ARM: an overview from the Operating System perspective

César Huegel Richa

huegel@lisha.ufsc.br

Software/Hardware Integration Laboratory (LISHA)
Federal University of Santa Catarina (UFSC)

Florianópolis, Santa Catarina, Brazil
April, 2018

Summary

- Introduction
- ARMv7 architecture
- Data Sizes and Instruction Sets
- Programmer's Model
- Exceptions and Interruptions
- Systick

History of ARM

- **A**corn **R**ISC **M**achine (ARM1): a new, powerful, CPU design for the replacement of the 8-bit 6502 in Acorn Computers (Cambridge, 1985).

History of ARM

- **A**corn **R**ISC **M**achine (ARM1): a new, powerful, CPU design for the replacement of the 8-bit 6502 in Acorn Computers (Cambridge, 1985).
- 1990 spin-off: ARM (**A**dvanced **R**ISC Machines)

History of ARM

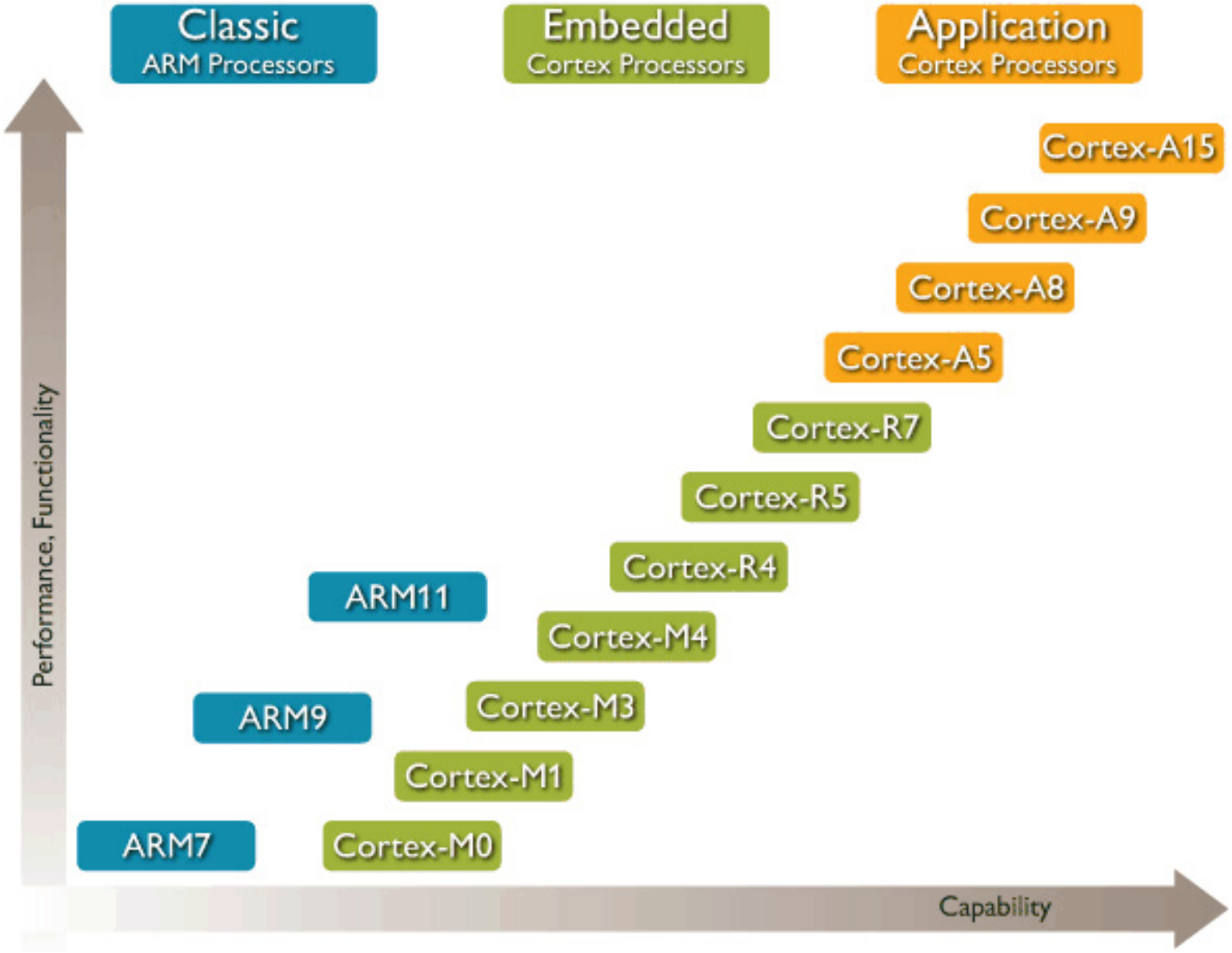
- **A**corn **R**ISC **M**achine (ARM1): a new, powerful, CPU design for the replacement of the 8-bit 6502 in Acorn Computers (Cambridge, 1985).
- 1990 spin-off: ARM (**A**dvanced **R**ISC M**M**achines)
- Also develops technologies to assist the ARM architecture designing
  - Software tools, development boards, debug hw, ...

History of ARM

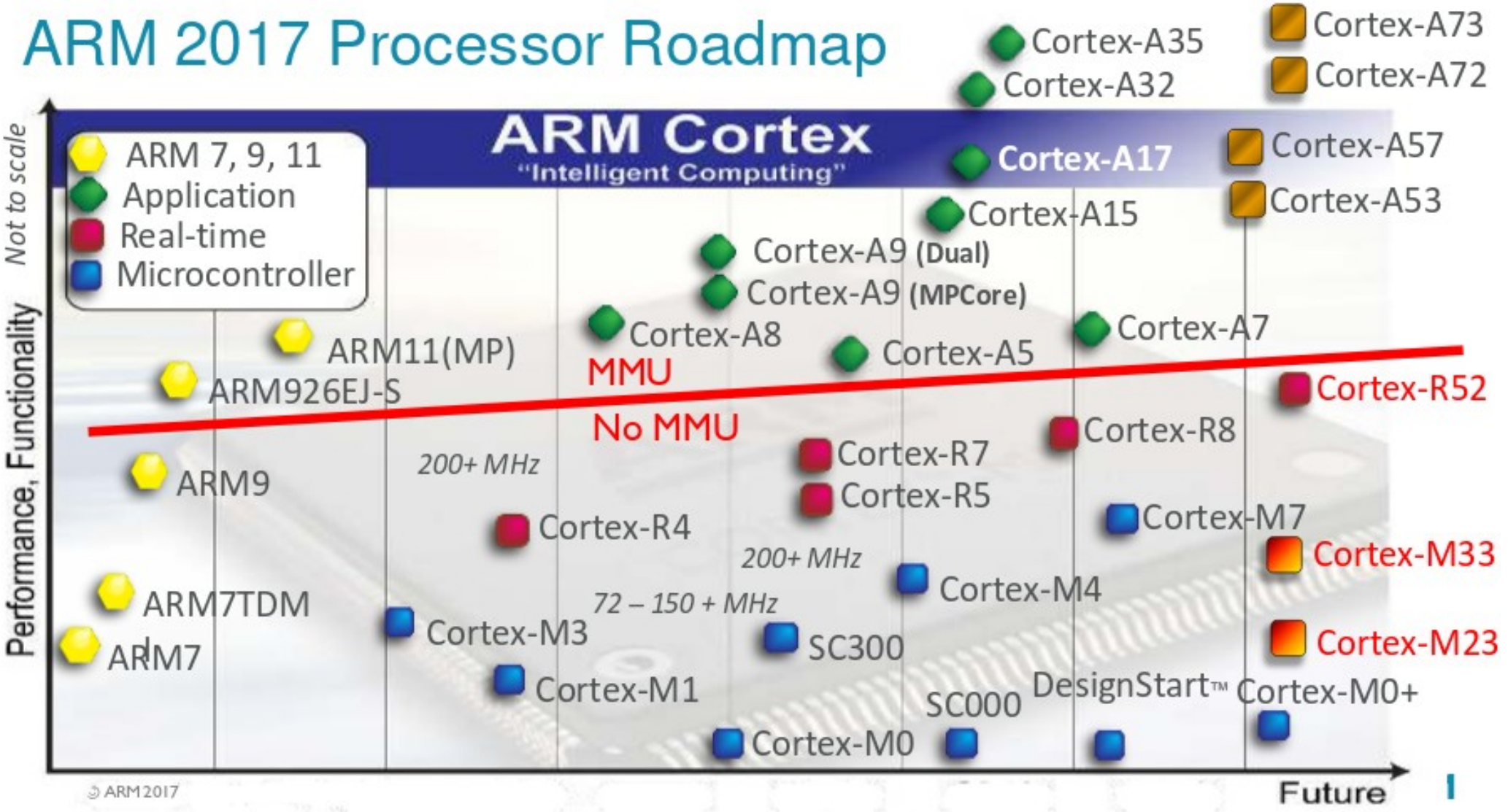
- **A**corn **R**ISC **M**achine (ARM1): a new, powerful, CPU design for the replacement of the 8-bit 6502 in Acorn Computers (Cambridge, 1985).
- 1990 spin-off: ARM (**A**dvanced **R**ISC M**M**achines)
- Also develops technologies to assist the ARM architecture designing
  - Software tools, development boards, debug hw, ...
- The ARM Business Model

  ARM does not manufacture processors.

ARM Processor Roadmap

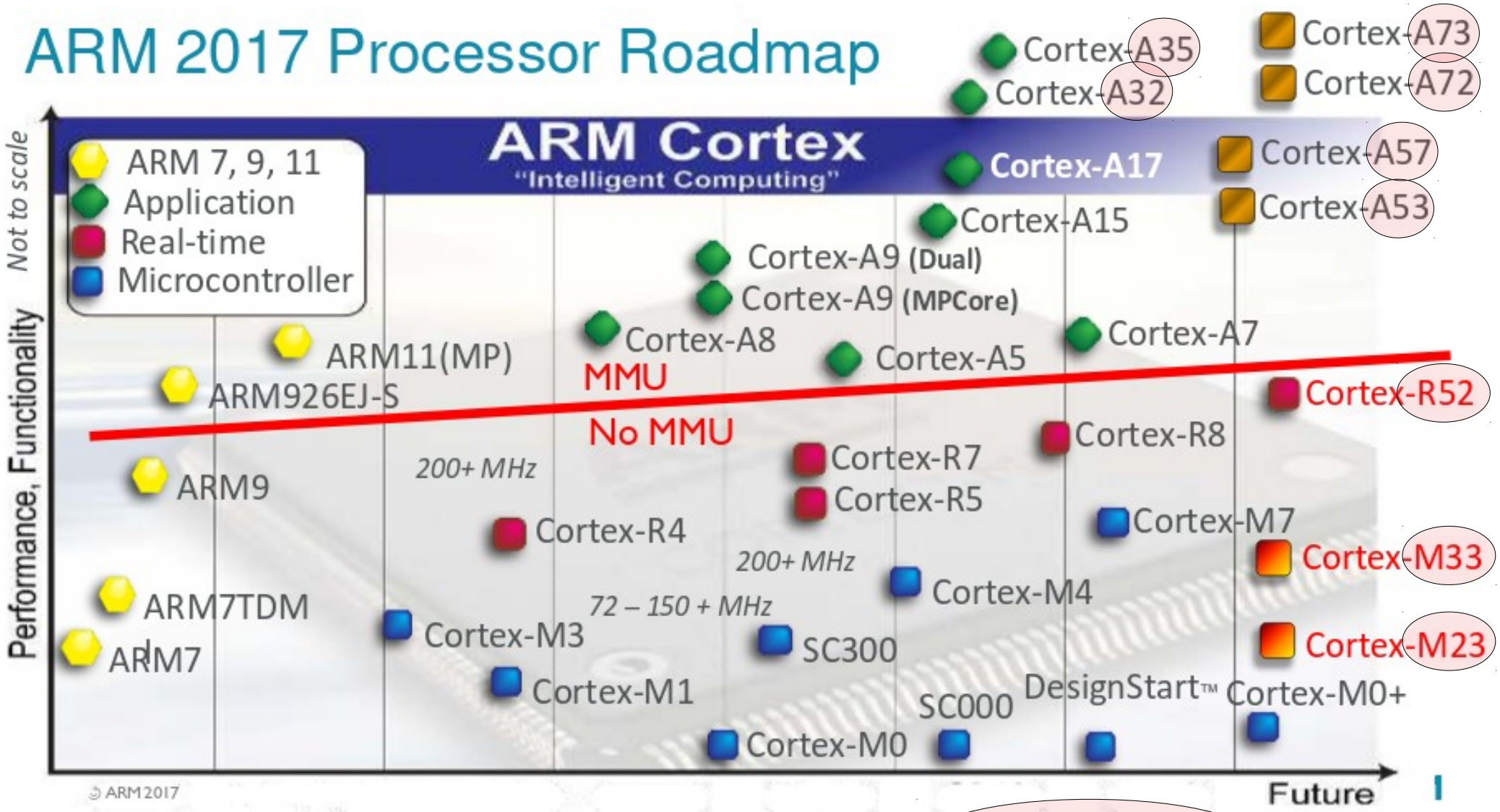


ARM Processor Roadmap



From: http://wiki.csie.ncku.edu.tw/embedded/arm-roadmap-2017q1.pdf

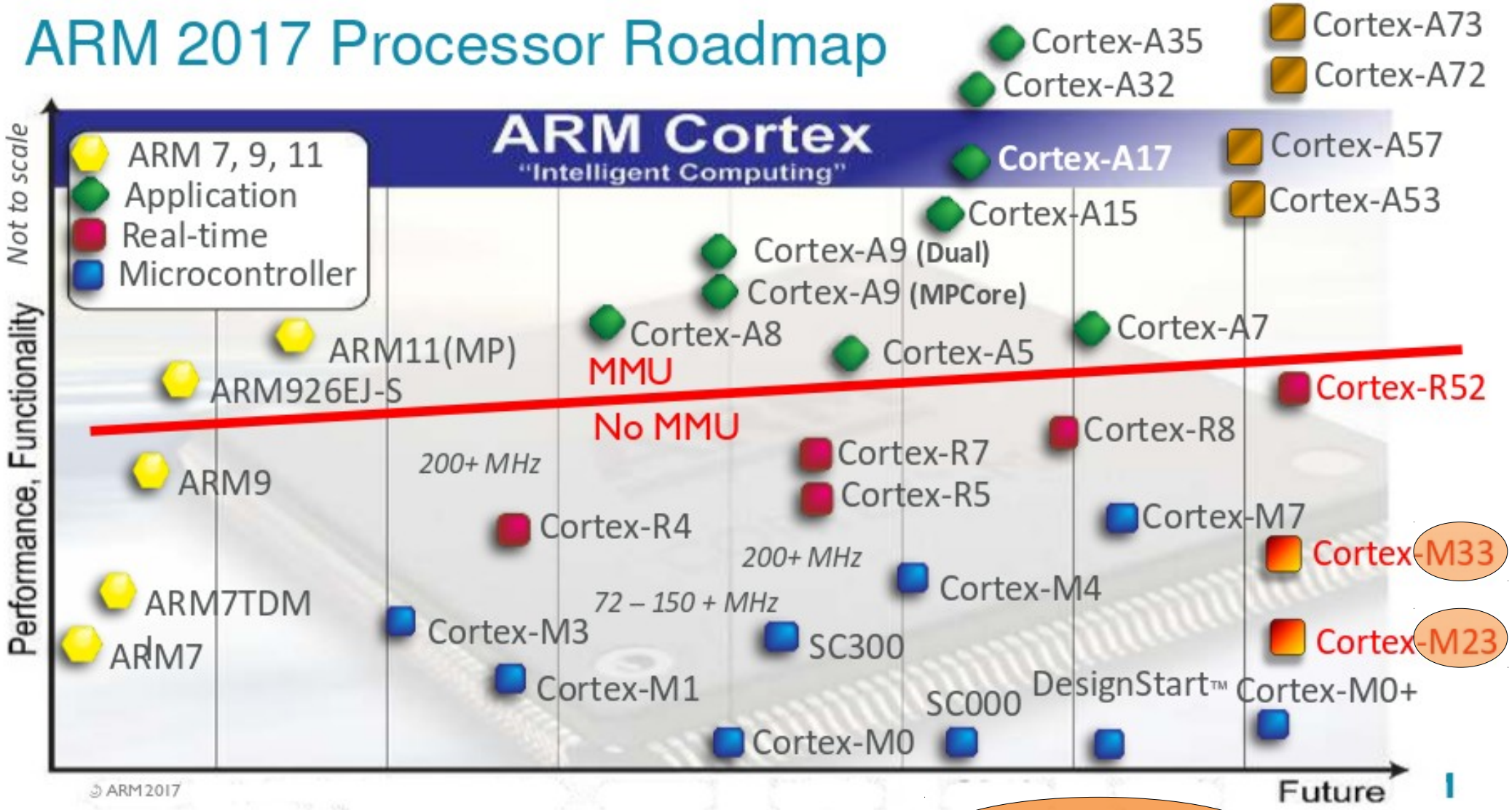
ARM Processor Roadmap



From: http://wiki.csie.ncku.edu.tw/embedded/arm-roadmap-2017q1.pdf

ARMv8-A/R/M

ARM Processor Roadmap



From: http://wiki.csie.ncku.edu.tw/embedded/arm-roadmap-2017q1.pdf

TrustZone

Families, architectures, versions, and cores? What?

ARM Family	ARM Architecture	ARM Core
….	….	….
ARM11	ARMv6, …	ARM1136J(F)-S, …
Cortex-M	ARMv6-M	Cortex-M0, M0+, M1
	ARMv7-M	Cortex-M3
	ARMv7E-M	Cortex-M4, M7
Cortex-R	ARMv7-R	Cortex-R4, R5, R7, R8
Cortex-A (32-bit)	ARMv7-A	Cortex-A5, A7, A8, A9, A12, A15, A17
	ARMv8-A	Cortex-A32
Cortex-A (64-bit)	ARMv8-A	Cortex-A35, A53, A57, A72, A73
	ARMv8.2-A	Cortex-A55, A75

Architecture ARMv7 profiles

- Application profile (ARMv7-A)
  - Highest performance (optimized for rich operating systems)
  - e.g. Cortex-A5,A7,A8,A9,A12,A15, ….

Architecture ARMv7 profiles

- Application profile (ARMv7-A)
  - Highest performance (optimized for rich operating systems)
  - e.g. Cortex-A5,A7,A8,A9,A12,A15, ….
- Real-time profile (ARMv7-R)
  - Fast response (optimized for high-performance, hard real-time applications)
  - e.g. Cortex-R7

Architecture ARMv7 profiles

- Application profile (ARMv7-A)
  - Highest performance (optimized for rich operating systems)
  - e.g. Cortex-A5,A7,A8,A9,A12,A15, ….
- Real-time profile (ARMv7-R)
  - Fast response (optimized for high-performance, hard real-time applications)
  - e.g. Cortex-R7
- Microcontroller profile (ARMv7-M)
  - Smallest/lowest power (optimized for discrete processing and microcontroller)
  - e.g. Cortex-M3

Data Sizes and Instruction Sets

- ARM is a 32-bit load/store RISC architecture
  - No direct manipulation of memory contents

Data Sizes and Instruction Sets

- ARM is a 32-bit load/store RISC architecture
  No direct manipulation of memory contents

- Supported data types
  - Byte (8 bits)
  - Halfword (16 bits)
  - Word (32 bits)

Data Sizes and Instruction Sets

- ARM is a 32-bit load/store RISC architecture
  No direct manipulation of memory contents

- Supported data types
  - Byte (8 bits)
  - Halfword (16 bits)
  - Word (32 bits)

- ARM cores implement two basic instruction sets
  - ARM: all 32 bits long
  - Thumb: mix of 16 and 32 bits
    - Thumb2: Thumb instructions + many extra 32 and 16 bit instructions

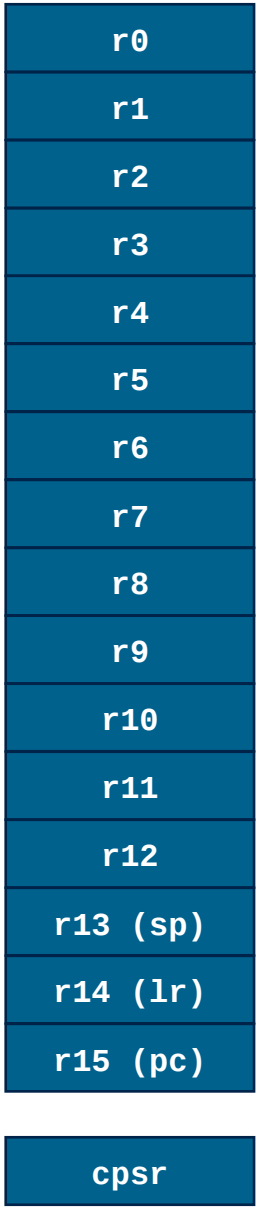
Note: ARMv7-M only supports Thumb/Thumb2 instructions

ARMv7-AR's Register Set

- ARM has a total of 37 registers
  - All of these are 32-bit long
    - 18 general purpose registers {r0, …., r12}
    - 6 dedicated stack pointer (sp) {r13}
    - 6 dedicated link register (lr) {r14}
    - 1 dedicated program counter (pc) {r15}
    - 1 dedicated current program status register {cpsr}
    - 5 dedicated saved program status register {spsr}

ARMv7-AR's Register Set

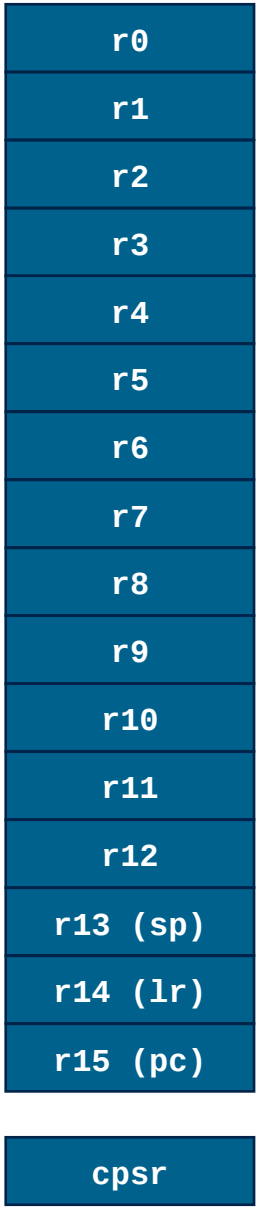
- ARM has a total of 37 registers
  - All of these are 32-bit long
    - 18 general purpose registers {r0, …., r12}
    - 6 dedicated stack pointer (sp) {r13}
    - 6 dedicated link register (lr) {r14}
    - 1 dedicated program counter (pc) {r15}
    - 1 dedicated current program status register {cpsr}
    - 5 dedicated saved program status register {spsr}



ARMv7-AR's Register Set

- ARM has a total of 37 registers
  - All of these are 32-bit long
    - 18 general purpose registers {r0, …., r12}
    - 6 dedicated stack pointer (sp) {r13}
    - 6 dedicated link register (lr) {r14}
    - 1 dedicated program counter (pc) {r15}
    - 1 dedicated current program status register {cpsr}
    - 5 dedicated saved program status register {spsr}

- **Where are the other registers?**



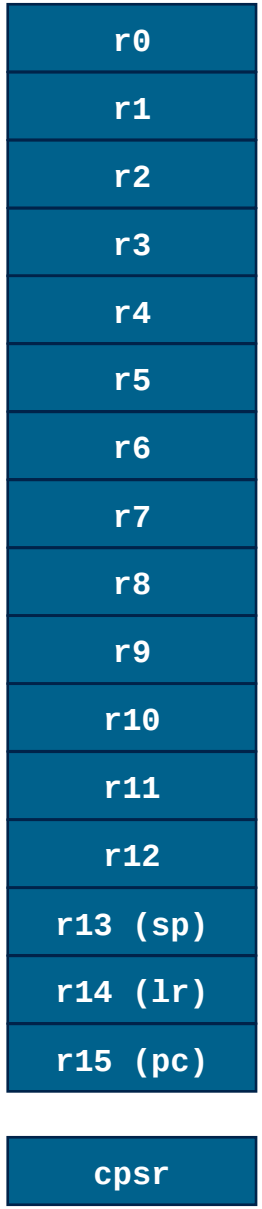
ARMv7-AR's Register Set

- ARM has a total of 37 registers
  - All of these are 32-bit long
    - 18 general purpose registers {r0, …., r12}
    - 6 dedicated stack pointer (sp) {r13}
    - 6 dedicated link register (lr) {r14}
    - 1 dedicated program counter (pc) {r15}
    - 1 dedicated current program status register {cpsr}
    - 5 dedicated saved program status register {spsr}

- **Where are the other registers?**

- **Banked register**

  A register that has multiple instances, with the instance that is in use depending on the processor mode, security state, or other process state.



Processor Modes

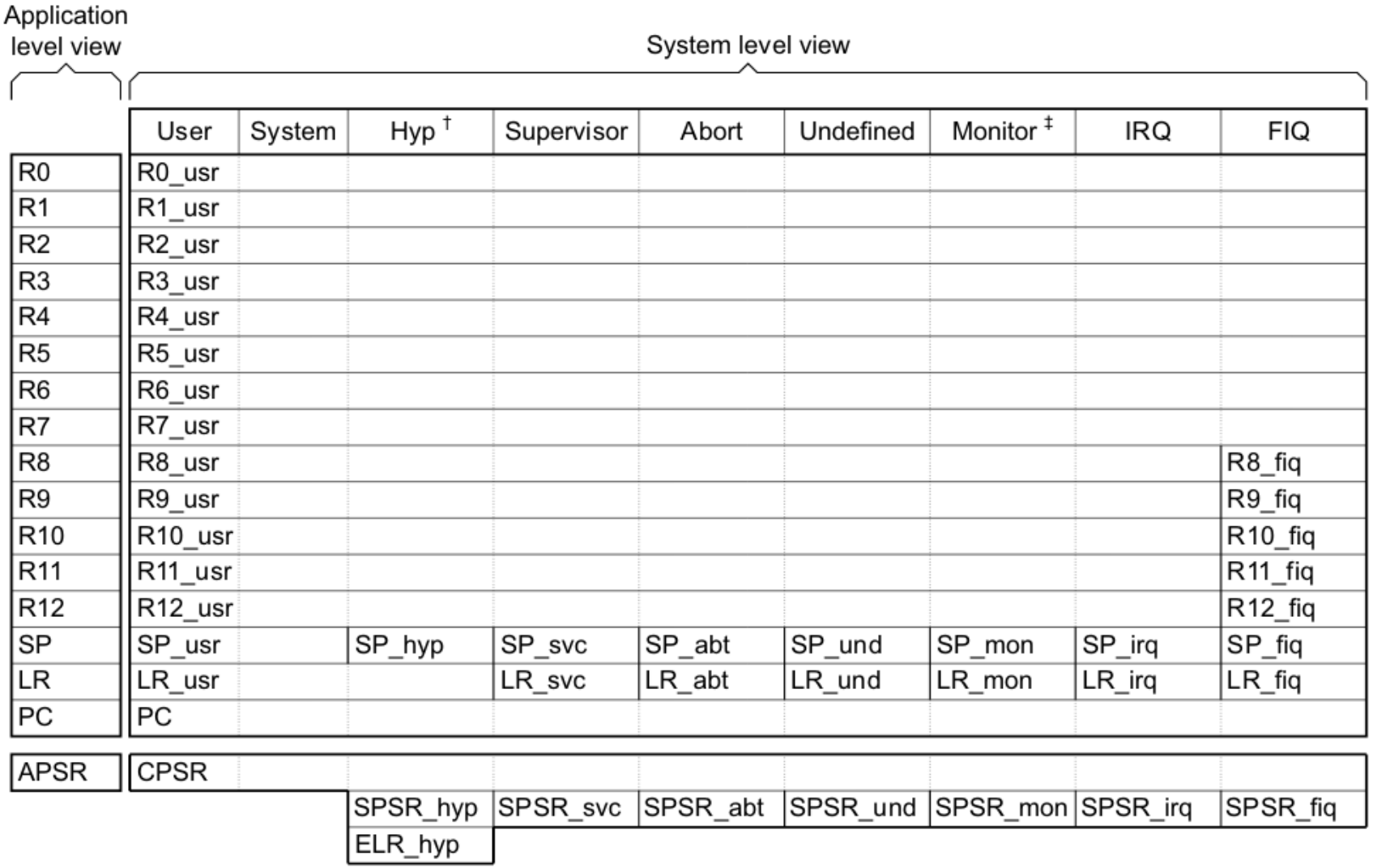
- ARMv7-AR

Exception modes

Mode	Description	
Supervisor (SVC)	Entered on reset and when a Supervisor call instruction (SVC) is executed	**Privileged modes**
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a normal priority interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	
User	Mode under which most Applications / OS tasks run	**Unprivileged mode**

Register Organization

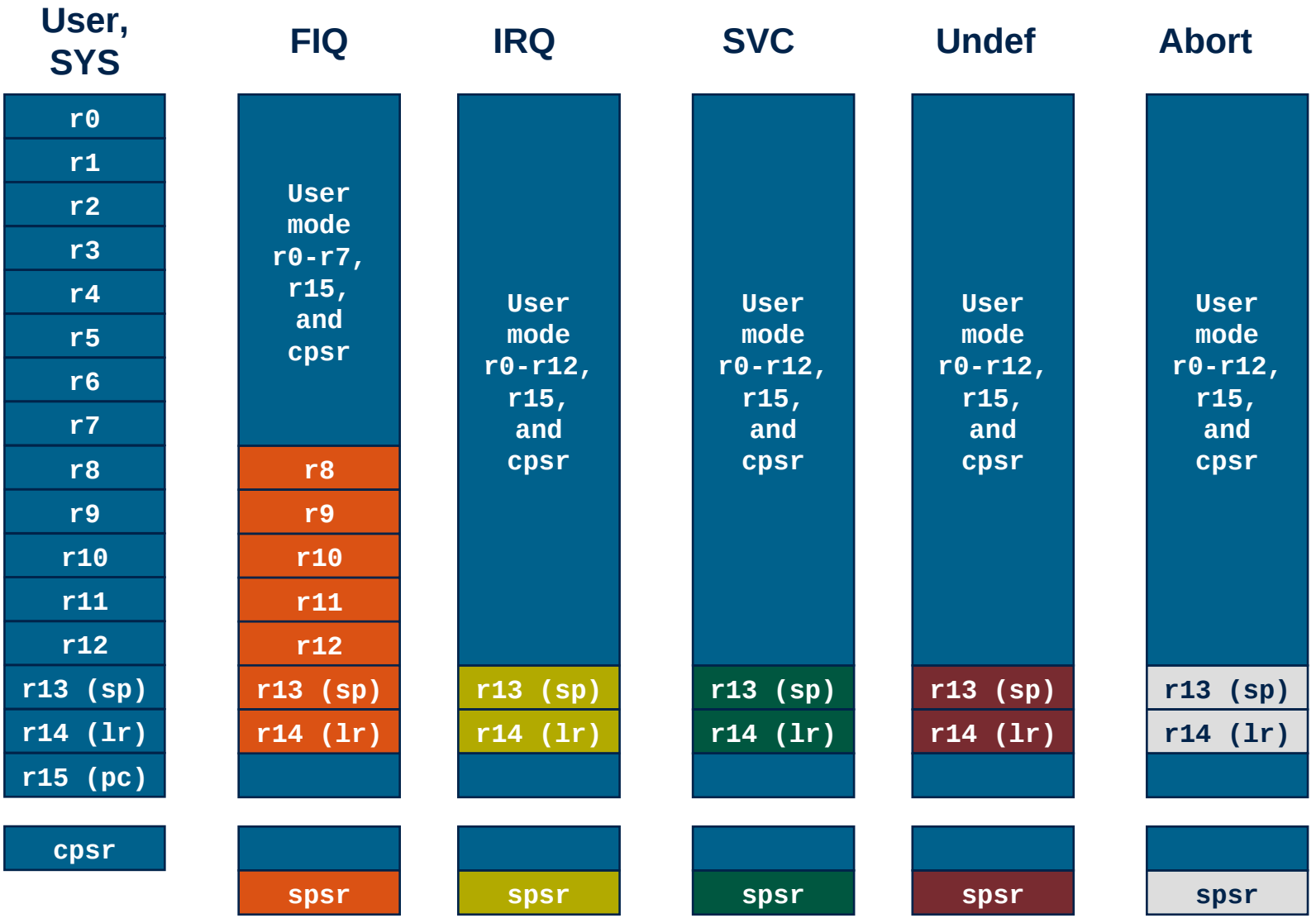
- ARMv7-AR



	User	System	Hyp †	Supervisor	Abort	Undefined	Monitor ‡	IRQ	FIQ
R0	R0_usr								
R1	R1_usr								
R2	R2_usr								
R3	R3_usr								
R4	R4_usr								
R5	R5_usr								
R6	R6_usr								
R7	R7_usr								
R8	R8_usr								R8_fiq
R9	R9_usr								R9_fiq
R10	R10_usr								R10_fiq
R11	R11_usr								R11_fiq
R12	R12_usr								R12_fiq
SP	SP_usr		SP_hyp	SP_svc	SP_abt	SP_und	SP_mon	SP_irq	SP_fiq
LR	LR_usr			LR_svc	LR_abt	LR_und	LR_mon	LR_irq	LR_fiq
PC	PC								
APSR	CPSR								
			SPSR_hyp	SPSR_svc	SPSR_abt	SPSR_und	SPSR_mon	SPSR_irq	SPSR_fiq
			ELR_hyp						

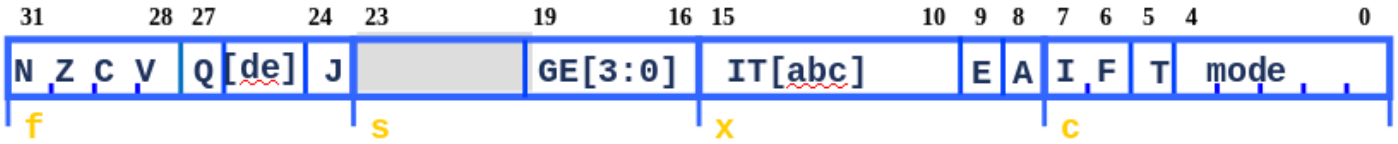
Register Organization

- ARMv7-AR



Program Status Registers

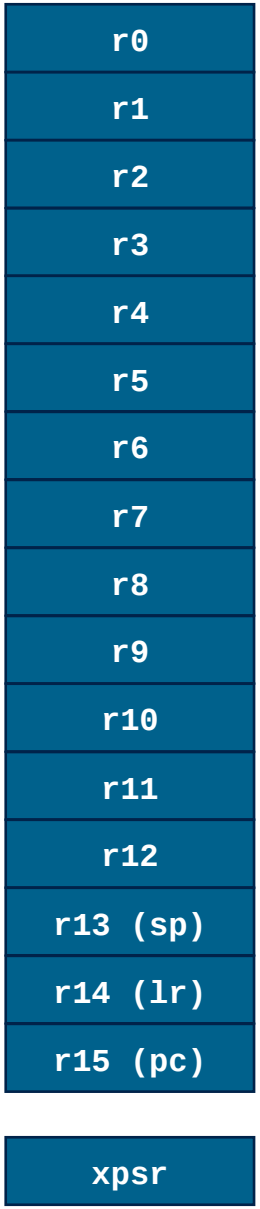
- ARMv7-AR



- **Condition code flags**
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- **Sticky Overflow flag - Q flag**
  - Indicates if saturation has occurred
- **SIMD Condition code bits – GE[3:0]**
  - Used by some SIMD instructions
- **IF THEN status bits – IT[abcde]**
  - Controls conditional execution of Thumb instructions
- **T bit**
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
- **J bit**
  - J = 1: Processor in Jazelle state
- **Mode bits**
  - Specify the processor mode
- **Interrupt Disable bits**
  - I = 1: Disables IRQ
  - F = 1: Disables FIQ
- **E bit**
  - E = 0: Data load/store is little endian
  - E = 1: Data load/store is bigendian
- **A bit**
  - A = 1: Disable imprecise data aborts

ARMv7-M Register Set

- ARM has a total of 18 registers
  - All of these are 32-bit long
    - 13 general purpose registers {r0, …., r12}
    - 2 dedicated stack pointer (sp) {r13}
      - 2 banked ver. (sp_main or MSP, sp_process or PSP)
    - 1 dedicated link register (lr) {r14}
    - 1 dedicated program counter (pc) {r15}
    - 1 dedicated program status register {xpsr}
      - not explicitly accessible
      - saved to the stack on an exception
      - subsets available as APSR, IPSR, EPSR



Processor Modes

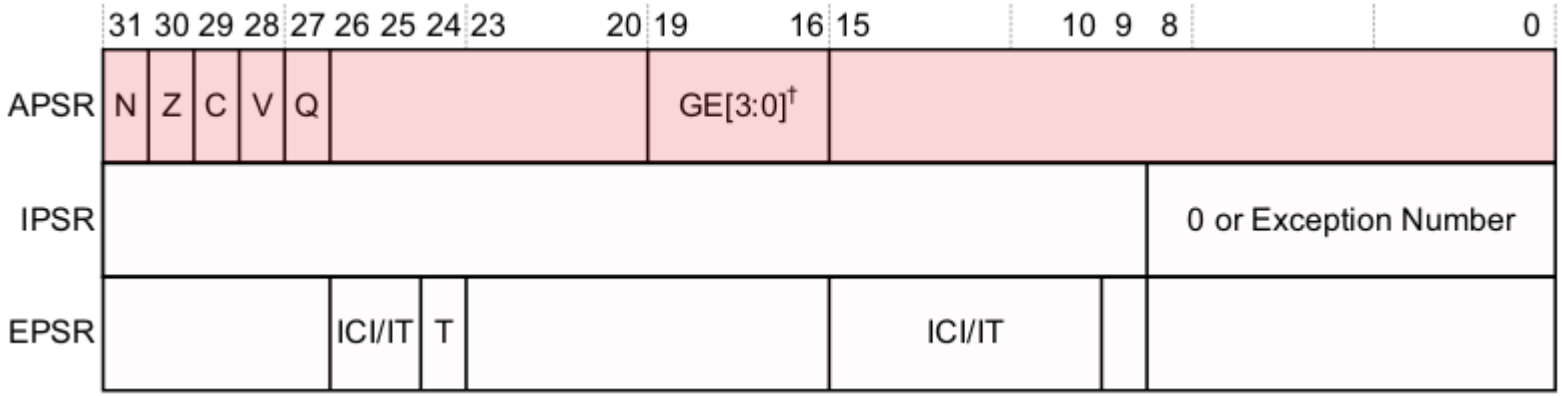
- ARMv7-M

Mode	Privilege	Stack pointer	Typical usage model
Handler	**Privileged**	Main	Exception handling
Thread		Main	Execution of a privileged process or thread using a common stack in a system that only supports privileged access
		Process	Execution of a privileged process or thread using a stack reserved for that process or thread in a system that only supports privileged access, or that supports a mix of privileged and unprivileged threads
Thread	**Unprivileged**	Main	Execution of an unprivileged process or thread using a common stack in a system that supports privileged and unprivileged access.
		Process	Execution of an unprivileged process or thread using a stack reserved for that process or thread in a system that supports privileged and unprivileged access

Main → SP_main (MSP)
Process → SP_process (PSP)

Program Status Registers

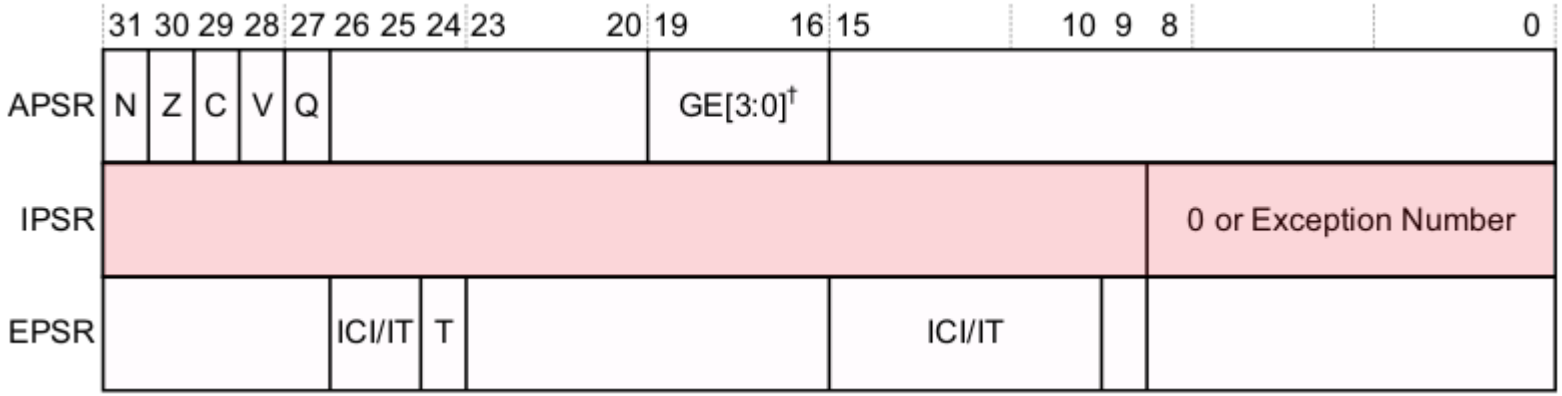
- ARMv7-M



- Application Program Status Register (APSR)
  - N: Negative result from ALU
  - Z: Zero result from ALU
  - C: ALU operation carry out
  - V: ALU operation overflow
  - Q: Saturated math overflow
  - GE: Reserved bits (DSP extension)

Program Status Registers

- ARMv7-M

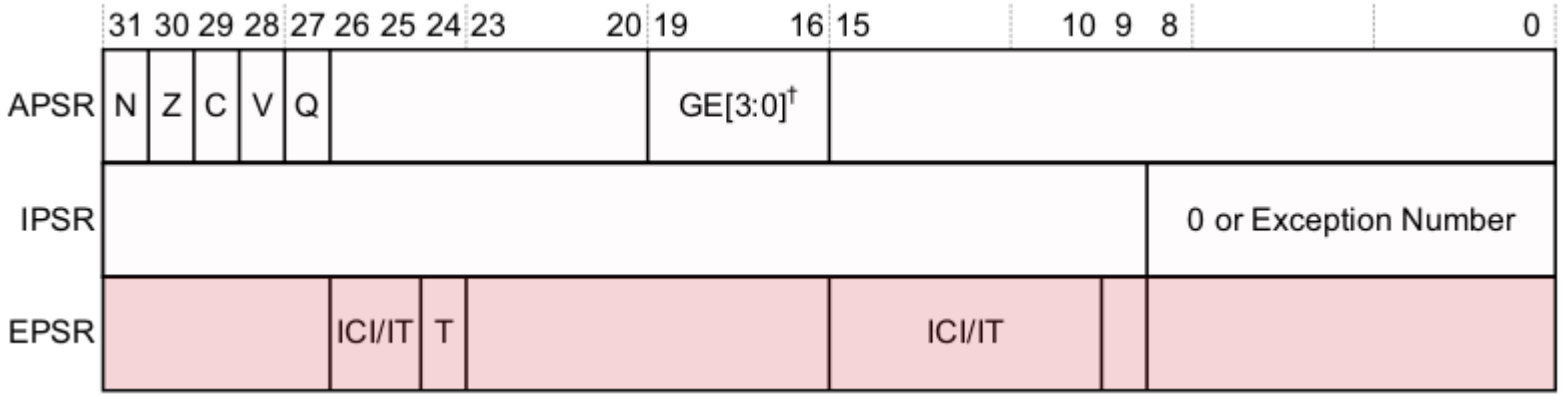


- Interrupt Program Status Register (IPSR)
  – When the processor is executing an exception handler, holds the exception number of the exception being processed. Otherwise, the IPSR value is zero.

    – Exception Number: currently executing exception and its entry vector.

Program Status Registers

- ARMv7-M



- Execution Program Status Register (EPSR)
  - T: current instruction set (always 1)
  - ICI/IT: IF-THEN base condition code or Interrupt continue information

Exception Handling

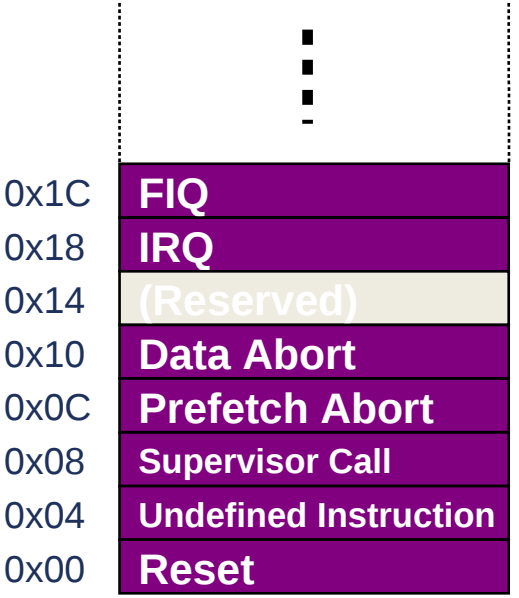
- ARMv7-AR
  - When an exception occurs, the core…

Exception Handling

- ARMv7-AR
  - When an exception occurs, the core…
    - ***1. Save processor status**
      - Copies CPSR into SPSR_<mode>
      - Stores the return address in LR_<mode>
      - Adjusts LR based on exception type

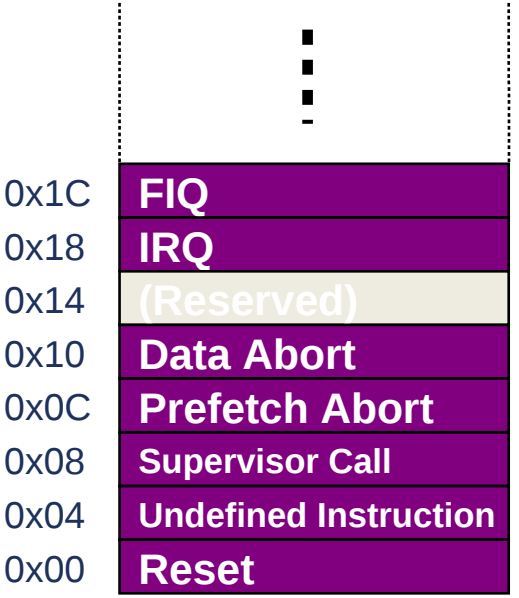
Exception Handling

- ARMv7-AR
  - When an exception occurs, the core…
    - ***1. Save processor status**
      - Copies CPSR into SPSR_<mode>
      - Stores the return address in LR_<mode>
      - Adjusts LR based on exception type
    - ***2. Change processor status**
      - Mode field bits
      - ARM or Thumb state
      - Interrupt disable bits
      - Sets PC to vector address



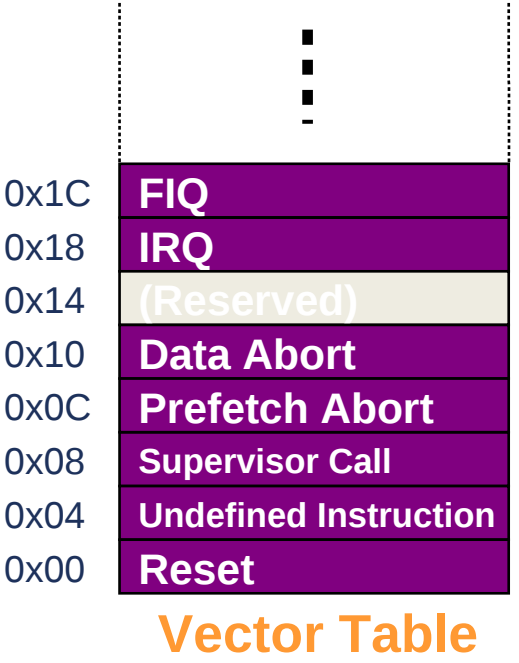
Exception Handling

- ARMv7-AR
  - When an exception occurs, the core…
    - ***1. Save processor status**
      - Copies CPSR into SPSR_<mode>
      - Stores the return address in LR_<mode>
      - Adjusts LR based on exception type
    - ***2. Change processor status**
      - Mode field bits
      - ARM or Thumb state
      - Interrupt disable bits
      - Sets PC to vector address
    - **3. Execute exception handler**
      - <users code>



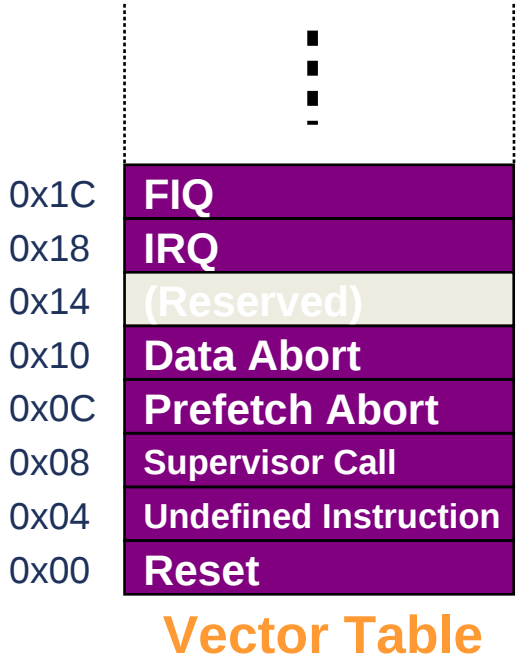
Exception Handling

- ARMv7-AR
  - When an exception occurs, the core…
    - ***1. Save processor status**
      - Copies CPSR into SPSR_<mode>
      - Stores the return address in LR_<mode>
      - Adjusts LR based on exception type
    - ***2. Change processor status**
      - Mode field bits
      - ARM or Thumb state
      - Interrupt disable bits
      - Sets PC to vector address
    - **3. Execute exception handler**
      - <users code>
  - To return, exception handler need to…



Exception Handling

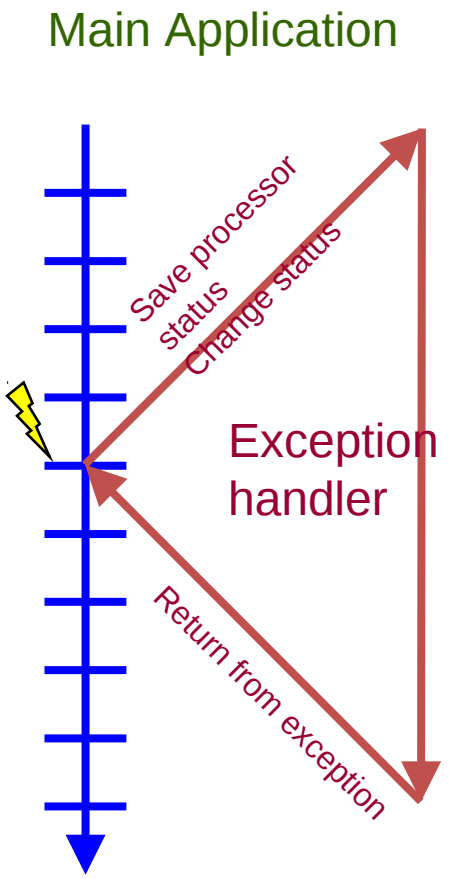
- ARMv7-AR
  - When an exception occurs, the core…
    - ***1. Save processor status**
      - Copies CPSR into SPSR_<mode>
      - Stores the return address in LR_<mode>
      - Adjusts LR based on exception type
    - ***2. Change processor status**
      - Mode field bits
      - ARM or Thumb state
      - Interrupt disable bits
      - Sets PC to vector address
    - **3. Execute exception handler**
      - <users code>
  - To return, exception handler need to…
    - **4. Return to main application**
      - Restore CPSR from SPSR_<mode>
      - Restore PC from LR_<mode>



* automatically performed by the core

Exception Handling

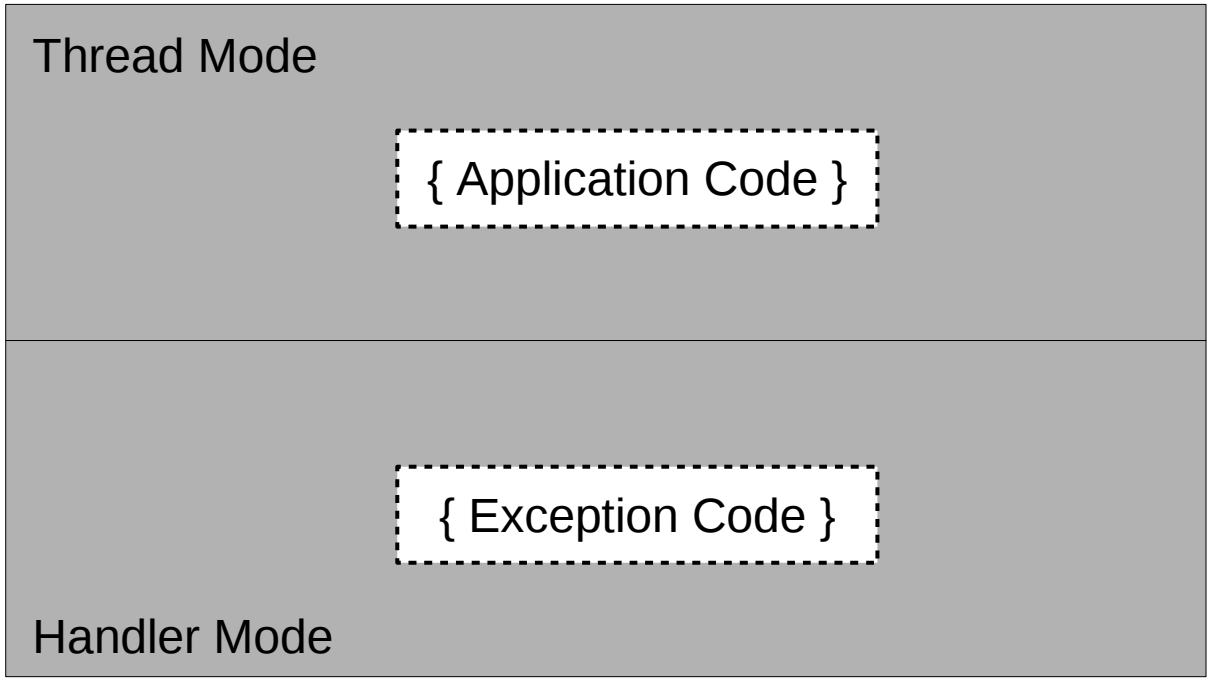
- ARMv7-AR
  - When an exception occurs, the core…
    - ***1. Save processor status**
      - Copies CPSR into SPSR_<mode>
      - Stores the return address in LR_<mode>
      - Adjusts LR based on exception type
    - ***2. Change processor status**
      - Mode field bits
      - ARM or Thumb state
      - Interrupt disable bits
      - Sets PC to vector address
    - **3. Execute exception handler**
      - <users code>
  - To return, exception handler need to…
    - **4. Return to main application**
      - Restore CPSR from SPSR_<mode>
      - Restore PC from LR_<mode>



* automatically performed by the core

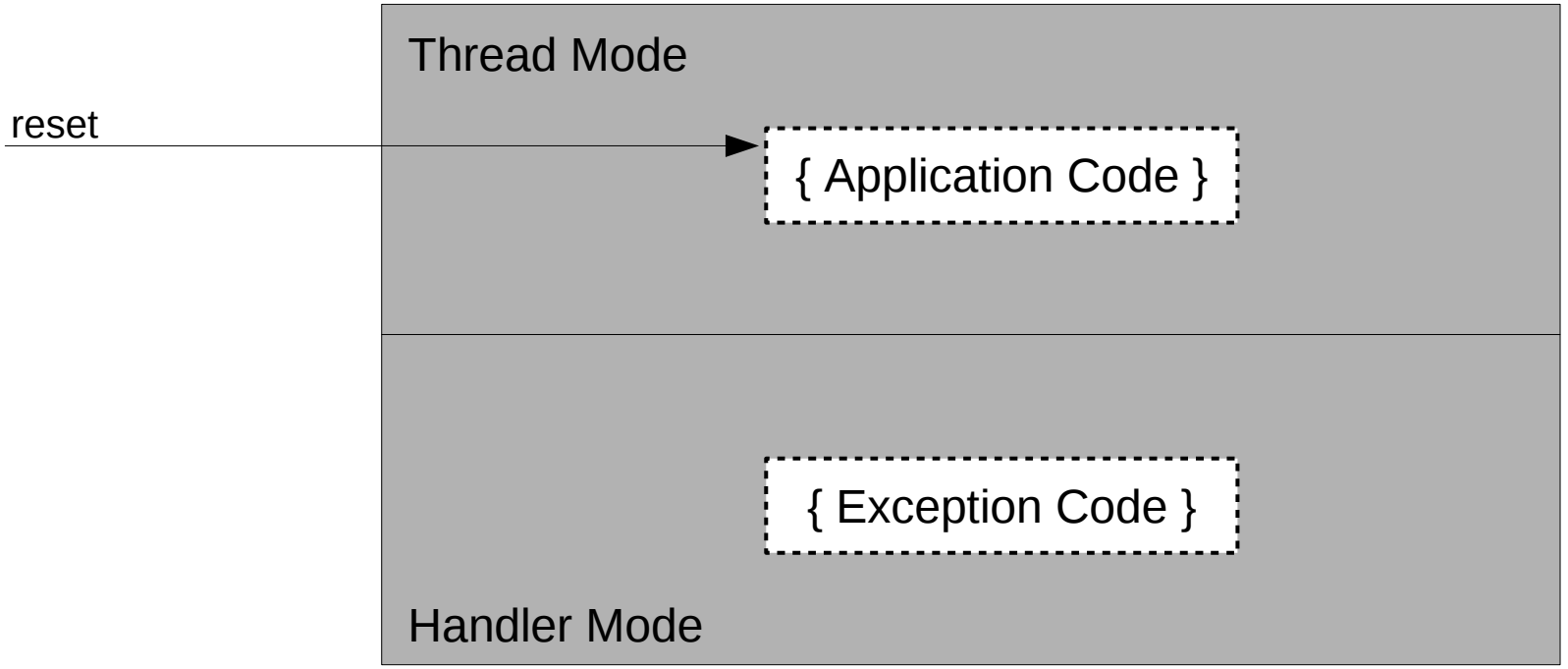
Exception Handling

- ARMv7-M
  - Processor Modes (overview)



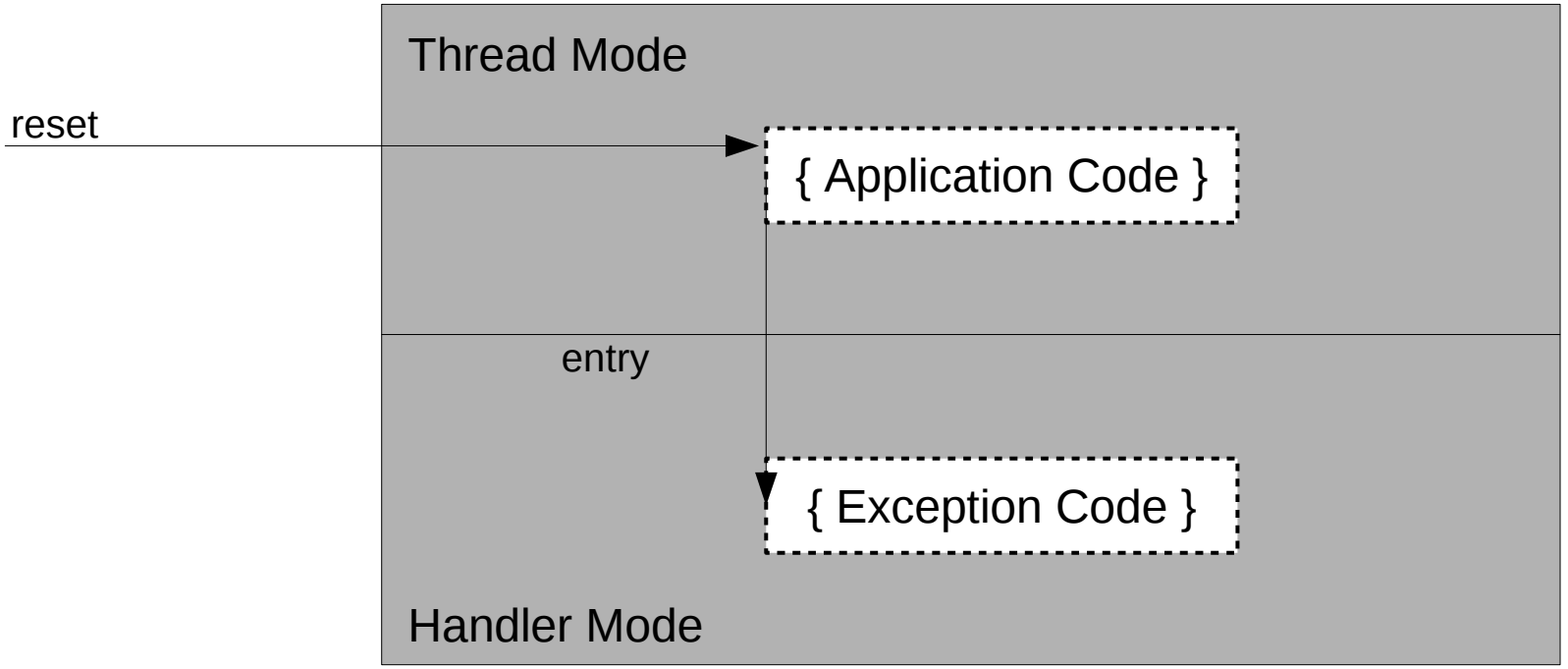
Exception Handling

- ARMv7-M
  - Processor Modes (overview)



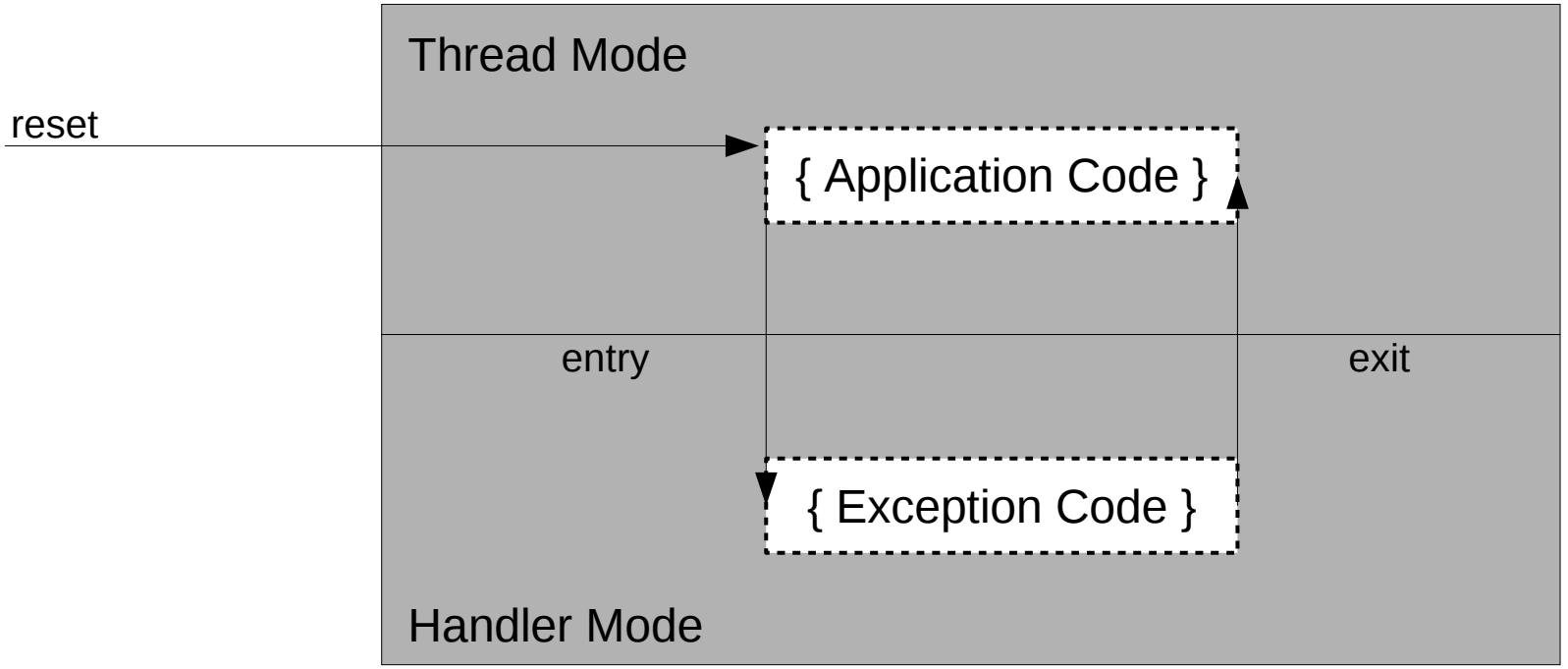
Exception Handling

- ARMv7-M
  - Processor Modes (overview)



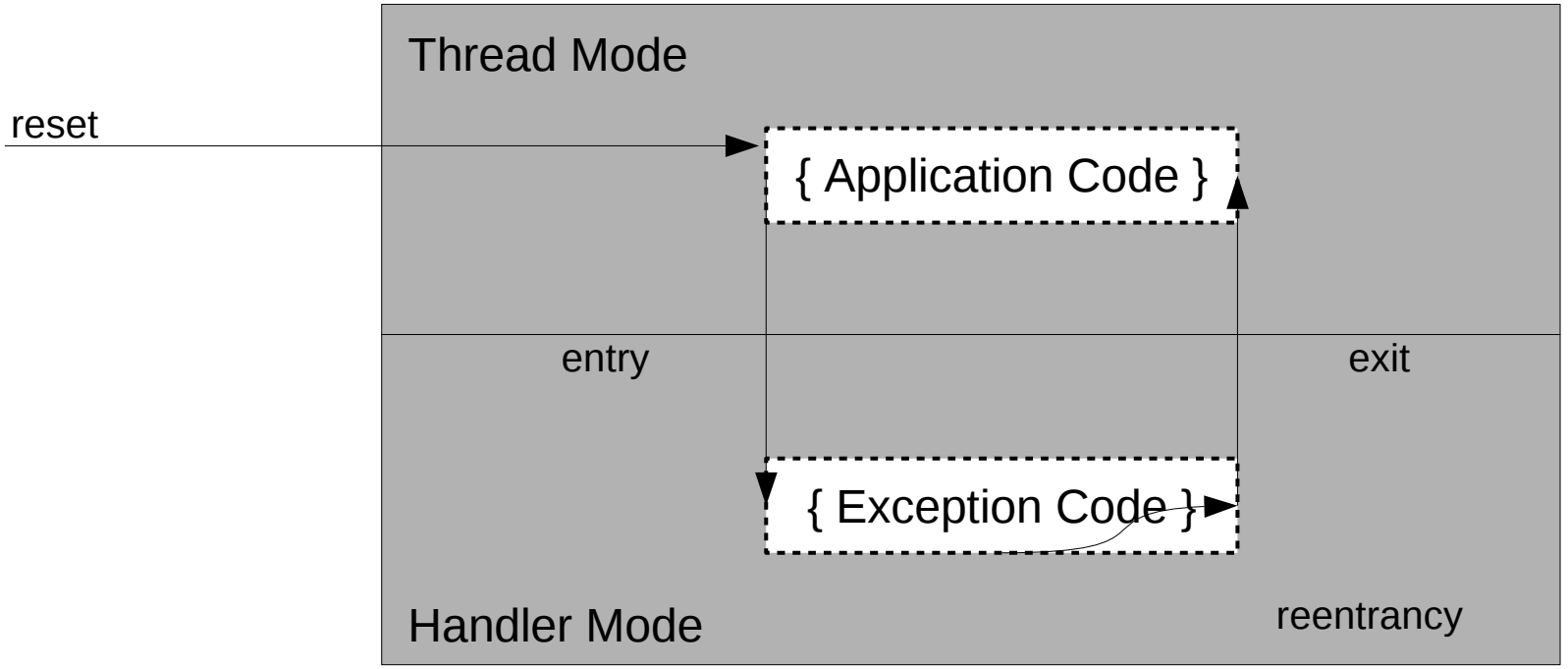
Exception Handling

- ARMv7-M
  - Processor Modes (overview)



Exception Handling

- ARMv7-M
  - Processor Modes (overview)



Exception Handling

- ARMv7-M

Exception Handling

- ARMv7-M
  - Exception types:
    - Reset
    - Non-maskable Interrupts (NMI)
    - Faults
    - PendSV
    - SVCall
    - External Interrupt
    - SysTick Interrupt

Exception Handling

- ARMv7-M
  - Exception types:
    - Reset
    - Non-maskable Interrupts (NMI)
    - Faults
    - PendSV
    - SVCall
    - External Interrupt
    - SysTick Interrupt
  - Exceptions processed in Handler mode (except reset)
    - Exceptions always run privileged

Exception Handling

- ARMv7-M
  - Exception types:
    - Reset
    - Non-maskable Interrupts (NMI)
    - Faults
    - PendSV
    - SVCall
    - External Interrupt
    - SysTick Interrupt
  - Exceptions processed in Handler mode (except reset)
    - Exceptions always run privileged
  - Interrupt handling
    - Interrupts are a sub-class of exception
    - Automatic save and restore processor registers
      {xPSR, PC, LR, R12, R3-R0}
    - Allows handler to be written entirely in ‘C’

Vector Table

- ARMv7-M
  - First entry contains initial Main SP

Address		Vector #
0x40 + 4*N	External N	16 + N
…	…	…
0x40	External 0	16
0x3C	SysTick	15
0x38	PendSV	14
0x34	Reserved	13
0x30	Debug Monitor	12
0x2C	SVC	11
0x1C to 0x28	Reserved (x4)	7-10
0x18	Usage Fault	6
0x14	Bus Fault	5
0x10	Mem Manage Fault	4
0x0C	Hard Fault	3
0x08	NMI	2
0x04	Reset	1
0x00	Initial Main SP	N/A

Vector Table

- ARMv7-M
  - First entry contains initial Main SP
  - All other entries are addresses for exception handler

Address		Vector #
0x40 + 4*N	External N	16 + N
…	…	…
0x40	External 0	16
0x3C	SysTick	15
0x38	PendSV	14
0x34	Reserved	13
0x30	Debug Monitor	12
0x2C	SVC	11
0x1C to 0x28	Reserved (x4)	7-10
0x18	Usage Fault	6
0x14	Bus Fault	5
0x10	Mem Manage Fault	4
0x0C	Hard Fault	3
0x08	NMI	2
0x04	Reset	1
0x00	Initial Main SP	N/A

Vector Table

- ARMv7-M
  - First entry contains initial Main SP
  - All other entries are addresses for exception handler
  - Table has up 496 external interrupts
    - Implementation-defined

Address		Vector #
0x40 + 4*N	External N	16 + N
…	…	…
0x40	External 0	16
0x3C	SysTick	15
0x38	PendSV	14
0x34	Reserved	13
0x30	Debug Monitor	12
0x2C	SVC	11
0x1C to 0x28	Reserved (x4)	7-10
0x18	Usage Fault	6
0x14	Bus Fault	5
0x10	Mem Manage Fault	4
0x0C	Hard Fault	3
0x08	NMI	2
0x04	Reset	1
0x00	Initial Main SP	N/A

Vector Table

- ARMv7-M
  - First entry contains initial Main SP
  - All other entries are addresses for exception handler
  - Table has up 496 external interrupts
    - Implementation-defined
  - Table may be relocated
    - Use vector table offset register
    - Still require minimal table entries (0x0) for booting the core

Address		Vector #
0x40 + 4*N	External N	16 + N
…	…	…
0x40	External 0	16
0x3C	SysTick	15
0x38	PendSV	14
0x34	Reserved	13
0x30	Debug Monitor	12
0x2C	SVC	11
0x1C to 0x28	Reserved (x4)	7-10
0x18	Usage Fault	6
0x14	Bus Fault	5
0x10	Mem Manage Fault	4
0x0C	Hard Fault	3
0x08	NMI	2
0x04	Reset	1
0x00	Initial Main SP	N/A

Vector Table

- ARMv7-M
  - First entry contains initial Main SP
  - All other entries are addresses for exception handler
  - Table has up 496 external interrupts
    - Implementation-defined
  - Table may be relocated
    - Use vector table offset register
    - Still require minimal table entries (0x0) for booting the core
  - Each exception has a vector number
    - Used in Interrupt Control and State Register to indicate the active or pending exception type

Address		Vector #
0x40 + 4*N	External N	16 + N
…	…	…
0x40	External 0	16
0x3C	SysTick	15
0x38	PendSV	14
0x34	Reserved	13
0x30	Debug Monitor	12
0x2C	SVC	11
0x1C to 0x28	Reserved (x4)	7-10
0x18	Usage Fault	6
0x14	Bus Fault	5
0x10	Mem Manage Fault	4
0x0C	Hard Fault	3
0x08	NMI	2
0x04	Reset	1
0x00	Initial Main SP	N/A

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - – Tightly coupled with processor core

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - – Tightly coupled with processor core

      Why?

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - – Tightly coupled with processor core

      Why? **Low latency!**

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - – Tightly coupled with processor core

      Why? **Low latency!**
  - One Non-Maskable Interrupt (NMI) supported

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - – Tightly coupled with processor core

      Why? **Low latency!**
  - One Non-Maskable Interrupt (NMI) supported
  - Features:
    - – Supports up to 496 interrupts (impdef)

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - – Tightly coupled with processor core

      Why? **Low latency!**
  - One Non-Maskable Interrupt (NMI) supported
  - Features:
    - – Supports up to 496 interrupts (impdef)
    - – Programmable priority level (0-255) for each interrupt

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - Tightly coupled with processor core

      Why? **Low latency!**
  - One Non-Maskable Interrupt (NMI) supported
  - Features:
    - Supports up to 496 interrupts (impdef)
    - Programmable priority level (0-255) for each interrupt
    - Support for tail-chaining
    - Support for late arrival

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - Tightly coupled with processor core

      Why? **Low latency!**
  - One Non-Maskable Interrupt (NMI) supported
  - Features:
    - Supports up to 496 interrupts (impdef)
    - Programmable priority level (0-255) for each interrupt
    - Support for tail-chaining
    - Support for late arrival

      → **Less overhead of saving and restoring context.**

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - Tightly coupled with processor core

      Why? **Low latency!**
  - One Non-Maskable Interrupt (NMI) supported
  - Features:
    - Supports up to 496 interrupts (impdef)
    - Programmable priority level (0-255) for each interrupt
    - Support for tail-chaining
    - Support for late arrival

      → **Less overhead of saving and restoring context.**
    - xPSR

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - Tightly coupled with processor core

      Why? **Low latency!**
  - One Non-Maskable Interrupt (NMI) supported
  - Features:
    - Supports up to 496 interrupts (impdef)
    - Programmable priority level (0-255) for each interrupt
    - Support for tail-chaining
    - Support for late arrival

      → **Less overhead of saving and restoring context.**
    - xPSR
      - automatically saved on int. entry
      - automatically saved on int. exit

NVIC – Nested Vectored Interrupt Controller

- ARMv7-M
  - External interrupts are handled by NVIC
    - Tightly coupled with processor core

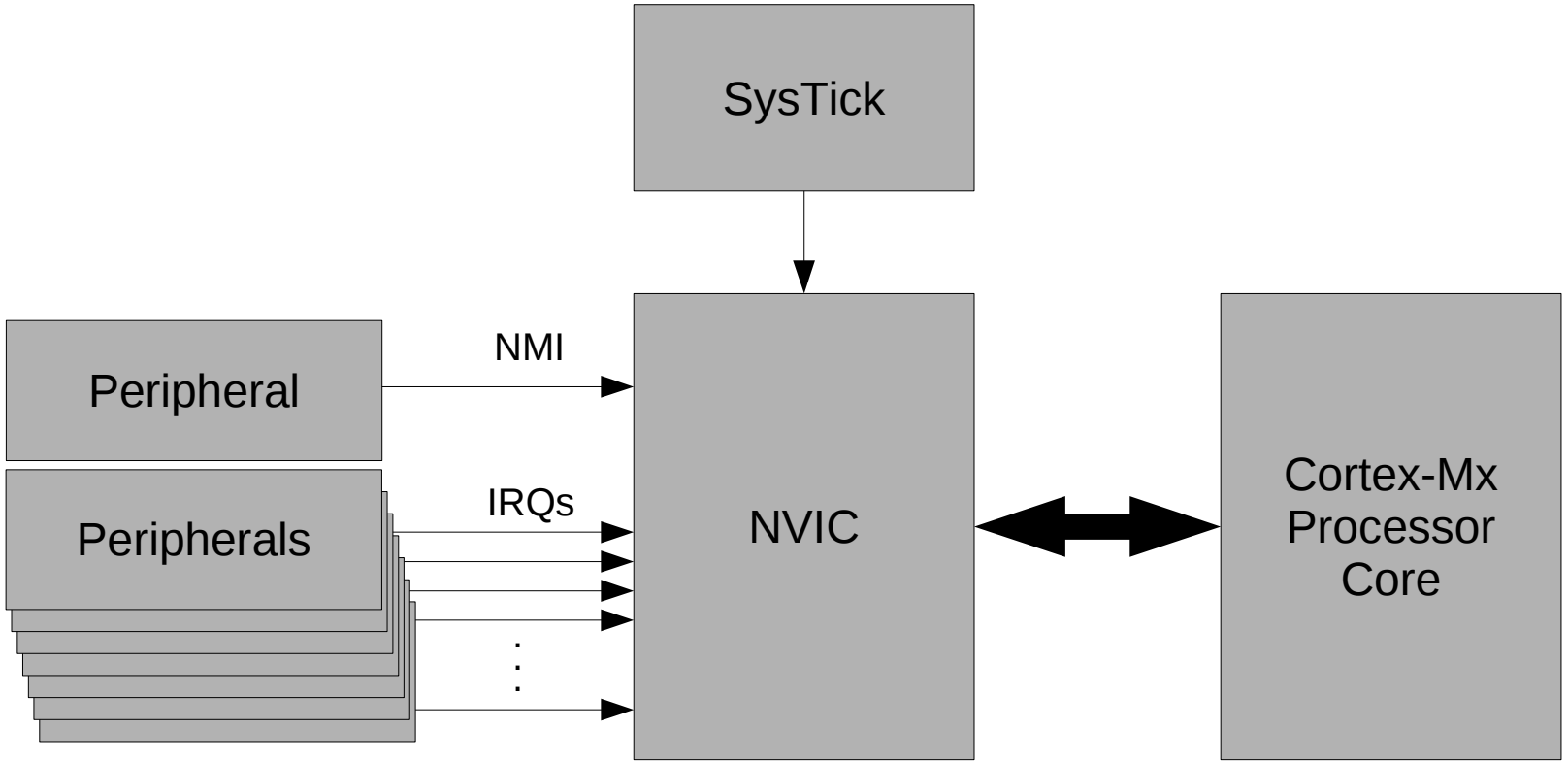
      Why? **Low latency!**
  - One Non-Maskable Interrupt (NMI) supported
  - Features:
    - Supports up to 496 interrupts (impdef)
    - Programmable priority level (0-255) for each interrupt
    - Support for tail-chaining
    - Support for late arrival

      → **Less overhead of saving and restoring context.**
    - xPSR
      - automatically saved on int. entry
      - automatically saved on int. exit

      → **No instruction overhead**

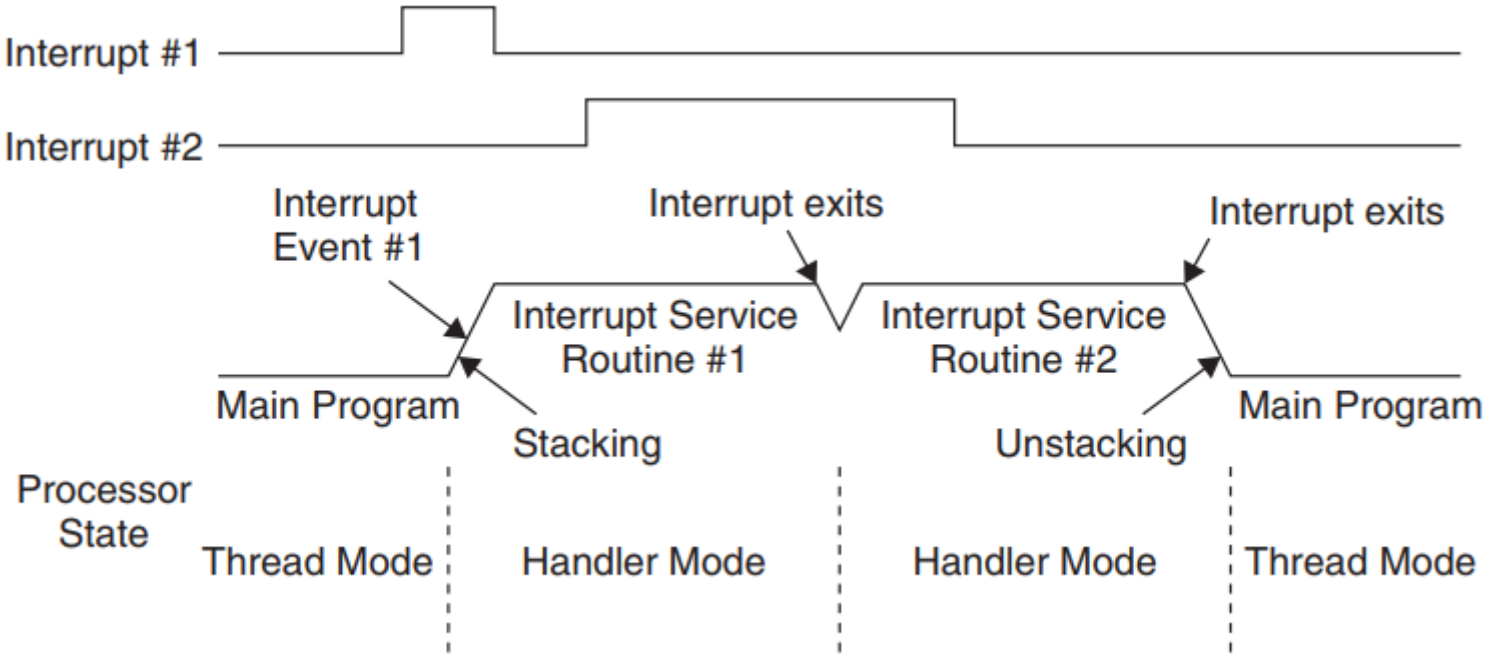
NVIC – Overview

- ARMv7-M



NVIC – Tail Chaining

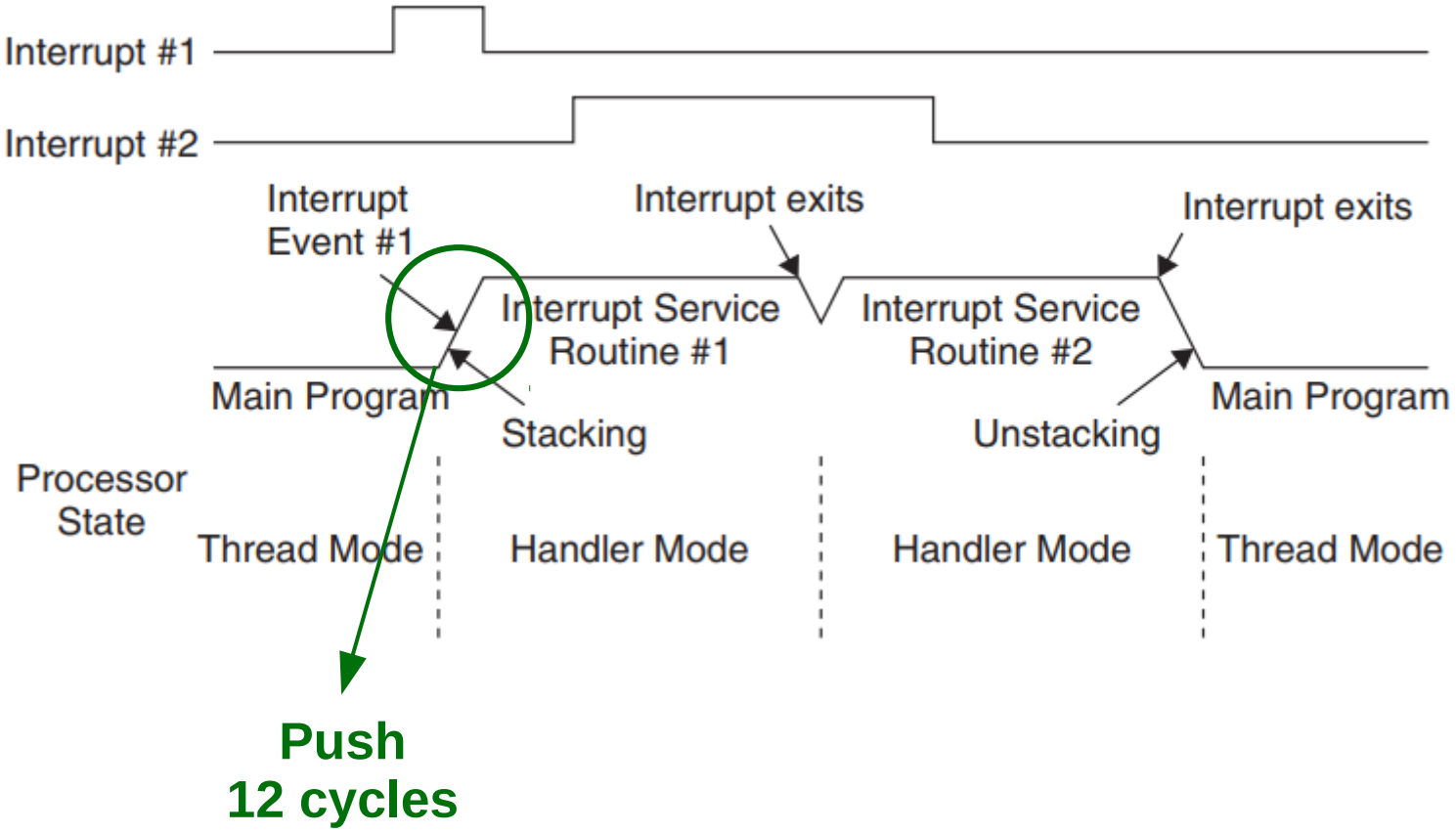
- ARMv7-M



From: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/slides/lec7.pdf

NVIC – Tail Chaining

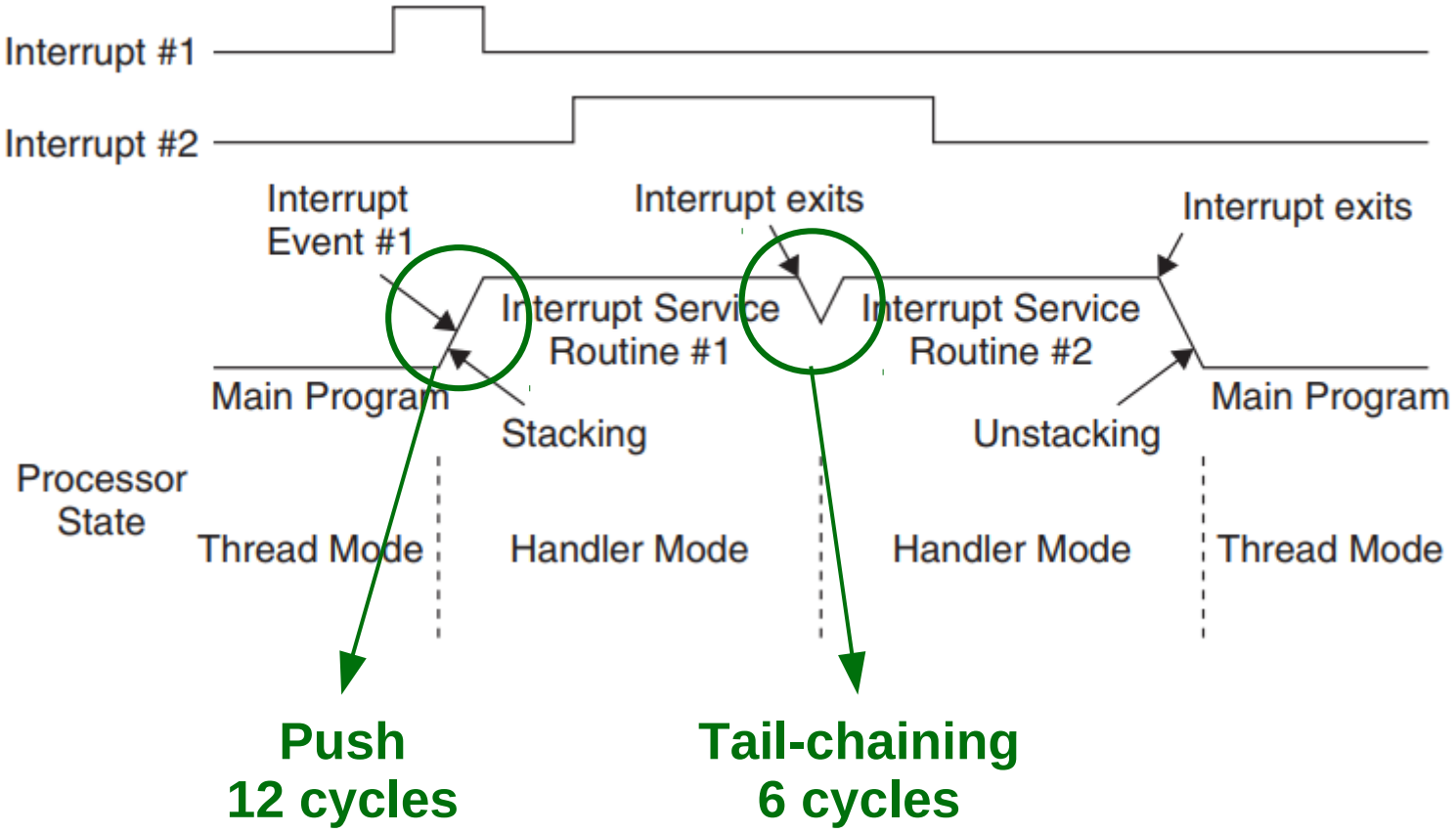
- ARMv7-M



From: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/slides/lec7.pdf

NVIC – Tail Chaining

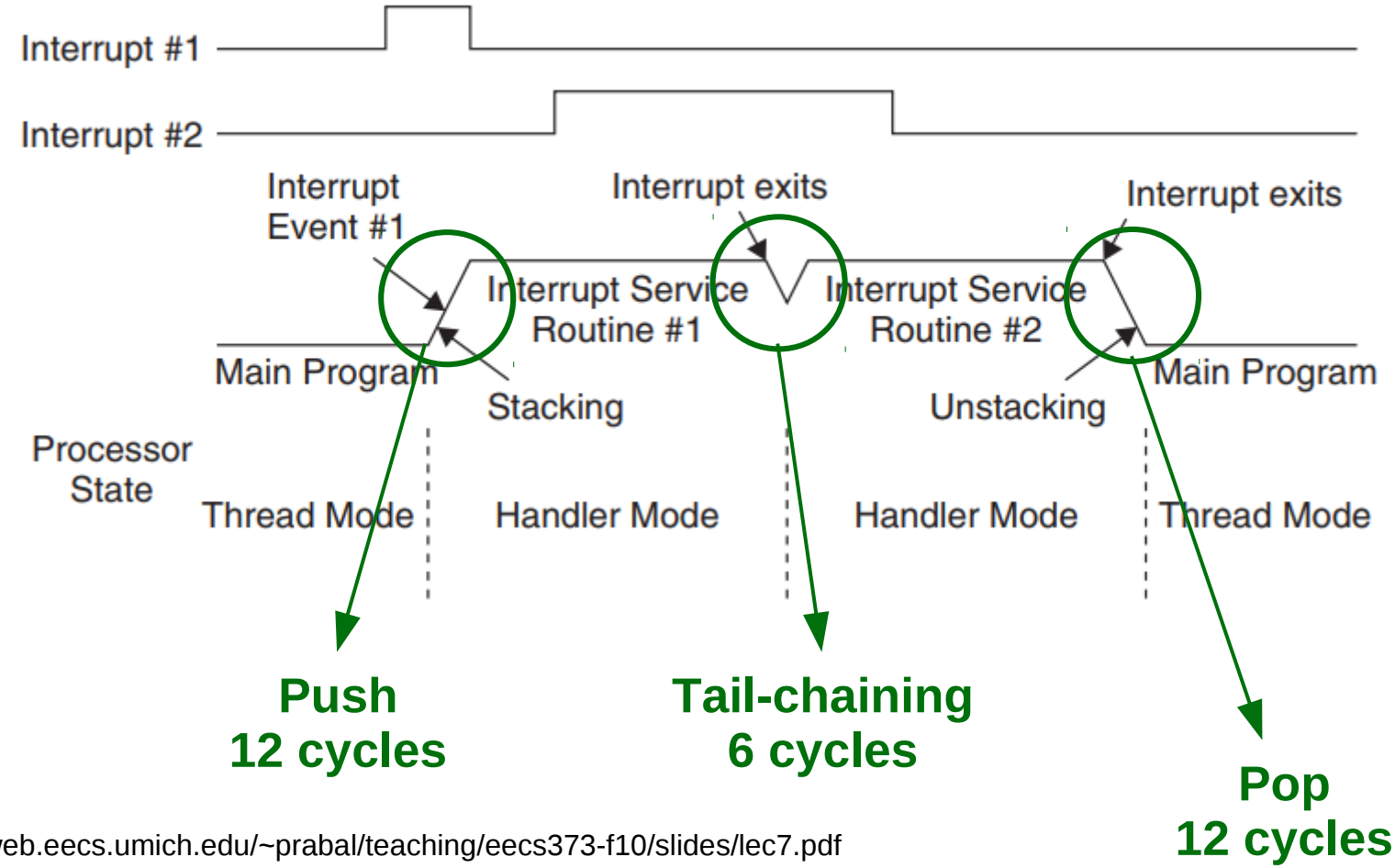
- ARMv7-M



From: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/slides/lec7.pdf

NVIC – Tail Chaining

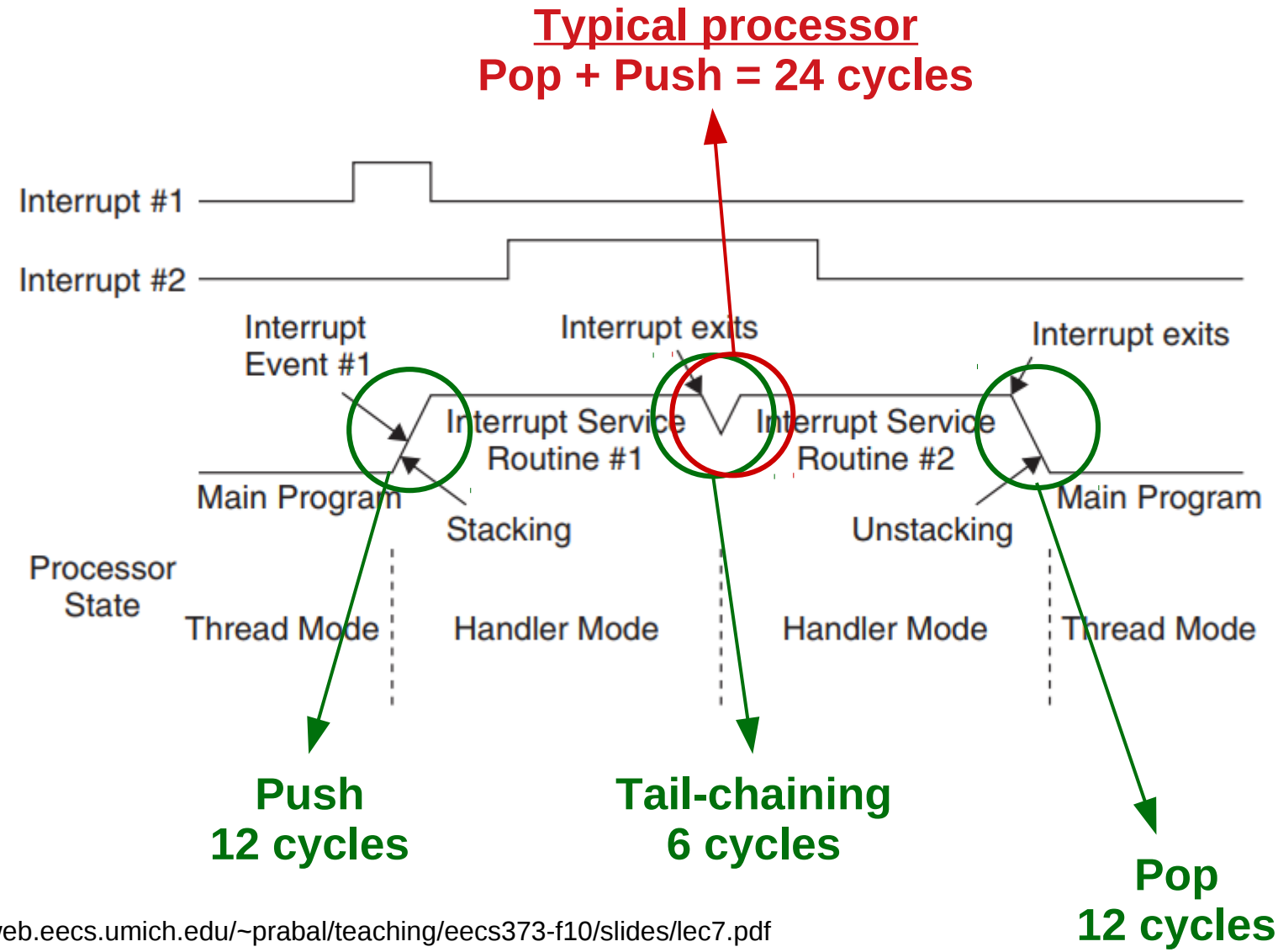
- ARMv7-M



From: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/slides/lec7.pdf

NVIC – Tail Chaining

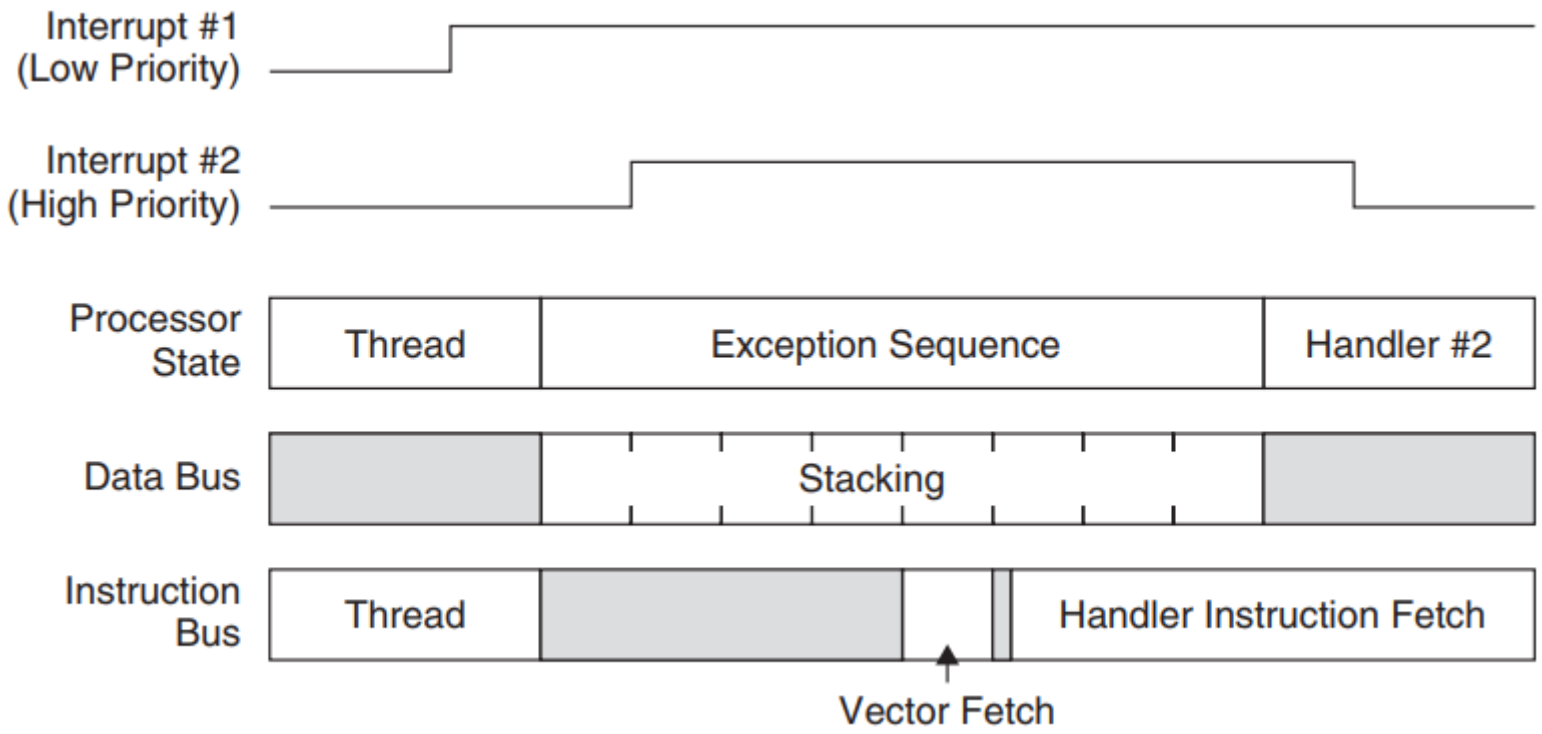
- ARMv7-M



From: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/slides/lec7.pdf

NVIC – Late Arrival

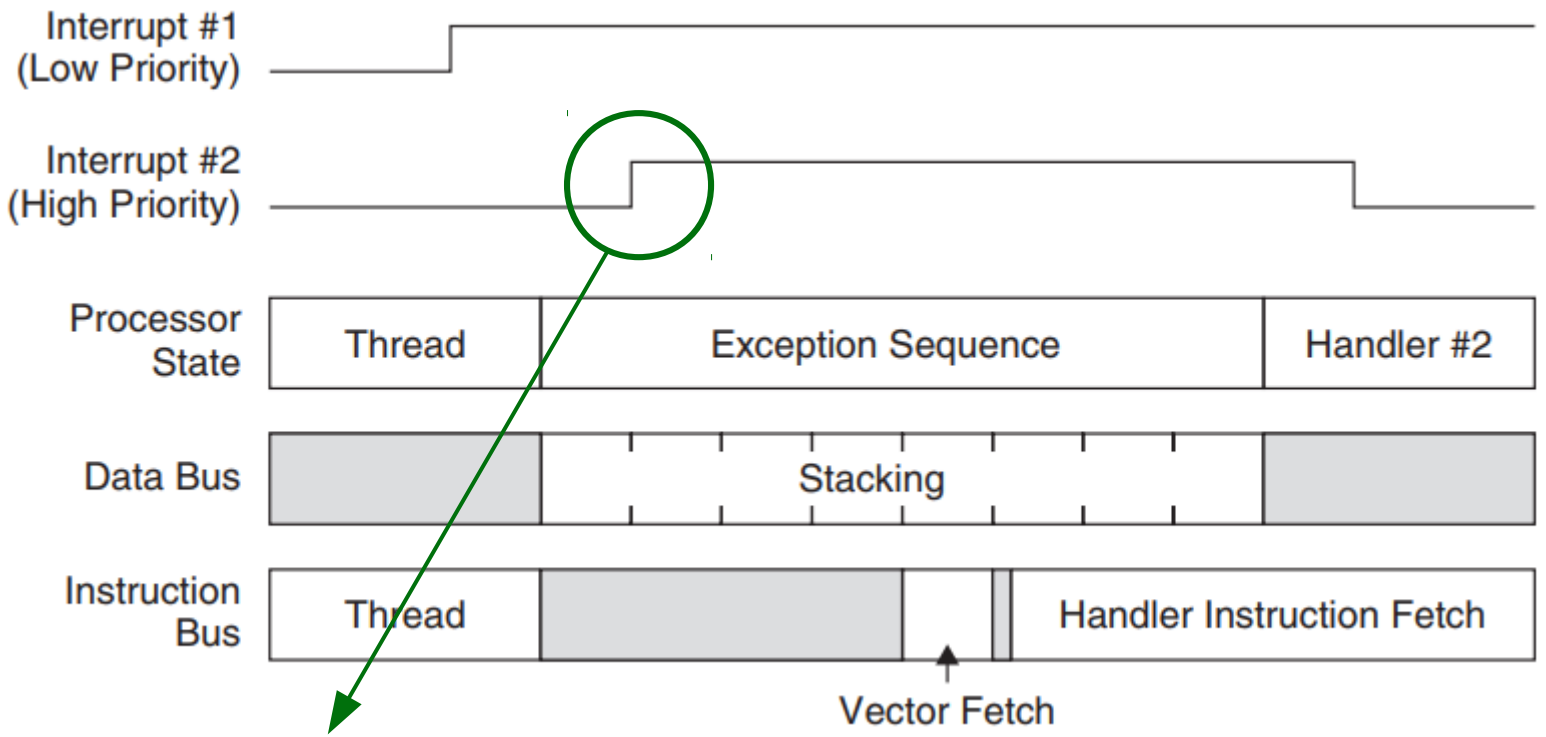
- ARMv7-M



From: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/slides/lec7.pdf

NVIC – Late Arrival

- ARMv7-M

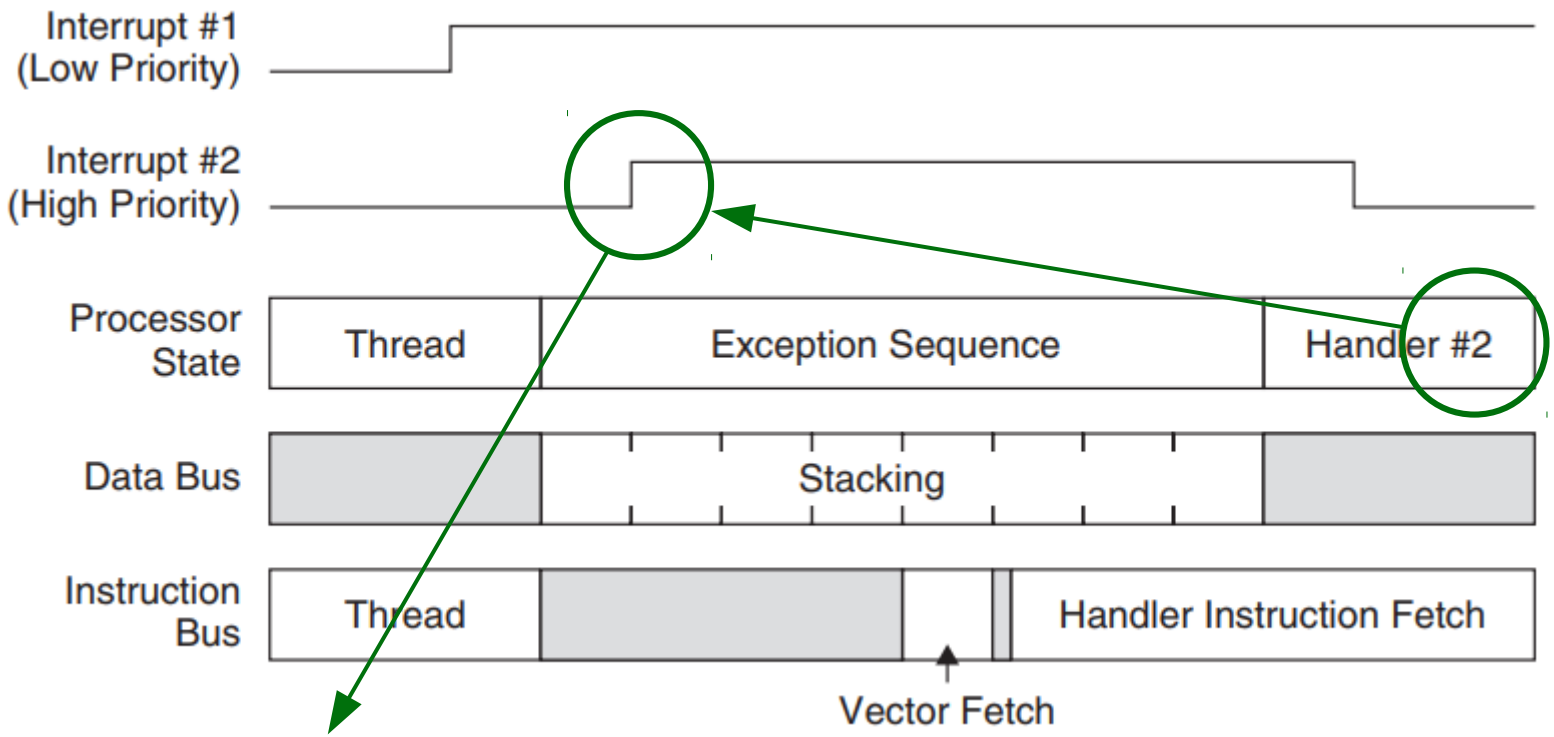


Late arrival of a higher priority interrupt

From: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/slides/lec7.pdf

NVIC – Late Arrival

- ARMv7-M

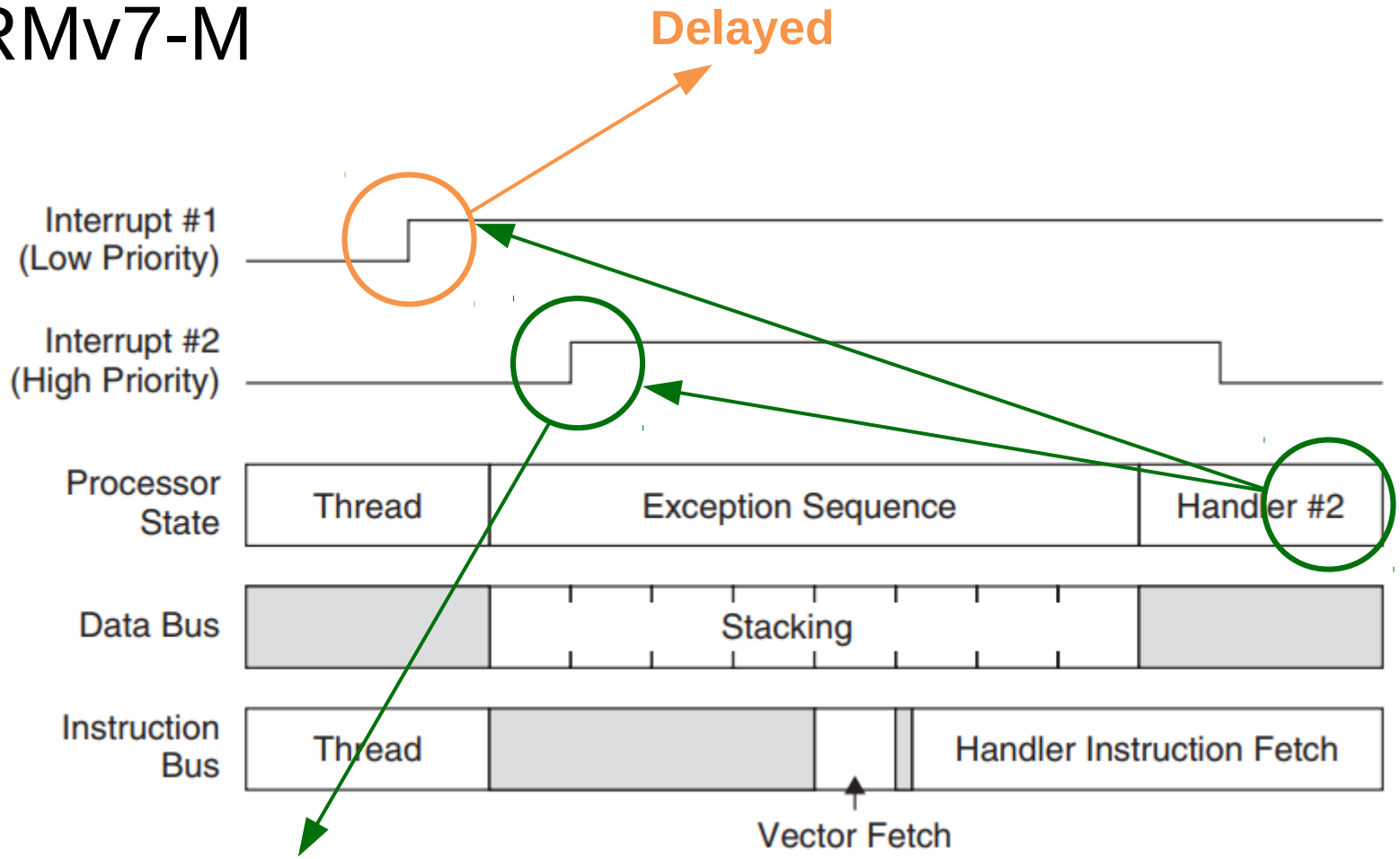


Late arrival of a higher priority interrupt

From: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/slides/lec7.pdf

NVIC – Late Arrival

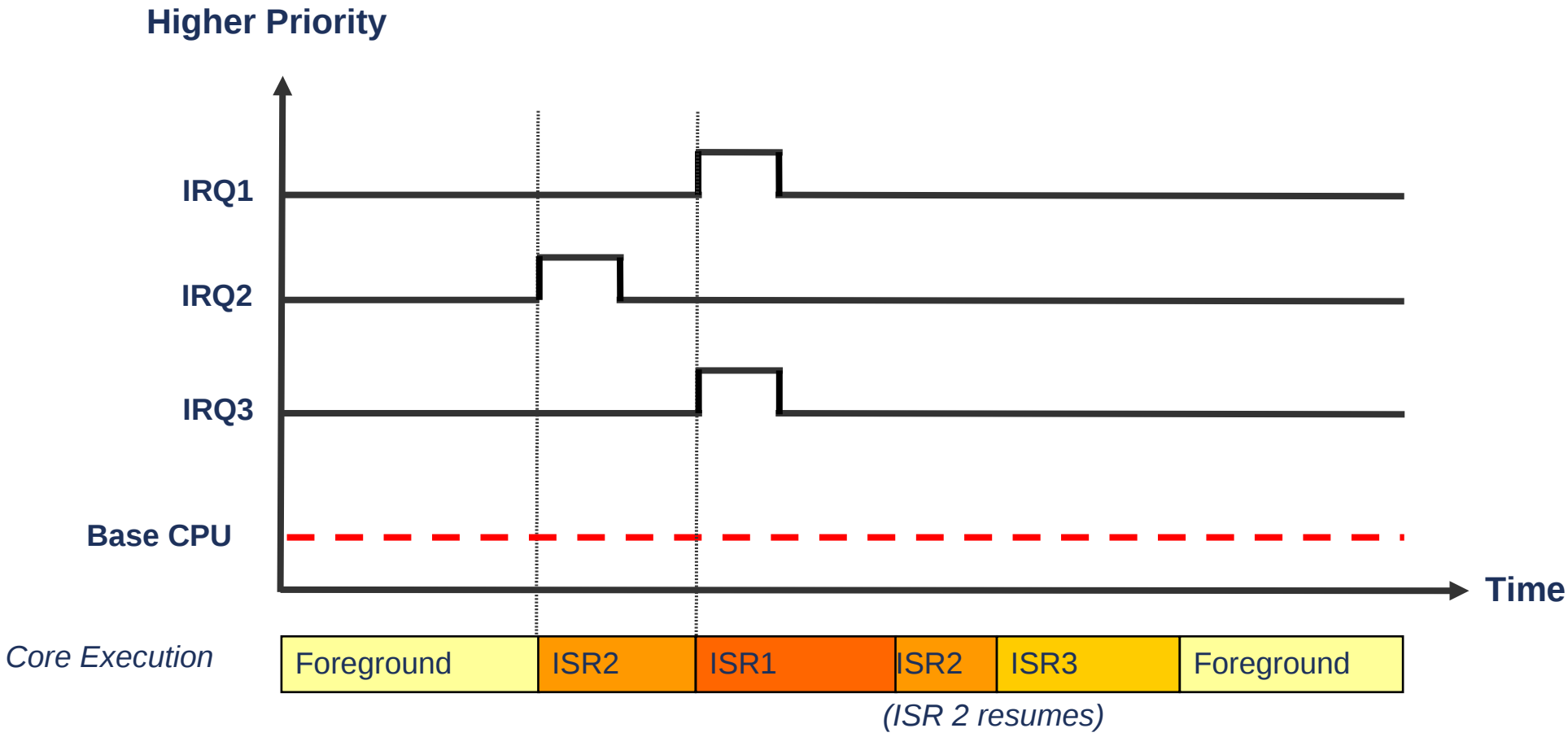
- ARMv7-M



From: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/slides/lec7.pdf

Exception Handling (example)

- ARMv7-M



NVIC – Registers

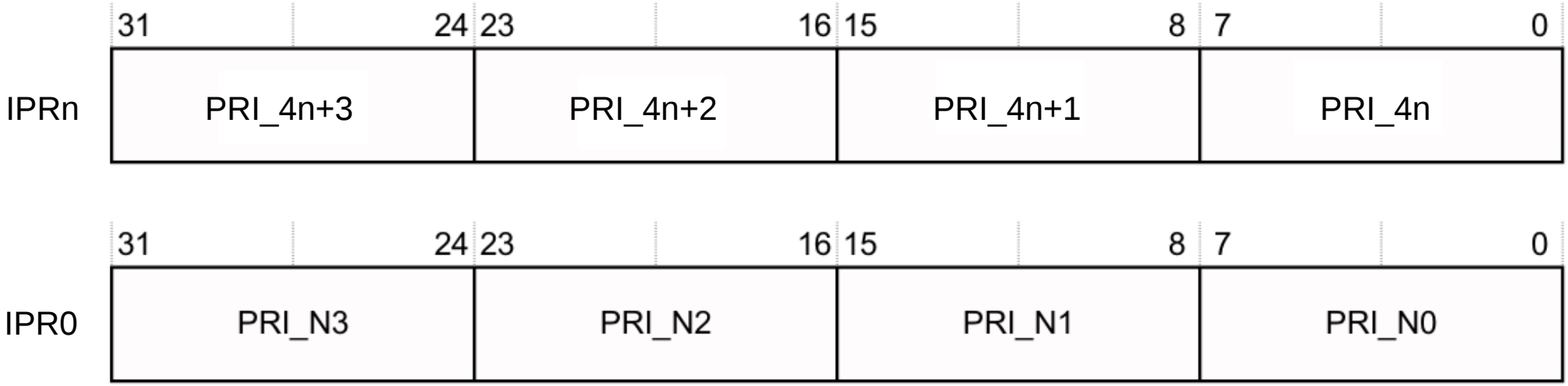
- ARMv7-M

Table B3-8 NVIC register summary

Address	Name	Type	Reset	Description
0xE000E100-0xE000E13C	NVIC_ISER0-NVIC_ISER15	RW	0x00000000	*Interrupt Set-Enable Registers, NVIC_ISER0-NVIC_ISER15* on page B3-684
0xE000E180-0xE000E1BC	NVIC_ICER0-NVIC_ICER15	RW	0x00000000	*Interrupt Clear-Enable Registers, NVIC_ICER0-NVIC_ICER15* on page B3-684
0xE000E200-0xE000E23C	NVIC_ISPR0-NVIC_ISPR15	RW	0x00000000	*Interrupt Set-Pending Registers, NVIC_ISPR0-NVIC_ISPR15* on page B3-685
0xE000E280-0xE000E2BC	NVIC_ICPR0-NVIC_ICPR15	RW	0x00000000	*Interrupt Clear-Pending Registers, NVIC_ICPR0-NVIC_ICPR15* on page B3-685
0xE000E300-0xE000E33C	NVIC_IABR0-NVIC_IABR15	RO	0x00000000	*Interrupt Active Bit Registers, NVIC_IABR0-NVIC_IABR15* on page B3-686
0xE000E340-0xE000E3FC	-	-	-	Reserved
0xE000E400-0xE000E5EC	NVIC_IPR0-NVIC_IPR123	RW	0x00000000	*Interrupt Priority Registers, NVIC_IPR0-NVIC_IPR123* on page B3-686
0xE000E5F0-0xE000ECFC	-	-	-	Reserved

NVIC – Registers

- ARMv7-M
  - Interrupt Priority Registers, NVIC_IPR 0 - 123



PRI_N3, bits[31:24] For register NVIC_IPR*n*, priority of interrupt number 4*n*+3.

PRI_N2, bits[23:16] For register NVIC_IPR*n*, priority of interrupt number 4*n*+2.

PRI_N1, bits[15:8] For register NVIC_IPR*n*, priority of interrupt number 4*n*+1.

PRI_N0, bits[7:0] For register NVIC_IPR*n*, priority of interrupt number 4*n*.

NVIC – Registers

- ARMv7-M

Table B3-9 Implemented NVIC registers, except NVIC_IPRs

ICTR.INTLINESNUM	Maximum number of interrupts	Last implemented NVIC_ISER	Corresponding interrupts
0b0000	32	NVIC_ISER0	0-31
0b0001	64	NVIC_ISER1	32-63
0b0010	96	NVIC_ISER2	64-95
0b0011	128	NVIC_ISER3	96-127
0b0100	160	NVIC_ISER4	128-159
0b0101	192	NVIC_ISER5	160-191
0b0110	224	NVIC_ISER6	192-223
0b0111	256	NVIC_ISER7	224-255
0b1000	288	NVIC_ISER8	256-287
0b1001	320	NVIC_ISER9	288-319
0b1010	352	NVIC_ISER10	320-351
0b1011	384	NVIC_ISER11	352-383
0b1100	416	NVIC_ISER12	384-415
0b1101	448	NVIC_ISER13	416-447
0b1110	480	NVIC_ISER14	448-479
0b1111	496	NVIC_ISER15	480-495

NVIC – Registers

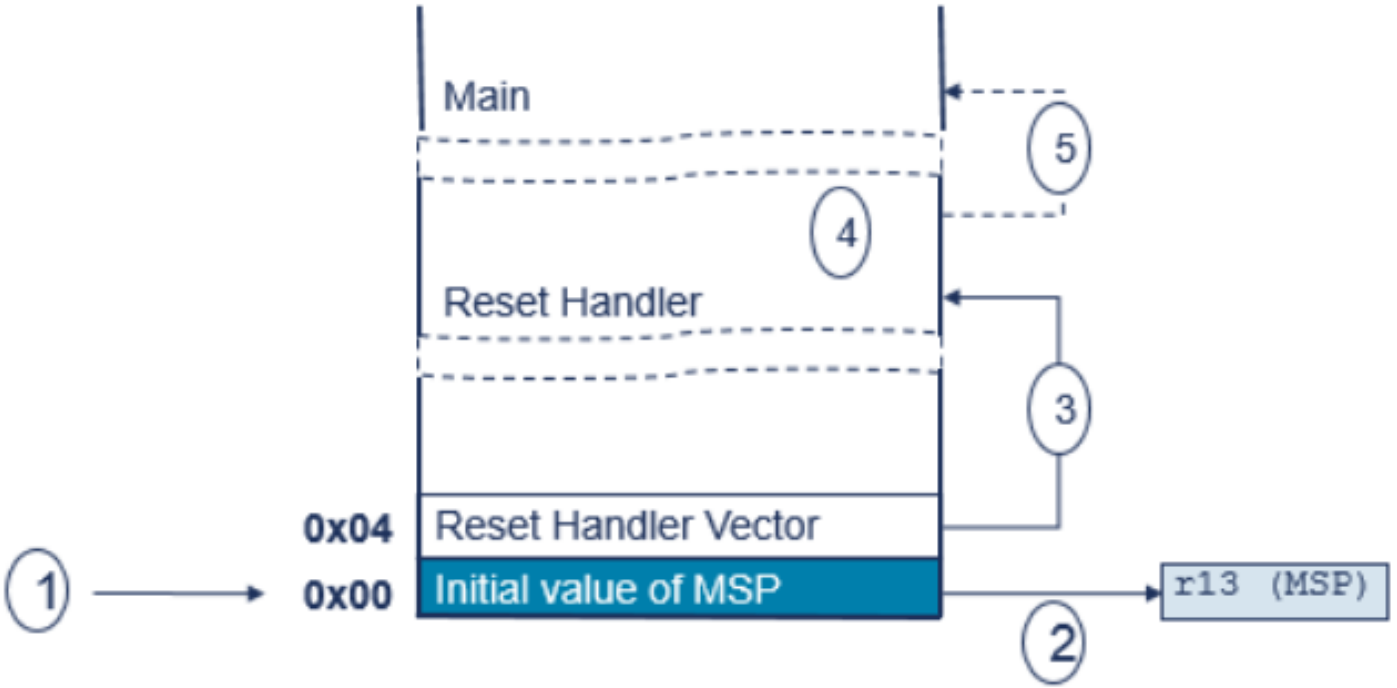
- ARMv7-M

Table B3-10 Implemented NVIC_IPRs

ICTR.INTLINESNUM	Maximum number of interrupts	Last implemented NVIC_IPR	Corresponding interrupts
0b0000	32	NVIC_IPR7	28-31
0b0001	64	NVIC_IPR15	60-63
0b0010	96	NVIC_IPR23	92-95
0b0011	128	NVIC_IPR31	124-127
0b0100	160	NVIC_IPR39	156-159
0b0101	192	NVIC_IPR47	188-191
0b0110	224	NVIC_IPR55	220-223
0b0111	256	NVIC_IPR63	252-255
0b1000	288	NVIC_IPR71	284-287
0b1001	320	NVIC_IPR79	316-319
0b1010	352	NVIC_IPR87	348-351
0b1011	384	NVIC_IPR95	380-383
0b1100	416	NVIC_IPR103	412-415
0b1101	448	NVIC_IPR111	444-447
0b1110	480	NVIC_IPR119	476-479
0b1111	496	NVIC_IPR123	492-495

Reset Behavior

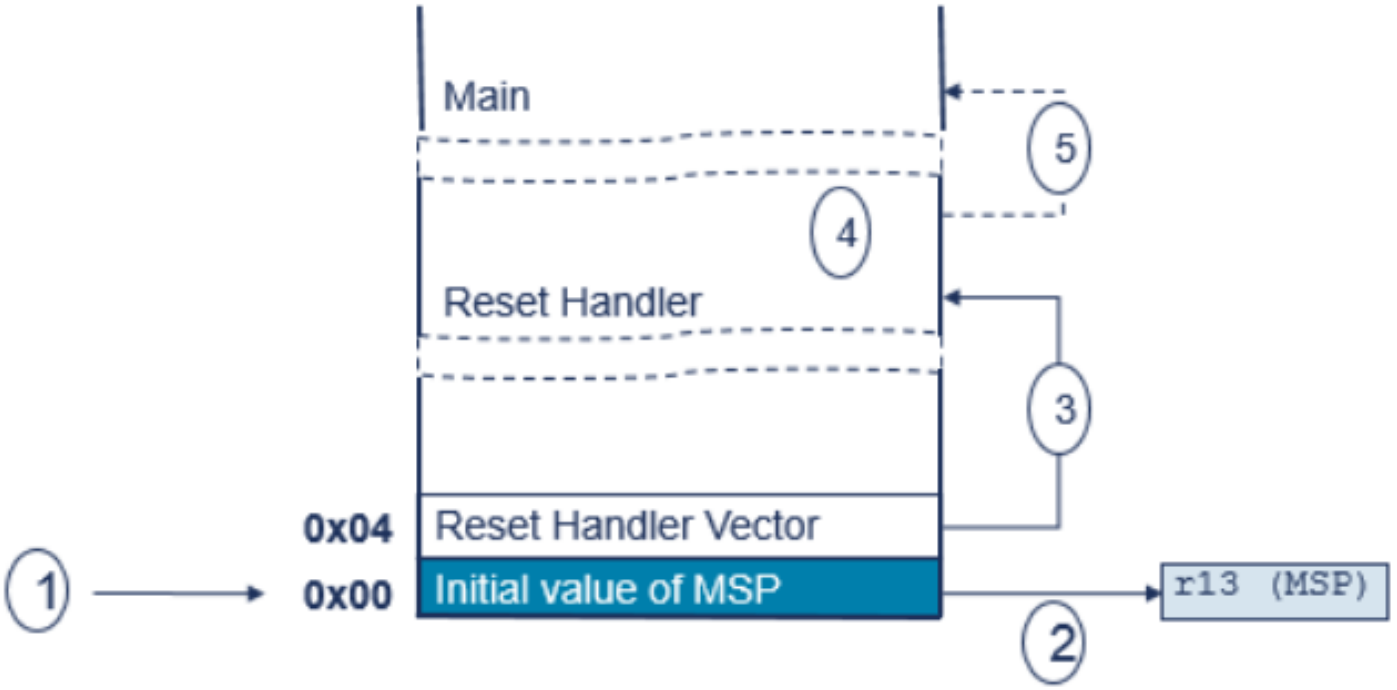
- ARMv7-M



Reset Behavior

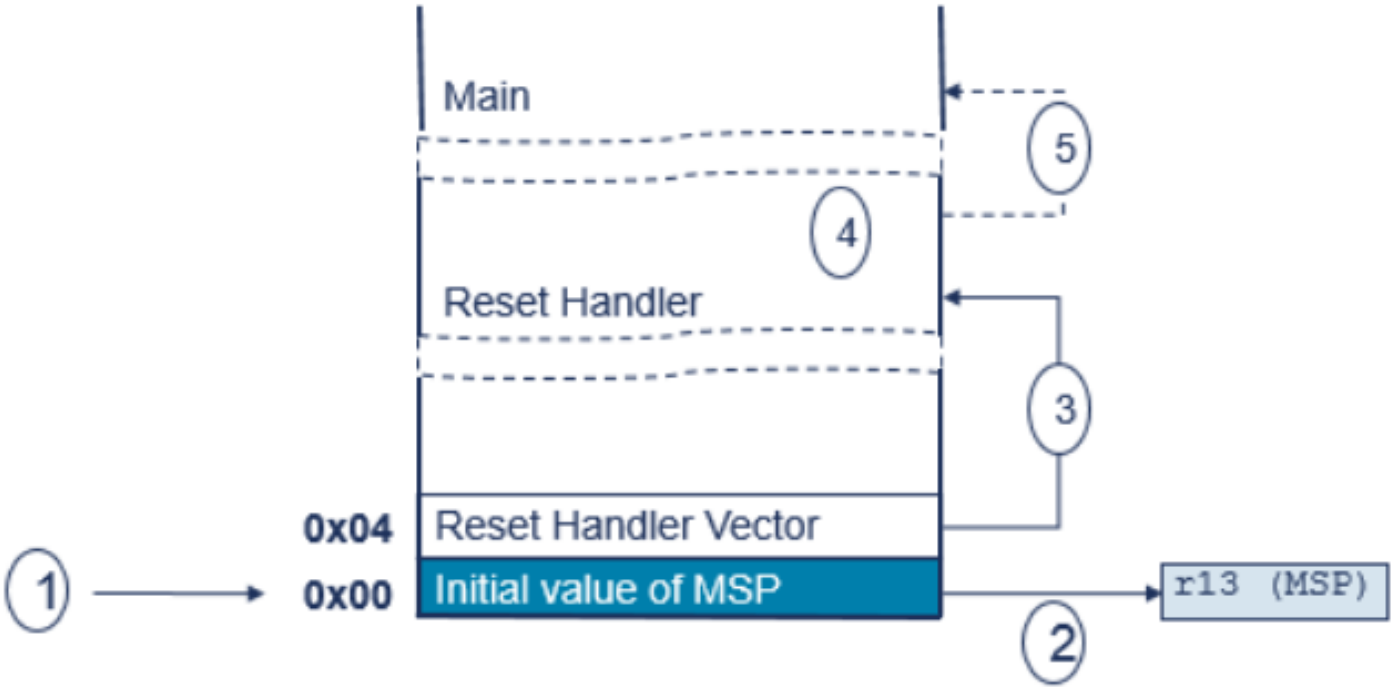
- ARMv7-M

  1. A reset occurs (reset input was asserted)



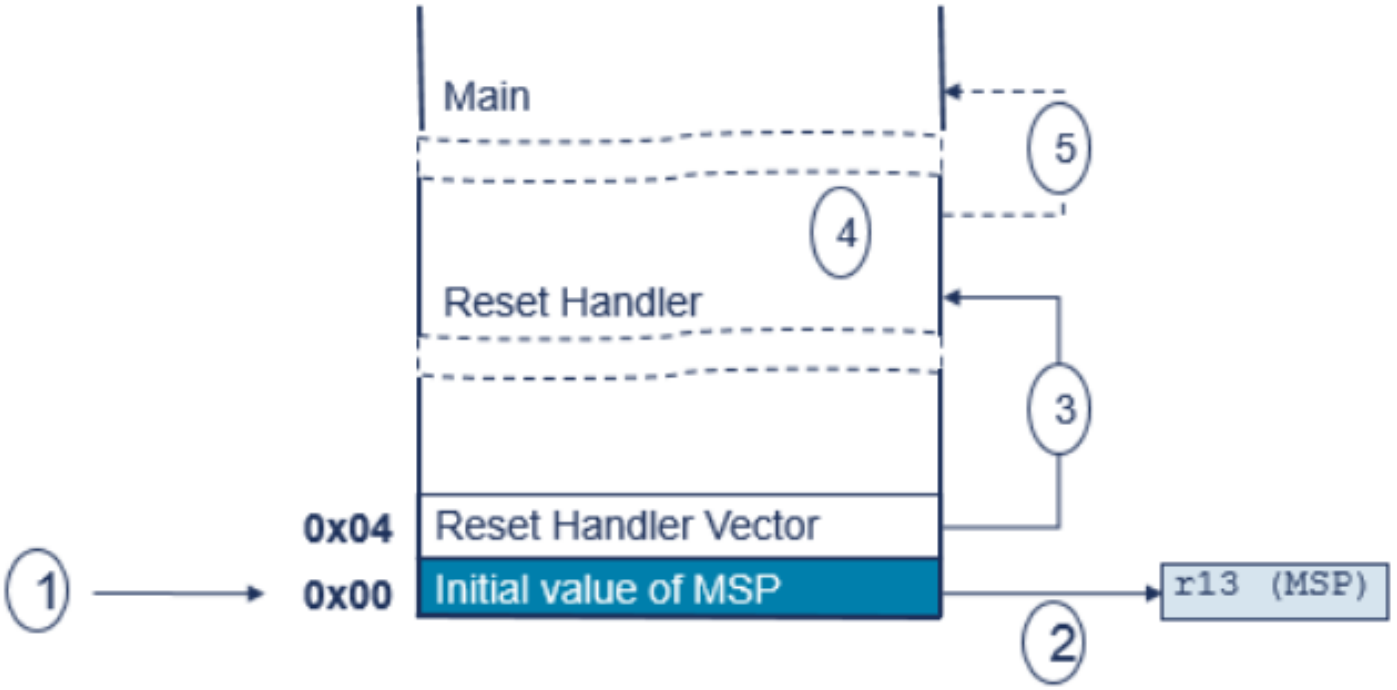
Reset Behavior

- ARMv7-M
  1. A reset occurs (reset input was asserted)
  2. Load MSP register initial value from address 0x00



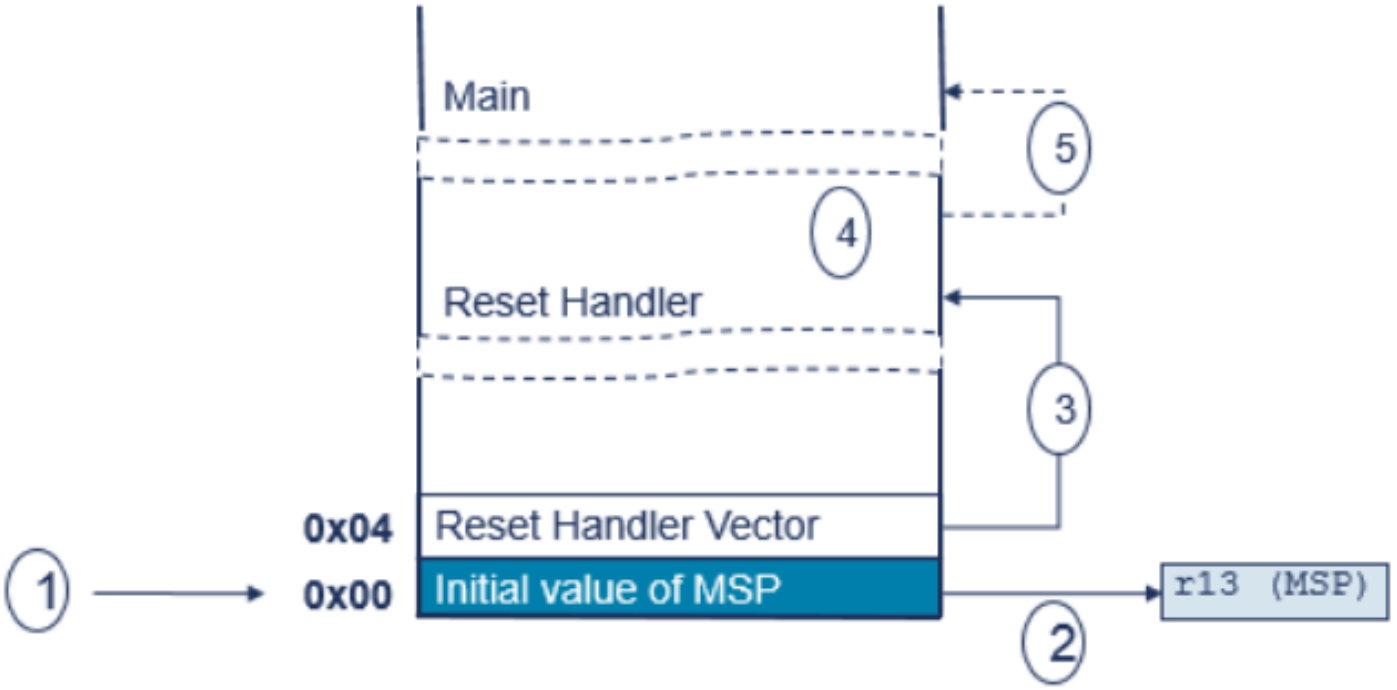
Reset Behavior

- ARMv7-M
  1. A reset occurs (reset input was asserted)
  2. Load MSP register initial value from address 0x00
  3. Load reset handler vector address from address 0x04



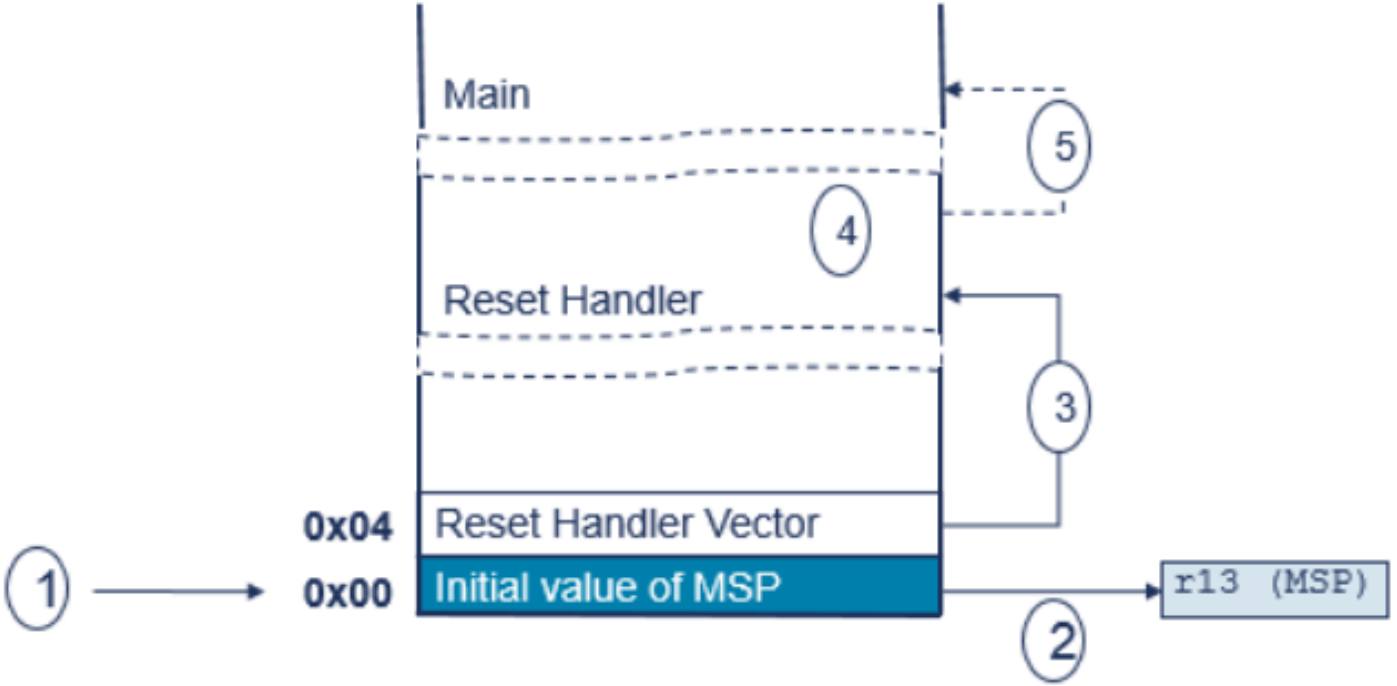
Reset Behavior

- ARMv7-M
  1. A reset occurs (reset input was asserted)
  2. Load MSP register initial value from address 0x00
  3. Load reset handler vector address from address 0x04
  4. Reset handler executes in Thread Mode



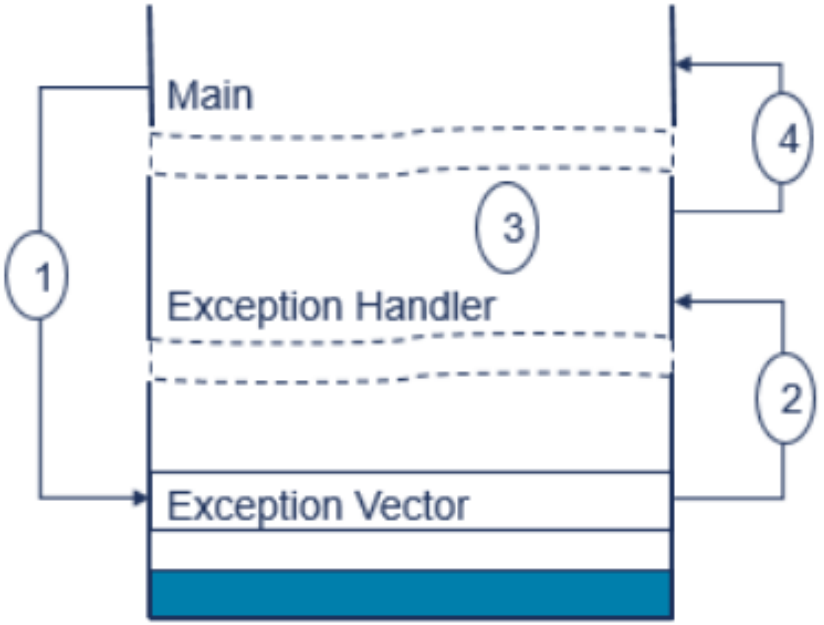
Reset Behavior

- ARMv7-M
  1. A reset occurs (reset input was asserted)
  2. Load MSP register initial value from address 0x00
  3. Load reset handler vector address from address 0x04
  4. Reset handler executes in Thread Mode
  5. Optional: reset handler branches to the main program



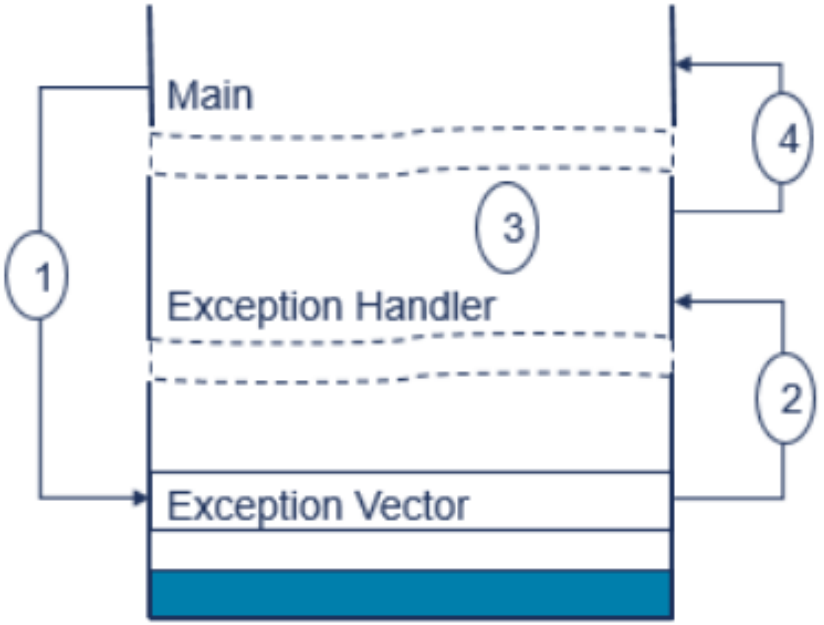
Exception Behavior

- ARMv7-M



Exception Behavior

- ARMv7-M
  1. Exception occurs
  - Current instruction stream stops
  - Processor accesses vector table

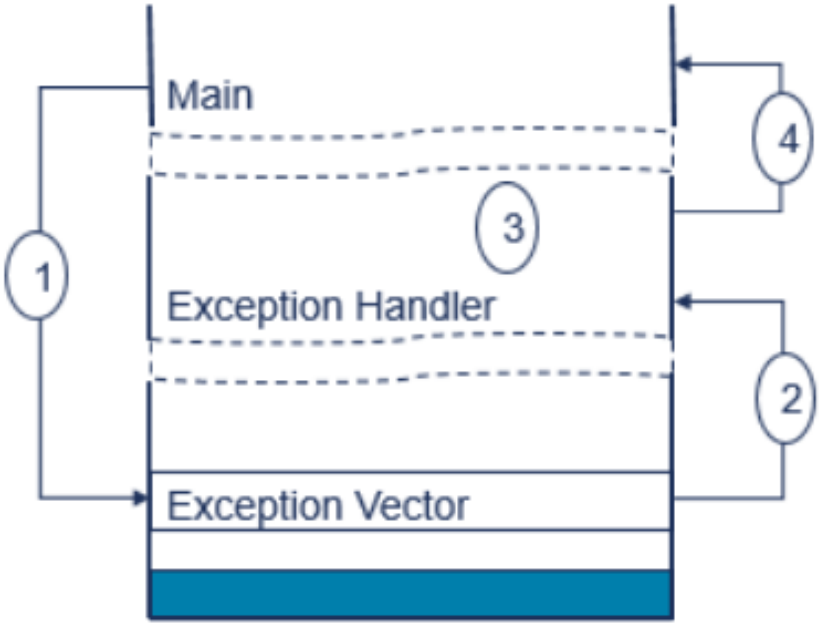


Exception Behavior

- ARMv7-M

1. Exception occurs
  - Current instruction stream stops
  - Processor accesses vector table

2. Vector address for the exception handler loaded from the vector table



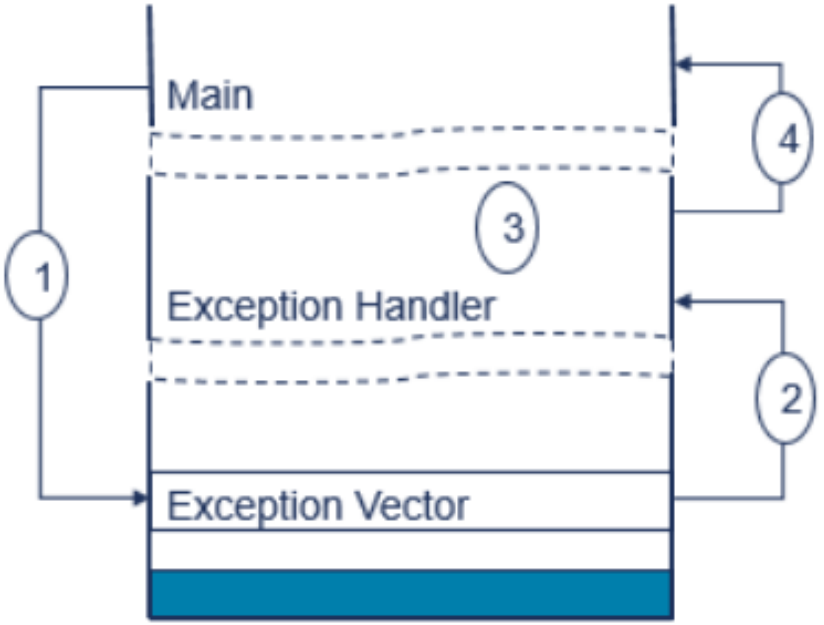
Exception Behavior

- ARMv7-M

1. Exception occurs
  - Current instruction stream stops
  - Processor accesses vector table

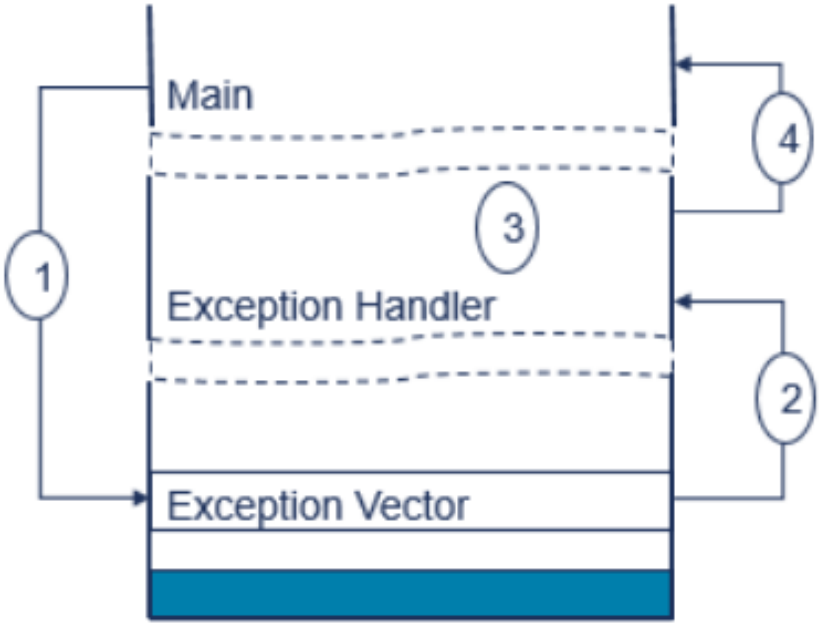
2. Vector address for the exception handler loaded from the vector table

3. Exception handler executes in Handler Mode



Exception Behavior

- ARMv7-M
  1. Exception occurs
     - Current instruction stream stops
     - Processor accesses vector table
  2. Vector address for the exception handler loaded from the vector table
  3. Exception handler executes in Handler Mode
  4. Exception handler returns to main



System Timer – SysTick Overview

- ARMv7-M

System Timer – SysTick Overview

- ARMv7-M
  - Provides a simple 24-bit system timer
    - Reload on count == 0
    - Optionally cause SysTick interrupt on count == 0

System Timer – SysTick Overview

- ARMv7-M
  - Provides a simple 24-bit system timer
    - Reload on count == 0
    - Optionally cause SysTick interrupt on count == 0
  - Four registers (mapped in memory)
    - SYST_CSR: control and status register
    - SYST_RVR: reload value register
    - SYST_CVR: current value register
    - SYST_CALIB: calibration value register

System Timer – SysTick Overview

- ARMv7-M
  - Provides a simple 24-bit system timer
    - Reload on count == 0
    - Optionally cause SysTick interrupt on count == 0
  - Four registers (mapped in memory)
    - SYST_CSR: control and status register
    - SYST_RVR: reload value register
    - SYST_CVR: current value register
    - SYST_CALIB: calibration value register
  - Example usages:
    - As a virtual counter that indicated virtual time
    - Measuring elapsed time
    - Executing tasks periodically

System Timer – SysTick Registers

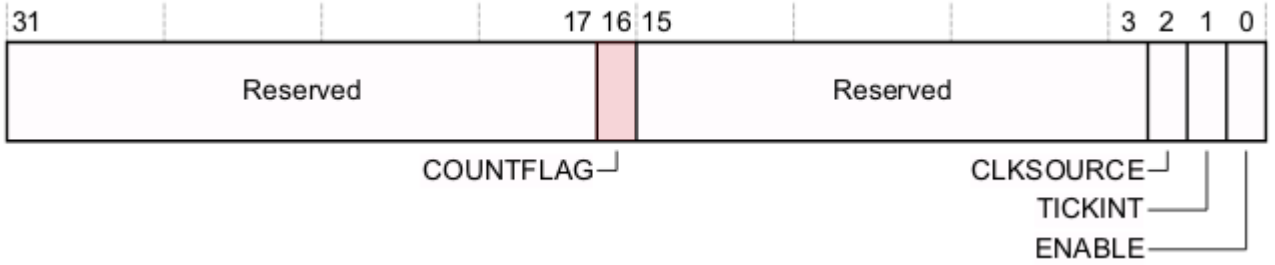
- ARMv7-M

Table B3-7 SysTick register summary

Address	Name	Type	Reset	Description
0xE000E010	SYST_CSR	RW	0x0000000x[a]	*SysTick Control and Status Register, SYST_CSR*
0xE000E014	SYST_RVR	RW	UNKNOWN	*SysTick Reload Value Register, SYST_RVR* on page B3-678
0xE000E018	SYST_CVR	RW	UNKNOWN	*SysTick Current Value Register, SYST_CVR* on page B3-678
0xE000E01C	SYST_CALIB	RO	IMP DEF	*SysTick Calibration value Register, SYST_CALIB* on page B3-679
0xE000E020-0xE000E0FC	-	-	-	Reserved

System Timer – SysTick Registers

- ARMv7-M
  - SysTick Control and Status Register, SYST_CSR



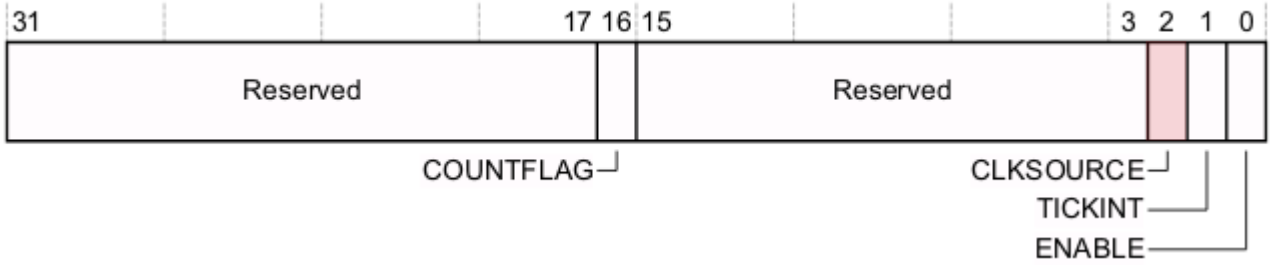
  - COUNTFLAG (read only)
    0 → Timer has not counted to 0
    1 → Timer has counted to 0

    – is set to 1 by a count transition from 1 to 0
    – is cleared to 0 by a SW read of this register, and by any write to the CVR.

    Note: Debugger reads do not clear the COUNTFLAG

System Timer – SysTick Registers

- ARMv7-M
  - SysTick Control and Status Register, SYST_CSR

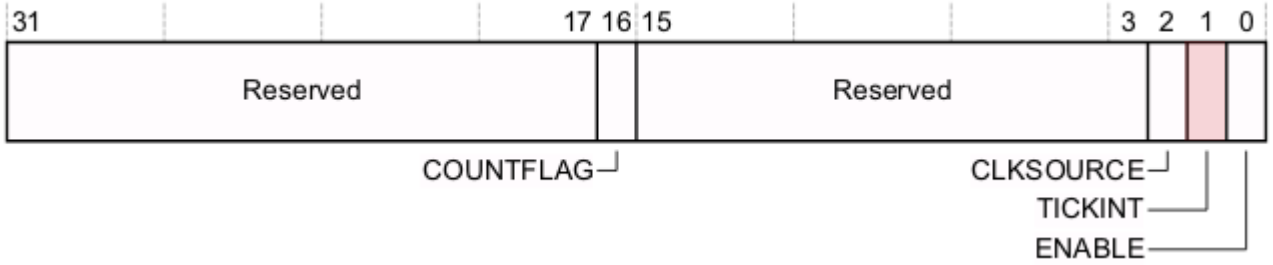


  - CLKSOURCE
    - 0 → uses the IMP DEF external reference clock
    - 1 → used the processor clock

    Note: if no external clock is provided, this bit is 1 and ignore writes

System Timer – SysTick Registers

- ARMv7-M
  - SysTick Control and Status Register, SYST_CSR



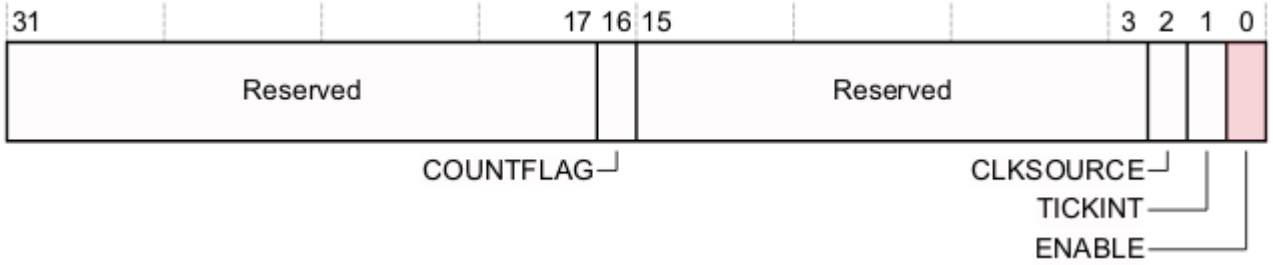
  - TICKINT

    0 → count to 0 does not affect the SysTick exception status

    1 → count to 0 changes the SysTick exception status to pending

System Timer – SysTick Registers

- ARMv7-M
  - SysTick Control and Status Register, SYST_CSR



  - ENABLE
    - 0 → counter is disabled
    - 1 → counter is operating

System Timer – SysTick Registers

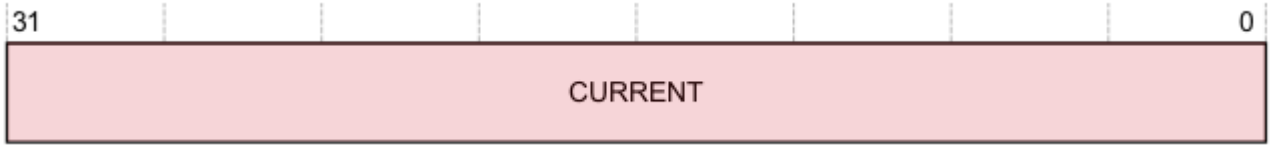
- ARMv7-M
  - SysTick Reload Value Register, SYST_RVR

  - RELOAD

    The value to load into the SYST_CVR when the counter reaches 0

System Timer – SysTick Registers

- ARMv7-M
  - SysTick Current Value Register, SYST_CVR



  - CURRENT

    This is the value of the counter at the time it is sampled

System Timer – SysTick Initialization

- ARMv7-M
  - The SysTick counter reload and current value are not initialized by hardware. This means the correct initialization sequence for the SysTick counter is:

System Timer – SysTick Initialization

- ARMv7-M
  - The SysTick counter reload and current value are not initialized by hardware. This means the correct initialization sequence for the SysTick counter is:

    1. Program reload value

System Timer – SysTick Initialization

- ARMv7-M
  - The SysTick counter reload and current value are not initialized by hardware. This means the correct initialization sequence for the SysTick counter is:

    1. Program reload value

    2. Clear current value

System Timer – SysTick Initialization

- ARMv7-M
  - The SysTick counter reload and current value are not initialized by hardware. This means the correct initialization sequence for the SysTick counter is:

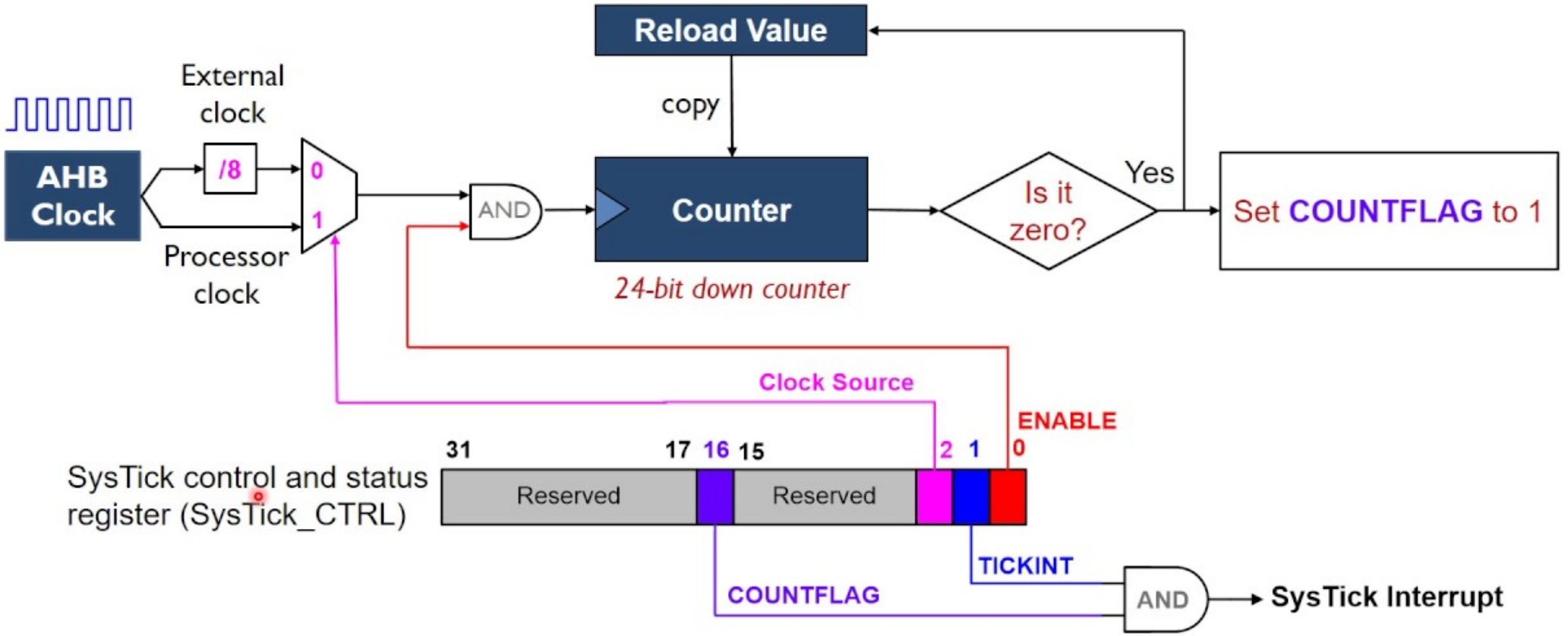
    1. Program reload value

    2. Clear current value

    3. Program Control and Status Register

System Timer – SysTick Operation

- ARMv7-M
  - Let's take a look at the SysTick Diagram



Memory System Architecture

- ARMv7-AR

Memory System Architecture

- ARMv7-AR
  - Virtual Memory System Architecture (VMSA)

Memory System Architecture

- ARMv7-AR
  - Virtual Memory System Architecture (VMSA)

Memory System Architecture

- ARMv7-AR
  - Virtual Memory System Architecture (VMSA)
    - Based on Memory Management Unit (MMU)

Memory System Architecture

- ARMv7-AR
  - Virtual Memory System Architecture (VMSA)
    - Based on Memory Management Unit (MMU)

Memory System Architecture

- ARMv7-AR
  - Virtual Memory System Architecture (VMSA)
    - Based on Memory Management Unit (MMU)
    - e.g. Cortex-A series
  - Protected Memory System Architecture (PMSA)
    - Based on Memory Protection Unit (MPU)

Memory System Architecture

- ARMv7-AR
  - Virtual Memory System Architecture (VMSA)
    - Based on Memory Management Unit (MMU)
    - e.g. Cortex-A series
  - Protected Memory System Architecture (PMSA)
    - Based on Memory Protection Unit (MPU)
    - e.g. Cortex-R series

Memory System Architecture

- ARMv7-AR
  - Virtual Memory System Architecture (VMSA)
    - Based on Memory Management Unit (MMU)
    - e.g. Cortex-A series

    Why?
  - Protected Memory System Architecture (PMSA)
    - Based on Memory Protection Unit (MPU)
    - e.g. Cortex-R series

Memory System Architecture

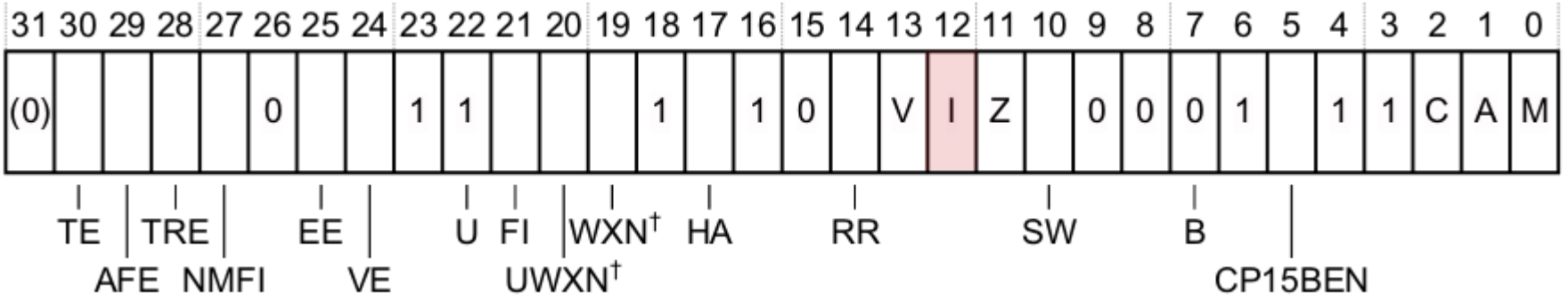
- ARMv7-AR
  - Virtual Memory System Architecture (VMSA)
    - Based on Memory Management Unit (MMU)
    - e.g. Cortex-A series

    Why?
  - Protected Memory System Architecture (PMSA)
    - Based on Memory Protection Unit (MPU)
    - e.g. Cortex-R series

    Why?

Memory System Architecture

- ARMv7-AR
  - The System Control Register (SCTLR)

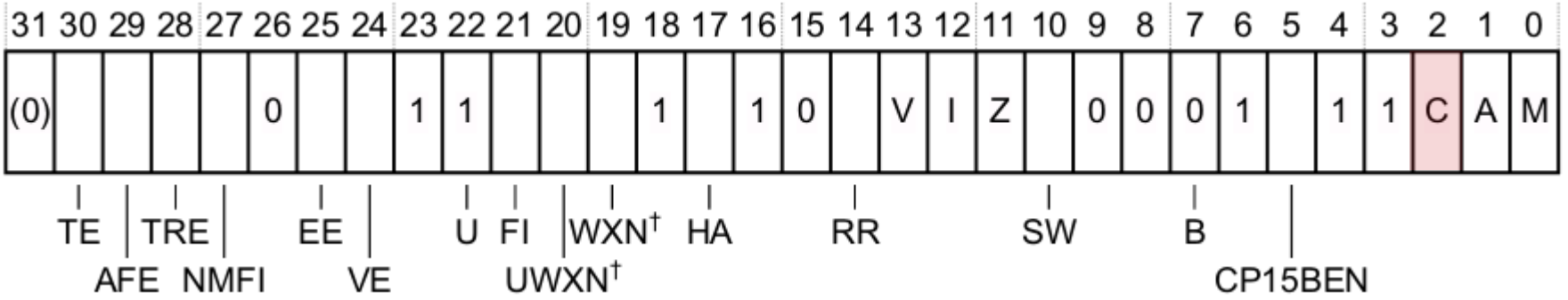


  – SCTLR.I: instruction caches enable

  0 → instruction cache disabled
  1 → instruction cache enabled

Memory System Architecture

- ARMv7-AR
  - The System Control Register (SCTLR)

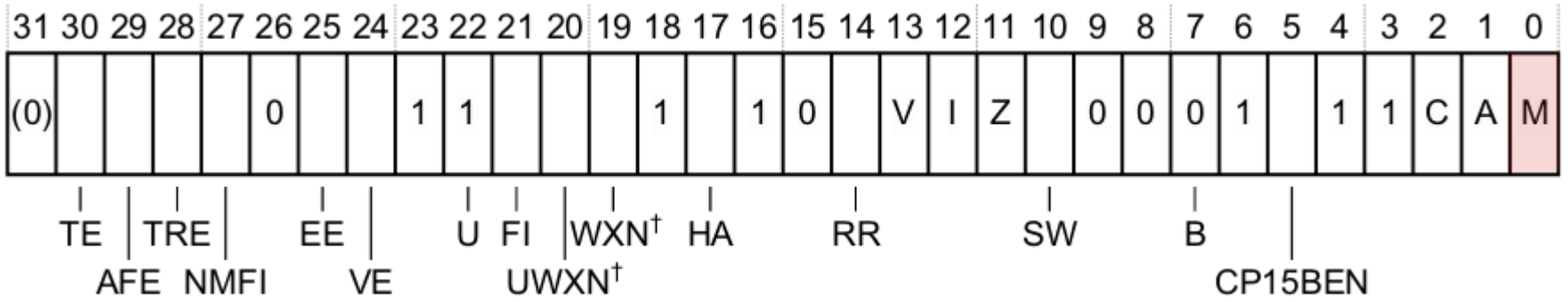


– SCTLR.C: data caches enable

0 → data cache disabled
1 → data cache enabled

Memory System Architecture

- ARMv7-AR
  - The System Control Register (SCTLR)



  – SCTLR.M: MMU enable

  0 → PL1&0 stage 1 MMU disabled
  1 → PL1&0 stage 1 MMU enabled

Note: Implementations that includes the Security Extensions, this bit is Banked between the Secure and Non-secure copies of the register.

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU
    - On Startup/Reset

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU
    - On Startup/Reset
      - No Security Extensions
        - SCTLR.M bit resets to 0 (MMU is disabled)

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU
    - On Startup/Reset
      - – No Security Extensions
        - – SCTLR.M bit resets to 0 (MMU is disabled)
      - – With Security Extensions
        - – SCTLR.M is banked (secure / non-secure)
        - – Only secure copy of SCTLR.M bit resets to 0
        - – Non-secure copy is UNKNOWN

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU (steps)

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU (steps)
    - – Disable caches and branch predictor
      Clear I, C and Z bits in SCTLR

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU (steps)
    - – Disable caches and branch predictor
      Clear I, C and Z bits in SCTLR
    - – Invalidate Everything

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU (steps)
    - – Disable caches and branch predictor
      Clear I, C and Z bits in SCTLR
    - – Invalidate Everything
      Invalidate instruction, data TLBs

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU (steps)
    - – Disable caches and branch predictor
      Clear I, C and Z bits in SCTLR
    - – Invalidate Everything
      Invalidate instruction, data TLBs
      Invalidate L1 instruction and data caches

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU (steps)
    - – Disable caches and branch predictor
      Clear I, C and Z bits in SCTLR
    - – Invalidate Everything
      Invalidate instruction, data TLBs
      Invalidate L1 instruction and data caches
      Invalidate branch predictor array

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU (steps)
    - – Disable caches and branch predictor
      Clear I, C and Z bits in SCTLR
    - – Invalidate Everything
      Invalidate instruction, data TLBs
      Invalidate L1 instruction and data caches
      Invalidate branch predictor array
    - – Set control registers

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU (steps)
    - – Disable caches and branch predictor
      Clear I, C and Z bits in SCTLR
    - – Invalidate Everything
      Invalidate instruction, data TLBs
      Invalidate L1 instruction and data caches
      Invalidate branch predictor array
    - – Set control registers
    - – Enable MMU

MMU – Memory Management Unit

- ARMv7-A
  - Enabling and disabling the MMU (steps)
    - – Disable caches and branch predictor
      Clear I, C and Z bits in SCTLR
    - – Invalidate Everything
      Invalidate instruction, data TLBs
      Invalidate L1 instruction and data caches
      Invalidate branch predictor array
    - – Set control registers
    - – Enable MMU
      Set SCTLR.M to enable the MMU

MPU – Memory Protection Unit

- ARMv7-AR
  - Enabling and disabling the MPU

MPU – Memory Protection Unit

- ARMv7-AR
  - Enabling and disabling the MPU
    - Software can use the SCTLR.M bit to enable and disable the MPU

MPU – Memory Protection Unit

- ARMv7-AR
  - Enabling and disabling the MPU
    - Software can use the SCTLR.M bit to enable and disable the MPU
    - On Startup/Reset

MPU – Memory Protection Unit

- ARMv7-AR
  - Enabling and disabling the MPU
    - Software can use the SCTLR.M bit to enable and disable the MPU
    - On Startup/Reset
      – SCTLR.M is cleared to 0 (MPU is disabled)

Caches and memory hierarchy

- ARMv7-AR

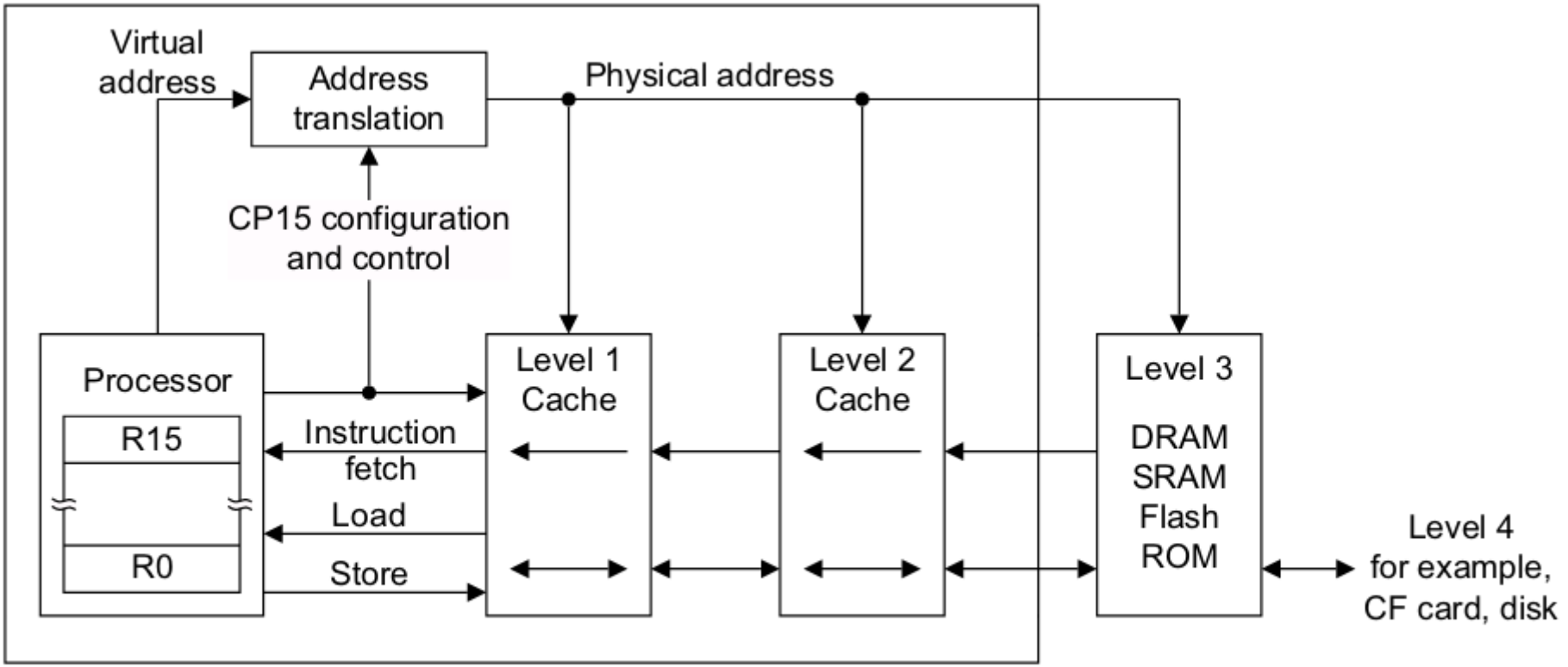


Figure A3-6 Multiple levels of cache in a memory hierarchy

Cache Initialization (behavior at reset)

- ARMv7-AR

Cache Initialization (behavior at reset)

- ARMv7-AR

Cache Initialization (behavior at reset)

- ARMv7-AR
  - All caches are disabled at reset

Cache Initialization (behavior at reset)

- ARMv7-AR
  - All caches are disabled at reset
  - Cache initialization → IMPDEF
    - – Initialization routine to invalidate its storage array before it is enabled
    - – When it is enabled, the cache state if UNPREDICTABLE if the appropriate initialization routine has not been performed

Cache Initialization (behavior at reset)

- ARMv7-AR

Cache Initialization (behavior at reset)

- ARMv7-AR
  - Enabling and disabling Cache

Cache Initialization (behavior at reset)

- ARMv7-AR

  - Enabling and disabling Cache
    - Instruction cache

      SCTLR.I → instruction cache enable

    - Cache enable

      SCTLR.C → data cache enable

Cache Initialization (behavior at reset)

- ARMv7-AR

  - Enabling and disabling Cache
    - Instruction cache
      SCTLR.I → instruction cache enable
    - Cache enable
      SCTLR.C → data cache enable

Note: The initialization routine is implementation defined

ARMv7-AR vs ARMv7-M

- The fundamental differences in ARMv7-M
  - No ARM instruction set support (Thumb only)
  - Only two operating modes
    - Thread mode and Handler Mode
  - Co-processors are not supported by default
    - except co-processors 10 and 11 (FP extensions)
  - The interrupt controller (NVIC) is part of the processor
  - NMI (Non-Maskable Interrupt)
  - State is automatically saved/restored on exception entry/return
  - The vector table contains address, not instructions

ARMv7-AR vs ARMv7-M

- The fundamental differences in ARMv7-AR
  - Advanced SIMD extension (branded as NEON™)
  - Performance Monitors Extensions
    – For advanced debugging/profiling
  - Virtualization Extensions (for virtual platform support)

- The ARMv7-A profile also provides the Security Extensions (branded as TrustZone®)

References

- ARMv7-M Architecture Reference Manual
- ARM Architecture Reference Manual: ARMv7-A and ARMv7-R edition
- ARM University Program Material
- https://pt.slideshare.net/GauravVerma3/arm-cortex-processor-compatibility-mode

ARM: an overview from the Operating System perspective

César Huegel Richa

huegel@lisha.ufsc.br

Software/Hardware Integration Laboratory (LISHA)
Federal University of Santa Catarina (UFSC)

Florianópolis, Santa Catarina, Brazil
April, 2018