

Exception and Interrupt Handling in ARM



Contents

- Introducing ARM
- Exceptions
- Interrupts
- Interrupt handling schemes
- Summary



Introducing ARM

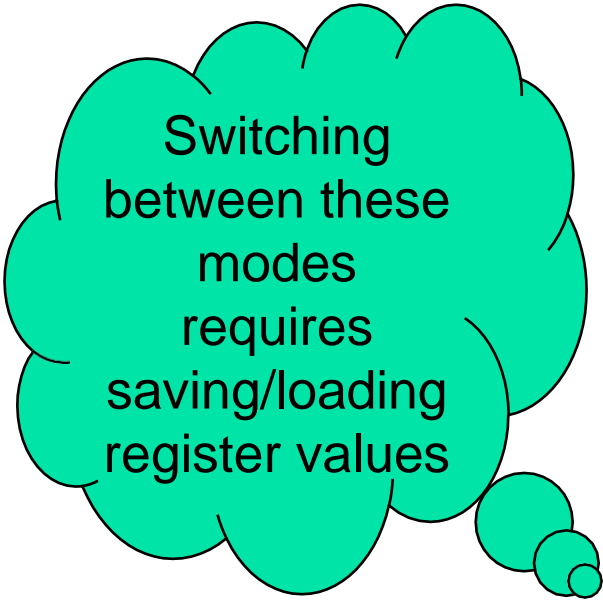
■ Modes of operation

- ARM processor has 7 modes of operation.
- Switching between modes can be done manually through modifying the mode bits in the CPSR register.
- Most application programs execute in user mode
- Non user modes (called privileged modes) are entered to serve interrupts or exceptions
- The system mode is special mode for accessing protected resources. It don't use registers used by exception handlers, so it can't be corrupted by any exception handler error!!!



Introducing ARM

■ Modes of operation



Switching
between these
modes
requires
saving/loading
register values

Processor Mode	Description
User (<i>usr</i>)	Normal program execution mode
FIQ (<i>fiq</i>)	Fast data processing mode
IRQ (<i>irq</i>)	For general purpose interrupts
Supervisor (<i>svc</i>)	A protected mode for the operating system
Abort (<i>abt</i>)	When data or instruction fetch is aborted
Undefined (<i>und</i>)	For undefined instructions
System (<i>sys</i>)	Privileged mode for OS Tasks



Introducing ARM

■ ARM register set

- ARM processor has 37 32-bit registers.
- 31 registers are general purpose registers.
- 6 registers are control registers
- Registers are named from R0 to R16 with some registers banked in different modes
- R13 is the stack pointer **SP** (Banked)
- R14 is subroutine link register **LR** (Banked)
- R15 is program counter **PC**
- R16 is current program status register **CPSR** (Banked)

Introducing ARM

■ ARM register set

ARM state general registers and program counter

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und
r14	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und
r15	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)

ARM state program status registers

CPSR	CPSR SPSR_fiq	CPSR SPSR_svc	CPSR SPSR_abt	CPSR SPSR_irq	CPSR SPSR_und
------	------------------	------------------	------------------	------------------	------------------

 = banked register

More banked registers, so context switching is faster



Contents

- Introducing ARM
- Exceptions
- Interrupts
- Interrupt handling schemes
- Summary



Exceptions

■ What is an exception?

An exception is any condition that needs to halt normal execution of the instructions

■ Examples

- Resetting ARM core
- Failure of fetching instructions
- HWI
- SWI



Exceptions

■ Exceptions and modes

Each exception causes the ARM core to enter a specific mode.

Exception	Mode	Purpose
Fast Interrupt Request	FIQ	Fast interrupt handling
Interrupt Request	IRQ	Normal interrupt handling
SWI and RESET	SVC	Protected mode for OS
Pre-fetch or data abort	ABT	Memory protection handling
Undefined Instruction	UND	SW emulation of HW coprocessors

Exceptions

■ Vector table

It is a table of addresses that the ARM core branches to when an exception is raised and there is always branching instructions that direct the core to the ISR.

At this place in memory, we find a branching instruction

`ldr pc, [pc, #_IRQ_handler_offset]`

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Exceptions

■ Exception priorities

decide which of the currently raised exceptions is more important

Decide if the exception handler itself can be interrupted during execution or not?

Both are caused by an instruction entering the execution stage of the ARM instruction pipeline

Exception	Priority	I bit	F bit
Reset	1	1	1
Data Abort	2	1	-
FIQ	3	1	1
IRQ	4	1	-
Prefetch abort	5	1	-
SWI	6	1	-
Undefined instruction	6	1	-



Exceptions

■ Link Register Offset

This register is used to return the **PC** to the appropriate place in the interrupted task since this is not always the old **PC** value. It is modified depending on the type of exception.

The **PC** has advanced beyond the instruction causing the exception. Upon exit of the prefetch abort exception handler, software must re-load the PC back one instruction from the **PC** saved at the time of the exception.

Exception	Returning Address
Reset	None
Data Abort	LR-8
FIQ, IRQ, prefetch Abort	LR-4
SWI, Undefined Instruction	LR



Exceptions

■ Entering exception handler

1. Save the address of the next instruction in the appropriate Link Register **LR**.
2. Copy **CPSR** to the **SPSR** of new mode.
3. Change the mode by modifying bits in **CPSR**.
4. Fetch next instruction from the vector table.

■ Leaving exception handler

1. Move the Link Register **LR** (minus an offset) to the **PC**.
2. Copy **SPSR** back to **CPSR**, this will automatically changes the mode back to the previous one.
3. Clear the interrupt disable flags (if they were set).



Contents

- Introducing ARM
- Exceptions
- **Interrupts**
- Interrupt handling schemes
- Summary



Interrupts

■ Assigning interrupts

It is up to the system designer who can decide which HW peripheral can produce which interrupt.

But system designers have adopted a standard design for assigning interrupts:

- SWI are used to call privileged OS routines.
- IRQ are assigned to general purpose interrupts like periodic timers.
- FIQ is reserved for one single interrupt source that requires fast response time.



Interrupts

■ Interrupt latency

It is the interval of time from an external interrupt signal being raised to the first fetch of an instruction of the ISR of the raised interrupt signal.

System architects try to achieve two main goals:

- To handle multiple interrupts simultaneously.
- To minimize the interrupt latency.

And this can be done by 2 methods:

- allow nested interrupt handling
- give priorities to different interrupt sources



Interrupts

■ Enabling and disabling Interrupt

This is done by modifying the **CPSR**, this is done using only 3 ARM instruction:

MRS To read *CPSR*

MSR To store in *CPSR*

BIC Bit clear instruction

ORR OR instruction

Enabling an IRQ/FIQ
Interrupt:

```
MRS  r1, cpsr
BIC   r1, r1,
MSR   #0x80/0x40
```

Disabling an IRQ/FIQ
Interrupt:

```
MRS  r1, cpsr
ORR   r1, r1, #0x80/0x40
MSR   cpsr_c, r1
```



Interrupts

■ Interrupt stack

Stacks are needed extensively for context switching between different modes when interrupts are raised.

The design of the exception stack depends on two factors:

- OS Requirements.
- Target hardware.

A good stack design tries to avoid stack overflow because it cause instability in embedded systems.

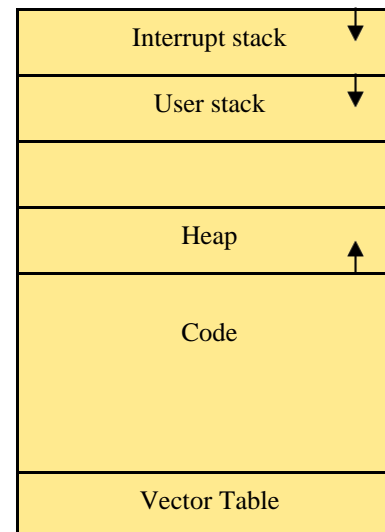
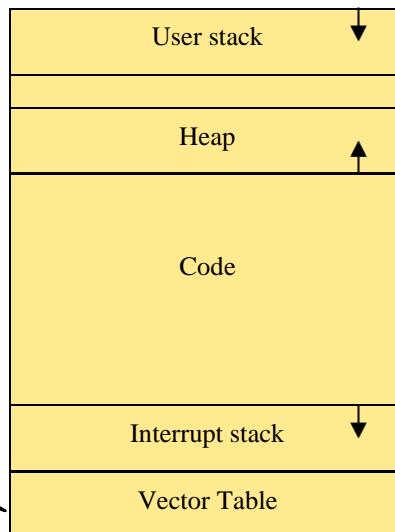
Interrupts

■ Interrupt stack

Two design decisions need to be made for the stacks:

- The location
- The size

Traditional
memory
layout



The benefit of
this layout is
that the vector
table remains
untouched if a
stack overflow
occured!!



Contents

- Introducing ARM
- Exceptions
- Interrupts
- Interrupt handling schemes
- Summary

Interrupt handling schemes

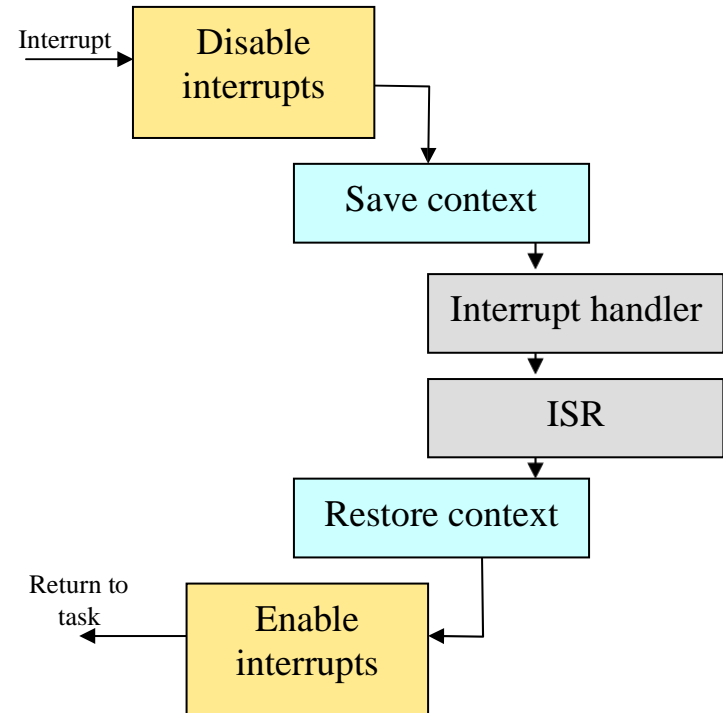
■ Non-nested interrupt handling scheme

- This is the simplest interrupt handler.

- Interrupts are disabled until control is returned back to the interrupted task.

- One interrupt can be served at a time.

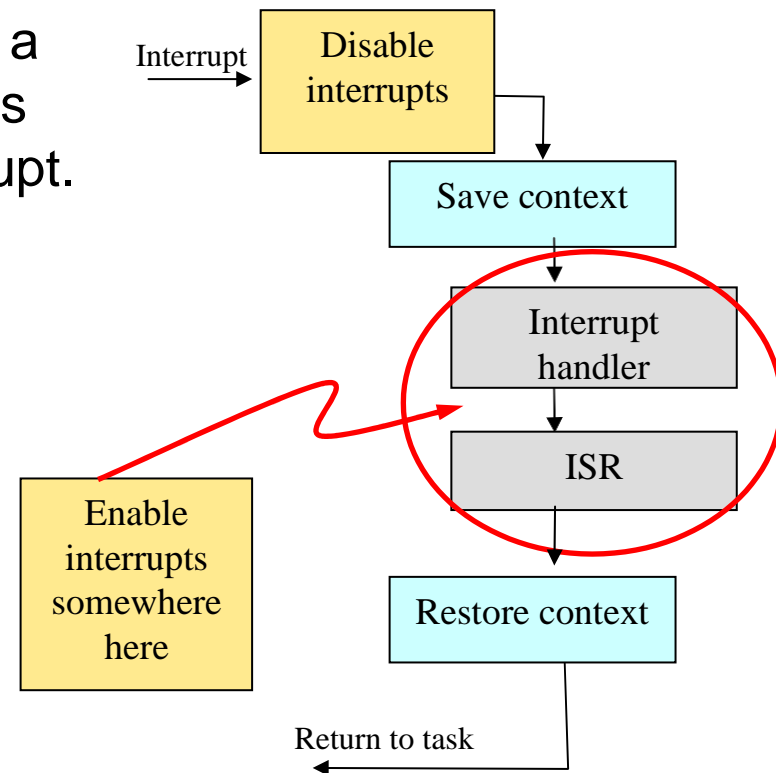
- Not suitable for complex embedded systems.



Interrupt handling schemes

■ Nested interrupt handling scheme(1)

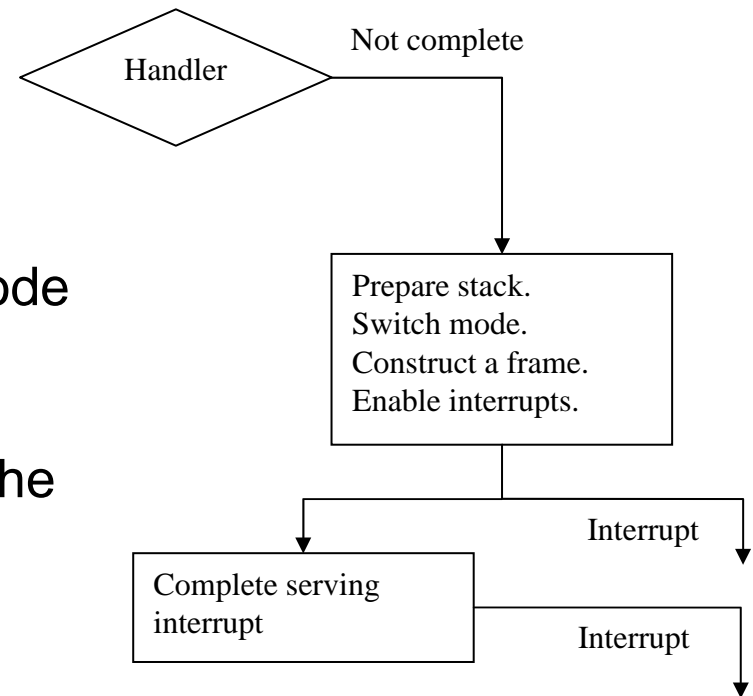
- Handling more than one interrupt at a time is possible by enabling interrupts before fully serving the current interrupt.
- Latency is improved.
- System is more complex.
- No difference between interrupts by priorities, so normal interrupts can block critical interrupts.



Interrupt handling schemes

■ Nested interrupt handling scheme(2)

- The handler tests a flag that is updated by the ISR
- Re enabling interrupts requires switching out of current interrupt mode to either SVC or system mode.
- Context switch involves emptying the IRQ stack into reserved blocks of memory on SVC stack called stack frames.





Interrupt handling schemes

- Prioritized simple interrupt handling
 - associate a priority level with a particular interrupt source.
 - Handling prioritization can be done by means of software or hardware.
 - When an interrupt signal is raised, a fixed amount of comparisons is done.
 - So the interrupt latency is deterministic.
 - But this could be considered a disadvantage!!



Interrupt handling schemes

■ Other schemes

There are some other schemes, which are actually modifications to the previous schemes as follows:

- “Re-entrant interrupt handler”: re-enable interrupts earlier and support priorities, so the latency is reduced.
- “Prioritized standard interrupt handler”: arranges priorities in a special way to reduce the time needed to decide on which interrupt will be handled.
- “Prioritized grouped interrupt handler”: groups some interrupts into subset which has a priority level, this is good for large amount of interrupt sources.



Contents

- Introducing ARM
- Exceptions
- Interrupts
- Interrupt handling schemes
- **Summary**



Summary

- Availability of different modes of operation in ARM helps in exception handling in a structured way.
- Context switching is one of the main issues affecting interrupt latency, and this is resolved in ARM FIQ mode by increasing number of banked registers.
- We can't decide on one interrupt handling scheme to be used as a standard in all systems, it depends on the nature of the system:
 - What type of interrupts are there?
 - How many interrupts are there?

Exception Handler:

- Typically, an **application** runs in **User mode**, but handling **exceptions** requires a **privileged mode**
- An exception changes the processor mode, and this in turn means that each exception handler has access to a certain subset of the banked out registers:
 - Its own Stack Pointer (**SP**)
 - Its own Link Register (**LR**)
 - Its own Saved Program Status Register (**SPSR**)
 - Five additional general purpose registers (**FIQ** only)
- Each exception handler must ensure that other registers are restored, if used by Handlers, to their original contents on exit
- It can be done by saving the contents of any registers that the handler uses onto its stack and restore them before returning

When an exception is generated, the processor performs the following actions:

- Copies the **CPSR** into the appropriate **SPSR**. This saves the current mode, interrupt mask, and condition flags
- Switches state (**Thumb** to **ARM**) automatically if the current state is Thumb to match the ARM instruction set used in the exception vector table
- Changes the appropriate **CPSR** mode bits to:
 - Change to the appropriate mode, and map in the appropriate banked out registers for that mode
 - Disable interrupts. **IROs** are disabled when any exception occurs. **FIQ** is disabled when an **FIQ** occurs and on reset
- Sets the appropriate **LR** with the **return address**

Exceptions: Their Own Stack

User and system

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 sp
r14 lr
r15 pc

Fast
interrupt
request

r8_fiq
r9_fiq
r10_fiq
r11_fiq
r12_fiq
r13_fiq
r14_fiq

Interrupt
request

r15_irq
r14_irq

Supervisor

r13_svc
r14_svc

Undefined

r13_undef
r14_undef

Abort

r13_abt
r14_abt

cpsr
-

spsr_fiq

spsr_irq

spsr_svc

spsr_undef

spsr_abt

User Code
Data Stack

Handlers

IRQ Stack

FIQ Stack

Reset Init
Code

FIQ ISR

FIQ Vector

:

SVC Vector

Undef Vector

Reset Vector

Setting up Stacks for Exception Modes

Setting up stack for **IRQ** and **FIQ** modes

.EQU **IRQ_Stack_Size**, 0x8000

.EQU **FIQ_Stack_Size**, 0x4000

.EQU **IRQ_Mode**, 0x12

.EQU **FIQ_Mode**, 0x11

.EQU **SVC_Mode**, 0x13

.EQU **SYS_Mode**, 0x1F

.EQU **MODE_MASK**, 0x1F ; lower 5 bits of **CPSR**

.data

IRQ_NewStack:

.skip **IRQ_Stack_Size**

FIQ_NewStack:

.skip **FIQ_Stack_Size**

; **Currently in Sys mode**

; **Init IRQ stack**

MRS r0, **CPSR**

BIC r0, r0, #**MODE_MASK**

ORR r0, r0, #**IRQ_Mode**

MSR **CPSR_C**, r0

LDR **r13**, =**IRQ_NewStack**

; **init FIQ stack**

MRS r0, **CPSR**

BIC r0, r0, #**MODE_MASK**

ORR r0, r0, #**FIQ_Mode**

MSR **CPSR_C**, r0

LDR **r13**, =**FIQ_NewStack**

- **Interrupt Handling Schemes**
 - **Non-nested Interrupt Handler**
 - **Nested Interrupt Handler**
 - **Prioritized Simple Interrupt Handler**
 - **Other Schemes**
 - **Examples of Interrupt Handlers**
 - **Non-nested Handlers**
 - **Nested Handlers**
-

Handling of Interrupts:

- **Interrupt Latency**

- It is the time interval from an external interrupt signal being raised to the first fetch of an instruction of the **ISR** of the raised interrupt signal
- Based on the **criticality** of source of the interrupt, the **latency** needs to be **minimized**

- There are **two goals** to be achieved by **system designers**

- Handling of multiple interrupts simultaneously (with the available two IRQ/FIQ inputs)
- To **minimize** the **Interrupt latency**

- This can be done in **two ways**

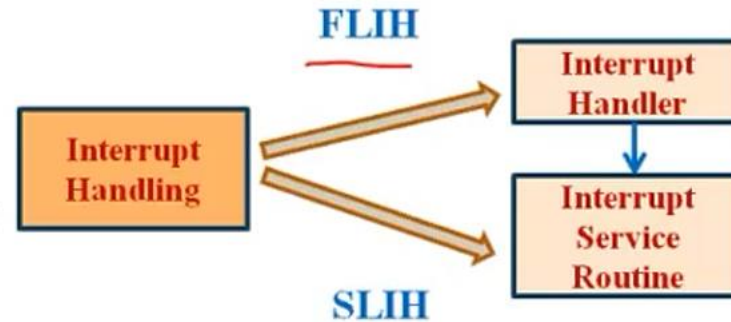
- Allow **nested Interrupt** handling
- Give **Priorities** to different sources

Assigning Interrupts:

- **System Designers** can decide which **HW** peripheral need to be connected to which interrupt (IRQ/FIQ)
 - Based on interrupt processing response expected from the Processor
- Standard assignment of various Interrupts are given below.
- **SWI** is used for Privileged **OS** routines
- **IRQs** are used to connect general purpose interrupts like serial ports, timers, etc. (normally through Interrupt controller)
- **FIQs** are reserved for one single interrupt source that requires fast response time
 - Single source is preferred to avoid delay in recognizing the source of the interrupt on receiving it in the FIQ handler

Interrupts handlers are **divided**
into **two parts**

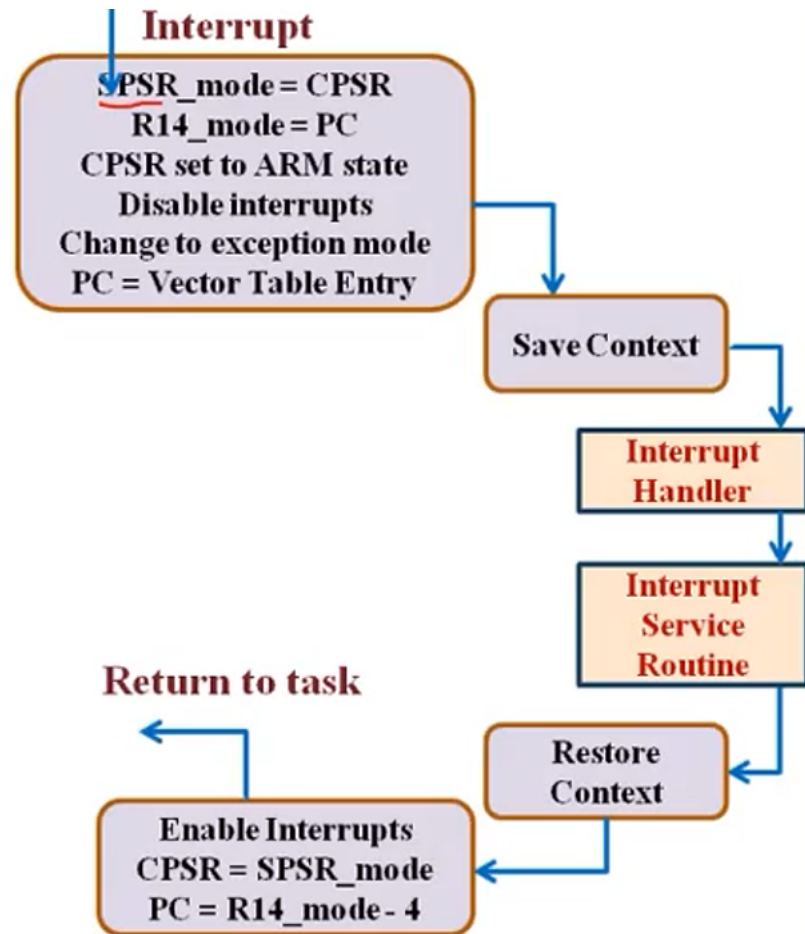
- **First Level** Interrupt Handler
 - Hard/Fast Int Handlers
- **Second Level** Interrupt Handler
 - Soft/Slow Int Handlers or
 - Deferred Procedure call



No	First Level Interrupt Handler	Second Level Interrupt Handler
1.	Implements minimum platform specific interrupt handling similar to int routines	Implements long interrupt processing tasks similar to a process
2.	Intent is to quickly service the interrupt	Deferred job is handled here
3.	Record platform-specific critical information which is only available at the time of the interrupt	Use the information provided by FLIH and perform further processing
4.	Schedule the execution of a SLIH for further long-lived interrupt handling	Run by the OS based on the available processing time (background task)
5.	If it is reentrant, the same interrupt is re-enabled or same/higher priority interrupts	Handles only the processing of information provided by FLIH
6.	Programmers move maximum processing to SLIH and minimize the execution time of FLIH	

Non-Nested Interrupt Handler:

- This is the **simplest** Interrupt Handler
- On **entering** the handler **interrupts are disabled**
 - Note that **FIQ** is not disabled if **IRQ** is being serviced





User Mode Stack Initialization

- It is common for the **user mode stack** to be the **last** to be set up because when the processor is in user mode there is no direct method to modify the **CPSR**
- An alternative is to **force** the processor into **system mode** to set up the user mode stack since both modes **share the same registers**
- Using **separate stacks** for **each mode** rather than using a single stack has one main advantage
- Errant tasks can be debugged and isolated from the rest of the system
- For safety reasons, it is always required to **enable IRQ and FIQ only after** their own stack pointers (**R13**) are **initialized**



Summary

- Availability of different modes of operation in ARM helps in exception handling in a structured way.
- Context switching is one of the main issues affecting interrupt latency, and this is resolved in ARM FIQ mode by increasing number of banked registers.
- We can't decide on one interrupt handling scheme to be used as a standard in all systems, it depends on the nature of the system:
 - What type of interrupts are there?
How many interrupts are there?