception handling optimization: Tail chaining, Late arrival, pop preemption.

Table of contents

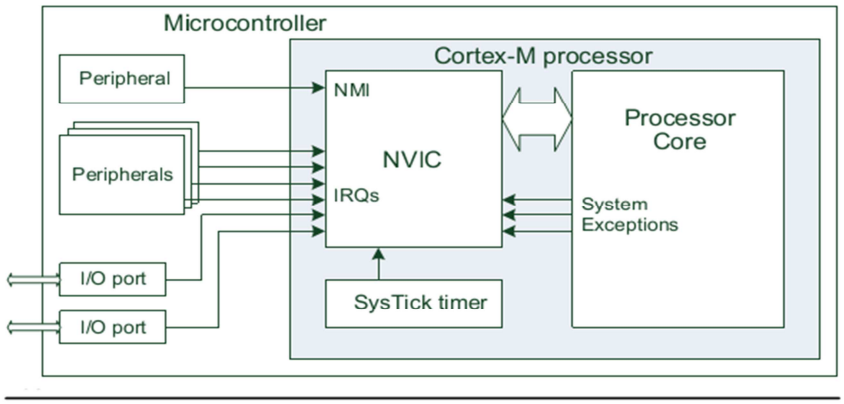
3

Giới thiệu nội dung tổng quan bài giảng: (Gồm 5 phần chính)
1- Tổng quan chung về quản lý ngắt trong ARM-Cortex M
2- Khái niệm Vector Table / Vector table relocation/ Implement Vector table.
3- Giới thiệu chi tiết về các thanh ghi phục vụ cho việc quản lý ngắt trong khối NVIC và SCB
4- Quá trình khi xảy ra một exeption
5- Interrupt lantency và các cơ chế giảm interrupt latency của ARM-Cortex M
(Khái niệm Interrupt latency: Là thời gian tính từ khi interrupt xảy ra tới khi trình phục vụ ngắt bắt đầu được thực hiện)
6- Tổng kết lại buổi học

Table of contents

Overview of interrupt management

- Exception sources



Various sources of exceptions in a typical microcontroller

5

- All Cortex-M processors provide a Nested Vectored Interrupt Controller (NVIC) for interrupt handling. The component receives interrupt requests from various sources: Peripherals & System (such as: from processor core (hard fault, SVC instruction, etc), system tick)

Overview of interrupt management

- System Exceptions and Interrupt

Excp. Number	Exception Type	Priority
1	Reset	-3 (highest)
2	NMI	-2
3	Hard Fault	-1
4	MemManage Fault	Programmable
5	Bus Fault	Programmable
6	Usage Fault	Programmable
7-10	Reserved	NA
11	SVC	Programmable
12	Debug Monitor	Programmable

Excp. Number	Exception Type	Priority
13	Reserved	NA
14	PendSV	Programmable
15	SysTick	Programmable
16	Interrupt #0	Programmable
17	Interrupt #1	Programmable
...		
47	Interrupt #31	Programmable
...		
255	Interrupt #239	Programmable

6

- Note: ARM Cortex M0/M1: support 32 interrupt sources; M3/M4: up to 240.
- Reset, NMI, Hard Fault: fix priority.

Overview of interrupt management

Exception Properties

- Exception number: Identification for the exception
- Vector address: Exception entry point in memory
- Priority level: Determines the order in which multiple pending exceptions are handled

Table of contents

Vector Table

- ***Vector Table:***
  - After powerup, vector table is located at 0x00000000
  - Vector table contains:
    - Initial value for Main Stack Pointer
    - Handler vector addresses

Address	Exception Number	Value (Word Size)
0x00000000	-	MSP initial value
0x00000004	1	Reset vector
0x0000008	2	NMI handler starting address
0x000000C	3	Hard fault handler starting address
……	…….	Other handler starting address

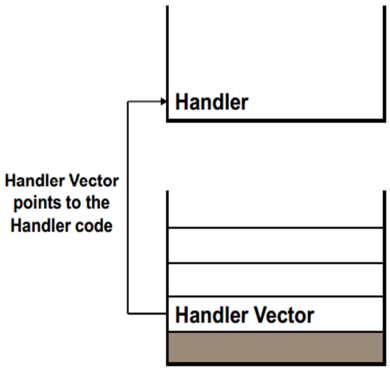
9

Vector Table

Vector Table Usage:

In the case of an exception, the core:

- Reads the vector handler address for the exception from the vector table
- Branches to the handler

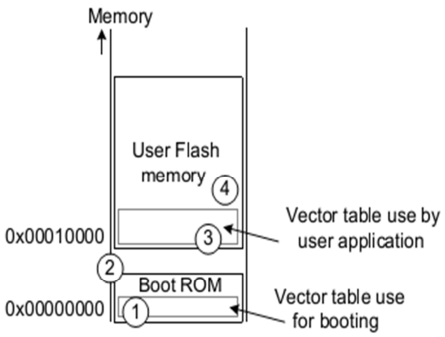


10

Vector Table & Vector Table Relocation

Vector Table Relocation:

- Vector table can be relocated to change interrupt handlers at run-time
- Some cases using vector table relocation feature:
  - Boot loader
  - Applications load to RAM
  - Dynamic changing of Vector



11

Example for vector table relocation :
Devices with boot loader
The boot loaders are often pre-programmed in the boot ROM by the microcontroller manufacturer. When the microcontrollers start, they first execute
the boot loader code in the boot ROM, and before branching to the user application in the user flash, the VTOR is programmed to point to the starting point of the user flash memory so that the vector table in user flash will be used.

Boot loader Step
1) Boot up using vector table in boot ROM
2) Carry out boot loader tasks
3) Program VTOR to point to vector table in user flash
4) Branch to the reset handler indicated by the vector table of user flash memory

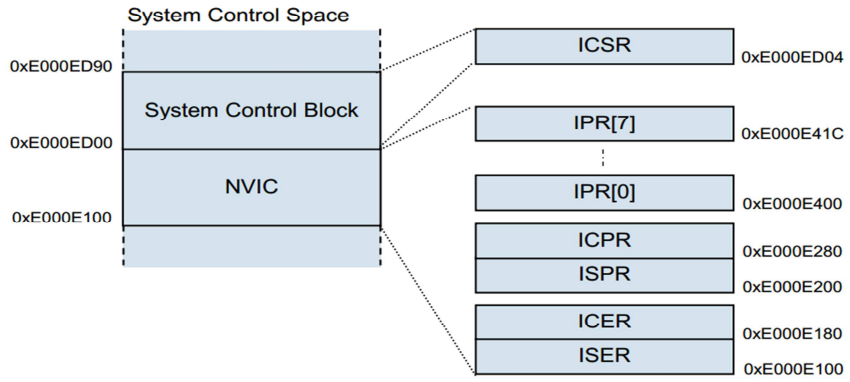
Vector Table

Vector Table Implementation (Startup.s & linker file):

```
; Vector Table Mapped to Address 0 at Reset

                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors
                EXPORT  __Vectors_End
                EXPORT  __Vectors_Size

__Vectors       DCD     __initial_sp   ; Top of Stack
                DCD     Reset_Handler  ; Reset Handler
                DCD     NMI_Handler    ; NMI Handler
                DCD     HardFault_Handler  ; Hard Fault Handler
                DCD     0  ; Reserved
                DCD     0  ; Reserved
                DCD     0  ; Reserved
                DCD     0  ; Reserved


; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x00000000 0x00020000  {    ; load region size_region
  ER_IROM1 0x00000000 0x00020000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
```

Table of contents

Details of NVIC & SCB registers



SCB & NVIC Register Set for Cortex M0

14

ISER	Interrupt Set Enable Register
ICER	Interrupt Clear Enable Register
ISPR	Interrupt Set Pending Register
ICPR	Interrupt Clear Pending Register
IPR[0] to IPR[7]	Interrupt Priority Register

Details of NVIC & SCB registers

NVIC registers

- ***Interrupt Priority Level Register (IPR0 – IPR7):***

Address	Name	Type	Reset Value	Descriptions
0xE000E400	PRIORITY0	R/W	0x00000000	Priority level for interrupt 0 to 3 [31:30] Interrupt priority 3 [23:22] Interrupt priority 2 [15:14] Interrupt priority 1 [7:6] Interrupt priority 0

Address	Name	Type	Reset Value	Descriptions
0xE000E404	PRIORITY1	R/W	0x00000000	Priority level for interrupt 4 to 7
0xE000E408	PRIORITY2	R/W	0x00000000	Priority level for interrupt 8 to 11
0xE000E40C	PRIORITY3	R/W	0x00000000	Priority level for interrupt 12 to 15
0xE000E410	PRIORITY4	R/W	0x00000000	Priority level for interrupt 16 to 19
0xE000E414	PRIORITY5	R/W	0x00000000	Priority level for interrupt 20 to 23
0xE000E418	PRIORITY6	R/W	0x00000000	Priority level for interrupt 24 to 27
0xE000E41C	PRIORITY7	R/W	0x00000000	Priority level for interrupt 28 to 31

15

- **Interrupt Priority Level:**

Each external interrupt has an associated priority-level register. Each of them is 2 bits wide, occupying the two MSBs of the Interrupt Priority Level Registers. Each Interrupt Priority Level Register occupies 1 byte (8 bits).

The Interrupt Priority Level Registers should be programmed before the interrupt is enabled. Changing of interrupt priority when the interrupt is already enabled should be avoided, as this is architecturally unpredictable in the ARMv6-M architecture and is not supported in Cortex-M0 processor.

Details of NVIC & SCB registers

NVIC registers

- ***Interrupt Clear Pending Register***

Address	Name	Type	Reset Value	Descriptions
0xE000E280	CLRPEND	R/W	0x00000000	Clear pending for interrupt 0 to 31; write 1 to clear bit to 0, write 0 has no effect Bit[0] for Interrupt #0 (exception #16) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current pending status

- ***Interrupt Set Pending Register***

Address	Name	Type	Reset Value	Descriptions
0xE000E200	SETPEND	R/W	0x00000000	Set pending for Interrupts 0 to 31; write 1 to set bit to 1, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) Bit[1] for Interrupt #1 (exception #17) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current pending status

16

- If an interrupt takes place but cannot be executed immediately (for instance, if another higher-priority interrupt handler is running), it will be pended.
- The interrupt-pending status can be accessed through the Interrupt Set Pending (NVIC->ISPR) and Interrupt Clear Pending (NVIC->ICPR) registers.
- **Interrupt Set Pending Register:** Write 1 to set pending status
- **Interrupt Clear Pending Register:** Write 1 to clear pending status

Details of NVIC & SCB registers

NVIC registers

- ***Interrupt Set Enable Register***

Address	Name	Type	Reset Value	Descriptions
0xE000E100	SETENA	R/W	0x00000000	Set enable for Interrupts 0 to 31; write 1 to set bit to 1, write 0 has no effect Bit[0] for Interrupt #0 (exception #16) Bit[1] for Interrupt #1 (exception #17) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current enable status

- ***Interrupt Clear Enable Register***

Address	Name	Type	Reset Value	Descriptions
0xE000E180	CLRENA	R/W	0x00000000	Clear enable for Interrupts 0 to 31;write 1 to clear bit to 0, write 0 has no effect Bit[0] for Interrupt #0 (exception #16) …

17

- **Interrupt Set Enable Register:**
- **Interrupt Clear Enable Register:**

Details of NVIC & SCB registers

SCB registers

- ***Interrupt control and state register***
  - Set and clear the pending status of system exceptions including SysTick, PendSV, and NMI.
  - Determine the currently executing exception/interrupt number by reading VECTACTIVE
- ***Vector table offset Register***

Bits	Name	Type	Reset Value	Description
31:30	**Reserved**	**-**	**-**	**Not implemented, tied to 0**
29	TBLBASE	R/W	0	Table base in Code (0) or RAM (1)
28:7	TBLOFF	R/W	0	Table offset value from Code region or RAM region

18

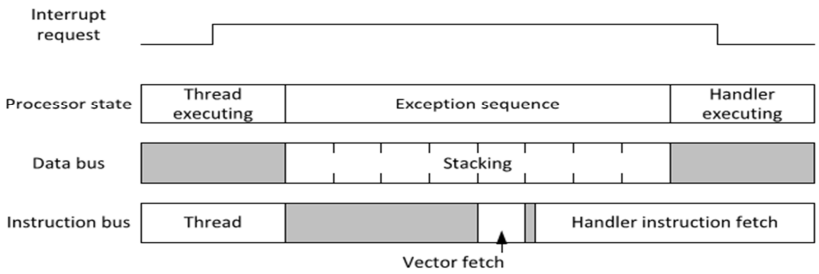
- Note: VTOR is optional on Cortex M0+/ not available on Cortex M0

Table of contents

Exception Sequence

An exception entrance sequence contains:

1. Stacking of a number of registers
2. Fetching the exception vector
3. Fetching the instructions for the exception handler to be executed.
4. Update NVIC and core registers



20

Note:

Stacking of a number of registers: If the processor was in Thread mode and was using the Process Stack Pointer (PSP), the stack area pointed to by the PSP will be used for this stacking. Otherwise the stack area pointed to by theMain Stack Pointer (MSP) will be used.

Fetching the exception vector: This can happen in parallel to the stacking operation to reduce latency.

Update of various NVIC registers and core registers: including: xPSR, MSP/PSP base on kind of used stack; PC, LR.
Link Register (LR) is updated with a special value called EXC_RE-TURN. This value is 32-bit, and the upper 27 bits are set to 1. Some of the lower 5 bits are used to hold status information about the exception sequence (e.g., which stack was used for stacking). This value is to be used in the exception return.

Exception Sequence

Exception handler execution:

1. The processor is in Handler mode

2. Nested Interrupt: If a higher-priority exception arrives, the currently executing handler will be suspended and pre-empted

21

Note:
The processor is in Handler mode when executing an exception handler. In Handler mode:
• The Main Stack Pointer (MSP) is used for stack operations
• The processor is executing in privileged access level

Nested Interrupt: If a higher-priority exception arrives during this stage, the new interrupt will be accepted, and the currently executing handler will be suspended and pre-empted by the higher-priority handler. If another exception with the same or lower priority arrives during this stage, the newly arrived exception will stay in the pending state and will be serviced when the current exception handler is completed

Exception Sequence

Exception return:

Instructions that can be Used for Triggering Exception Return

Return Instruction	Description
BX <*reg*>	If the EXC_RETURN value is still in the LR when the exception handler ends, we can use the "*BX LR*" instruction to perform the exception return.
POP {PC}, or POP {....., PC}	Very often the value of LR is pushed to the stack after entering the exception handler. We can use the POP instruction, either a single register POP or a POP operation with multiple registers including the PC, to put the EXC_RETURN value to the Program Counter (PC). This will cause the processor to perform the exception return.
Load (LDR) or Load multiple (LDM)	It is possible to produce an exception return using the LDR or LDM instructions with PC as the destination register.

22

-In ARM Cortex-M processors, the exception return mechanism is triggered using a special return address called EXC_RETURN. This value is generated at exception entrance and is stored in the Link Register (LR). When this value is written to the PC with one of the allowed exception return instructions, it triggers the exception return sequence.

Exception Sequence

Exception return:

- Unstacking: Restore Registers in the stack memory
- Update NVIC & core Registers
- Fetch the instructions of the previously interrupted program.

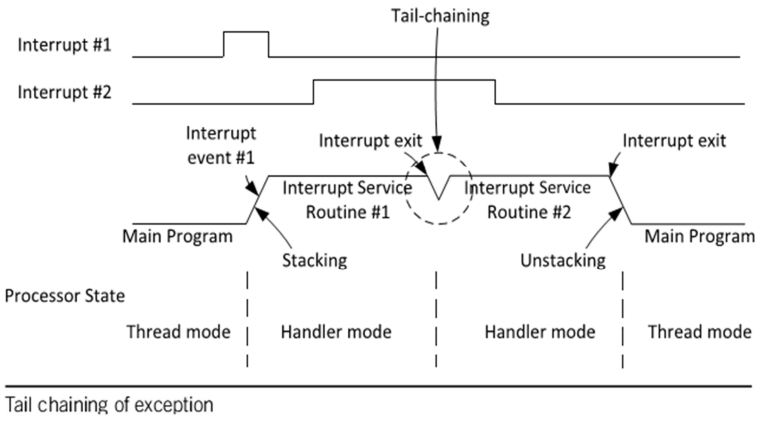
Table of contents

Exception handling optimization

- Tail chaining
- Late arrival
- Pop preemption
- Lazy stacking (Only support for Cortex M4 with FPT)

Exception handling optimization

Tail chaining:



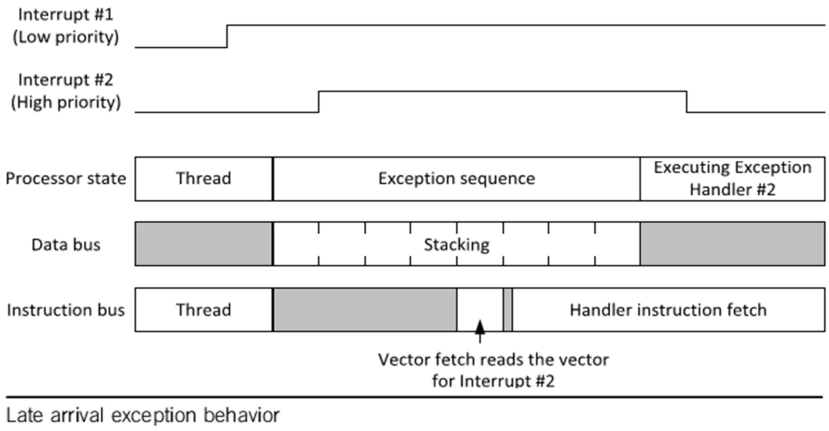
Tail chaining of exception

26

- When an exception takes place but the processor is handling another exception of the same or higher priority, the exception will enter the pending state. When the processor finishes executing the current exception handler, it can then proceed to process the pending exception/interrupt request. Instead of restoring the registers back from the stack (unstacking) and then pushing them on to the stack again (stacking), the processor skips the unstacking and stacking steps and enters the exception handler of the pended exception as soon as possible.

Exception handling optimization

Late arrival:



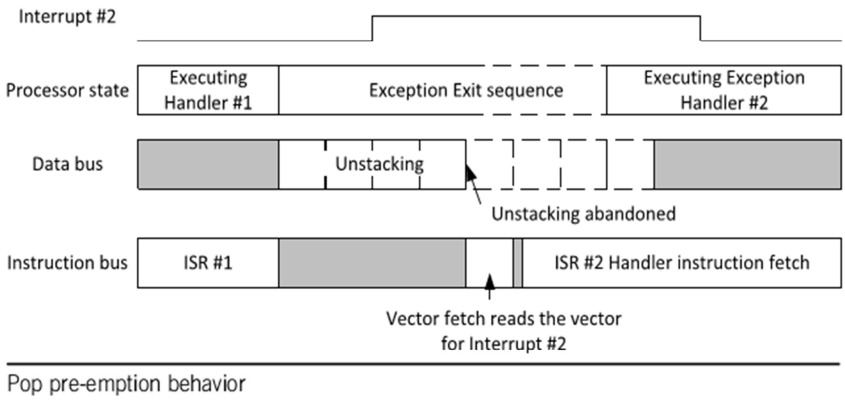
Late arrival exception behavior

27

- When an exception takes place, the processor accepts the exception request and starts the stacking operation. If during this stacking operation another exception of higher priority takes place, the higher priority late arrival exception will be serviced first.

Exception handling optimization

Pop preemption:



Pop pre-emption behavior

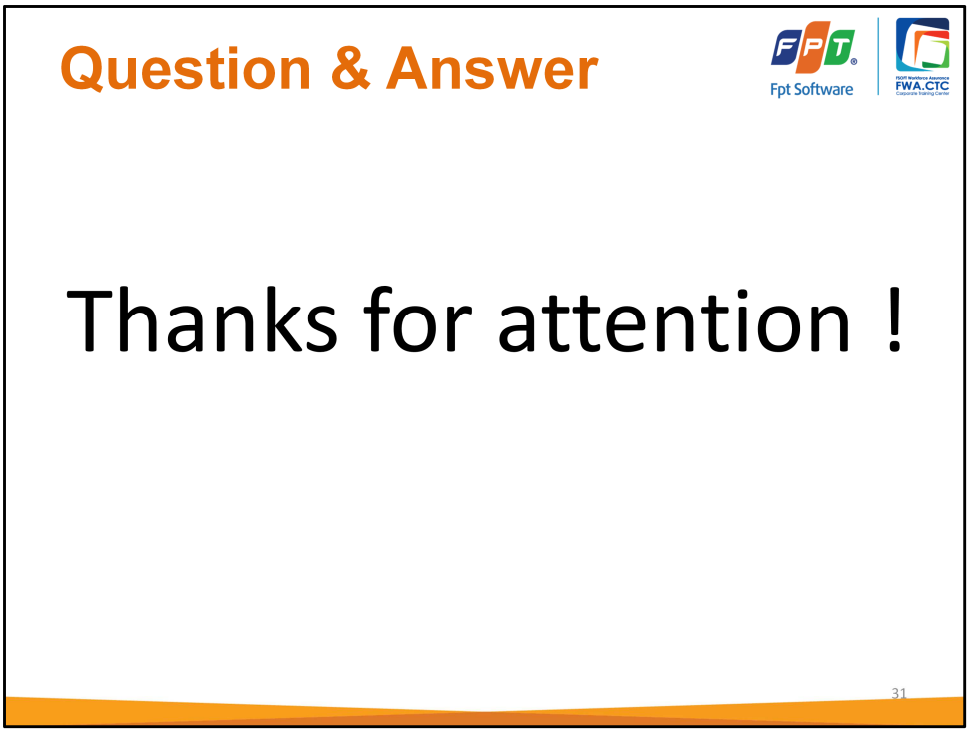
28

- If an exception request arrives during the unstacking process of another exception handler that has just finished, the unstacking operation would be abandoned and the vector fetch and instruction fetch for the next exception service begins. This optimization is called pop pre-emption.

Table of contents

Summary

- Cortex-M provides a Nested Vectored Interrupt Controller (NVIC) for interrupt handling.
- Exception entry sequence: Staking, Update registers, Fetch vector number & instruction of exception handler.
- Exception return sequence: Unstacking, Fetch & execute from the restored return address
- Exception handling optimization: Tail chaining, Late arrival, pop preemption.



- Trả lời các câu hỏi của học viên nếu có.

Copyright

32