Introduction to ARM Cortex-M Processors

CDAC ACTS, Pune

DESIGNING FOR THE FUTURE

Cortex M3 @ S/W Developer point of View

- Programming Model
- How exceptions are handled
- The Memory Map
- Peripheral Interfacing
- How to use software driver libraries from Microcontroller Vendor.
- CMSIS Core API's
- STM32CUBE Libraries

Agenda

What are the ARM Cortex M Processors?
- The CortexM3 and M4 Processors
- The Cortex-M Processor Family

Advantage of the Cortex-M Processors
- Low Power
- Performance
- Energy Efficiency
- Code Density
- Interrupts
- Ease of use, C friendly
- Scalability

Application of the ARM Cortex-M Processor

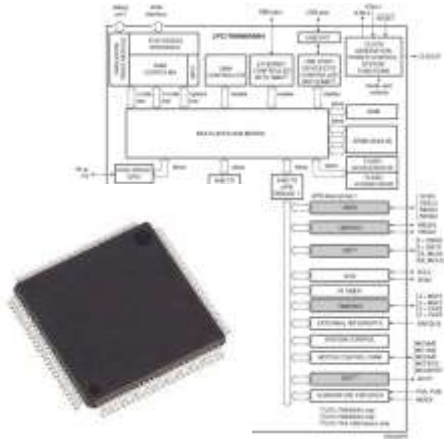
Background and History
- ARM processor evolution
- Architecture versions and Thumb ISA

Why ARM here ???

- ARM is one of the most licensed and thus widespread processor cores in the world
- Used especially in portable devices due to low power consumption and reasonable performance
- Several interesting extensions available like Thumb instruction set and Jazelle Java machine

What's Happening in Microcontrollers?

- Microcontrollers are getting **cheap**
  - 32-bit ARM Cortex-M3 Microcontrollers @ INR 1200
  - Some microcontrollers sell for as little as INR 400
- Microcontrollers are getting **powerful**
  - Lots of processing, memory, I/O in one package
  - Floating-point is even available in some!
- Microcontrollers are getting **interactive**
  - Internet connectivity, new sensors and actuators
  - LCD and display controllers are common



- **Creates new opportunities for microcontrollers**

ARM History

- ARM – Acorn RISC Machine(1983–1985)
  - Acorn Computers Limited, Cambridge, England
- ARM – Advanced RISC Machine 1990
  - ARM Limited, 1990
  - ARM has been licensed to many semiconductor manufacturers

ARM History

- Key component of many 32 – bit embedded systems
- Portable Consumer devices
- ARM1 prototype in 1985
- One of the ARM's most successful cores is the ARM7TDMI,provides high code density and low power consumption.
- **ARM7 + 16 bit Thumb + JTAG Debug + fast Multiplier + enhanced ICE**

Advanced RISC Machines

- ARM Core uses a _____ architecture
- ARM is Physical hardware design company.
- ARM licenses its cores out and other companies make processors based on its cores

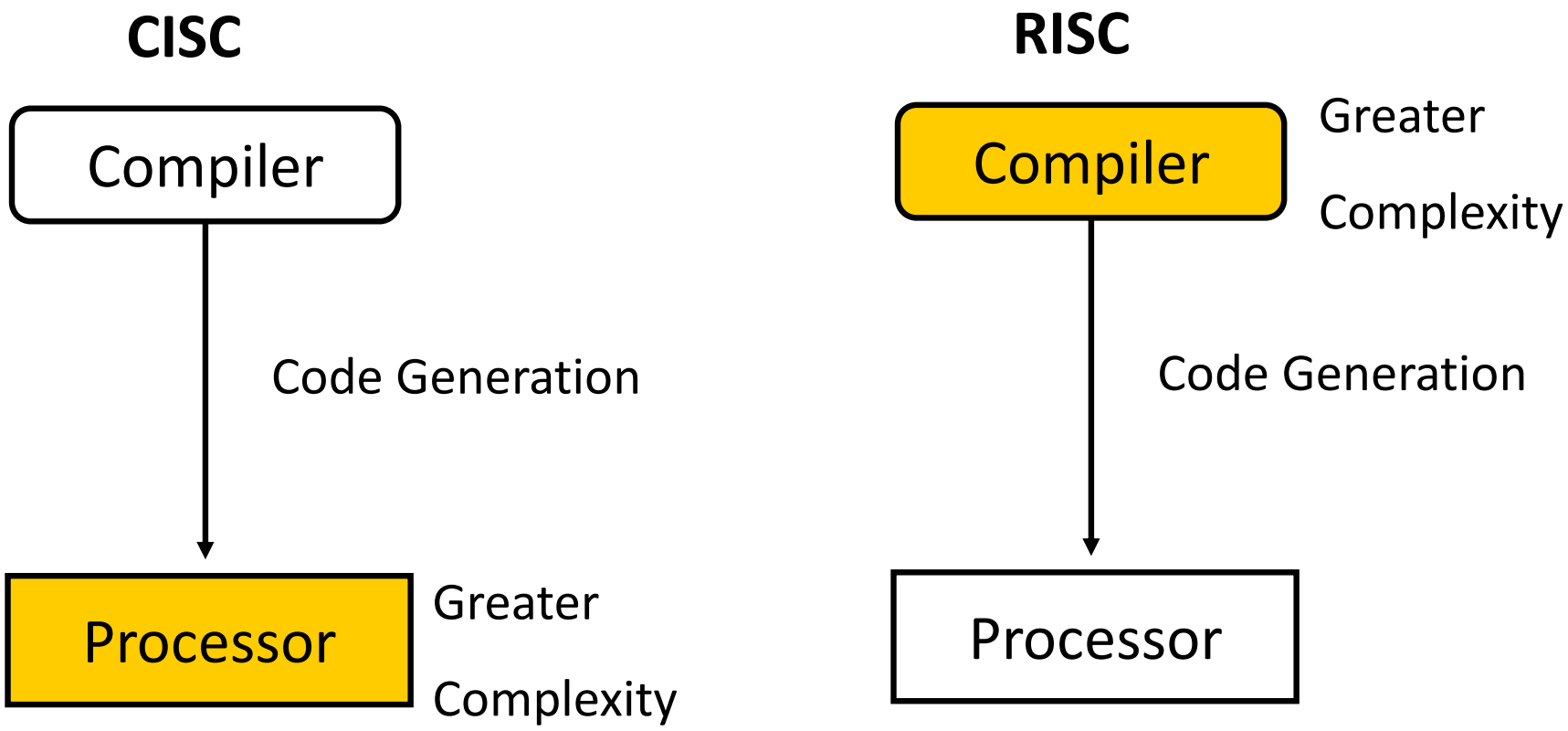
RISC vs. CISC Architecture

RISC	CISC
Fixed width instructions	Variable length instructions
Few formats of instructions	Several formats of instructions
Load/Store Architecture	Memory values can be used as operands in instructions
Large Register bank	Small Register Bank
Instructions are pipelinable	Pipelining is Complex

RISC

- Advantage(s)
  - A Smaller Die Size
  - A Shorter Development Time
  - Higher Performance (Bit Tricky)
- Disadvantage
  - Generally poor code density (Fixed Length Instruction)

CISC vs. RISC


CISC
Compiler
Code Generation
Processor
Greater
Complexity
RISC
Compiler
Greater
Complexity
Code Generation
Processor

Features used from RISC

- A Load/Store Architecture
- Fixed Length 32-bit Instructions
- 3- Address Instruction Formats

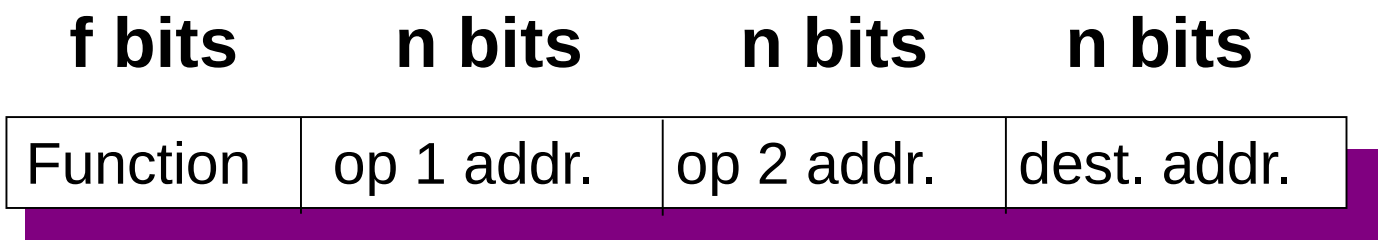
What am I going to get from this course?

- Programming Micro-controllers using ‘C'
- Learn about embedded software development and debugging using STM Cube IDE
- Learn about Mixed ‘C’ and Assembly Coding
- Demystifying Memory, Bus interfaces, NVIC, Exception handling with lots of animation
- Low level register Programming for interrupts, System Exceptions, Setting Priorities, Preemption etc.
- Learn writing IRQ handlers , IRQ numbers, NVIC and many more
- Learn about OS related features like SVC, SysTick, PendSv and many more

Load Store Architecture

- Memory can be accessed only through two dedicated instructions
  - LDR ; move word from memory to register
  - STR ; move word from register to memory
- All other instructions have to work on registers only.

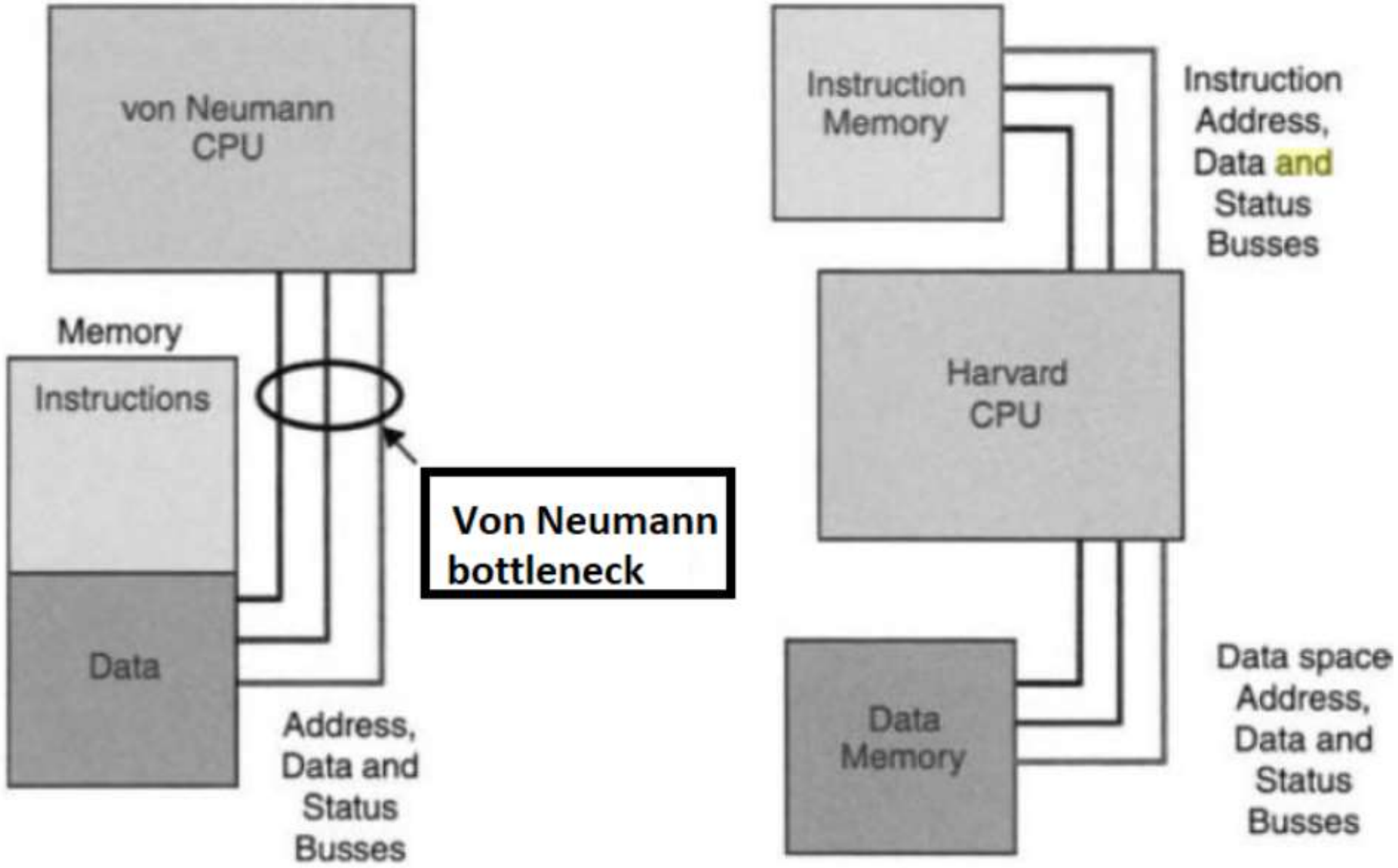
3 Address Instruction Format



Example

Add d, s1, s2 ; d =s1+s2

Von Neumann vs Harvard Architecture



What are ARM Cortex M processors?

The Cortex-M3 and Cortex-M4 are processors designed by ARM. The Cortex-M3 processor was the first of the Cortex generation of processors, released by ARM in 2005 (silicon products released in 2006). The Cortex-M4 processor was released in 2010 (released products also in 2010).

The Cortex-M3 and Cortex-M4 processors use a 32-bit architecture.

Internal registers in the register bank, the data path, and the bus interfaces are all 32 bits wide. The Instruction Set Architecture (ISA) in the Cortex-M processors is called the Thumb ISA and is based on Thumb-2 Technology which supports a mixture of 16-bit and 32-bit instructions.

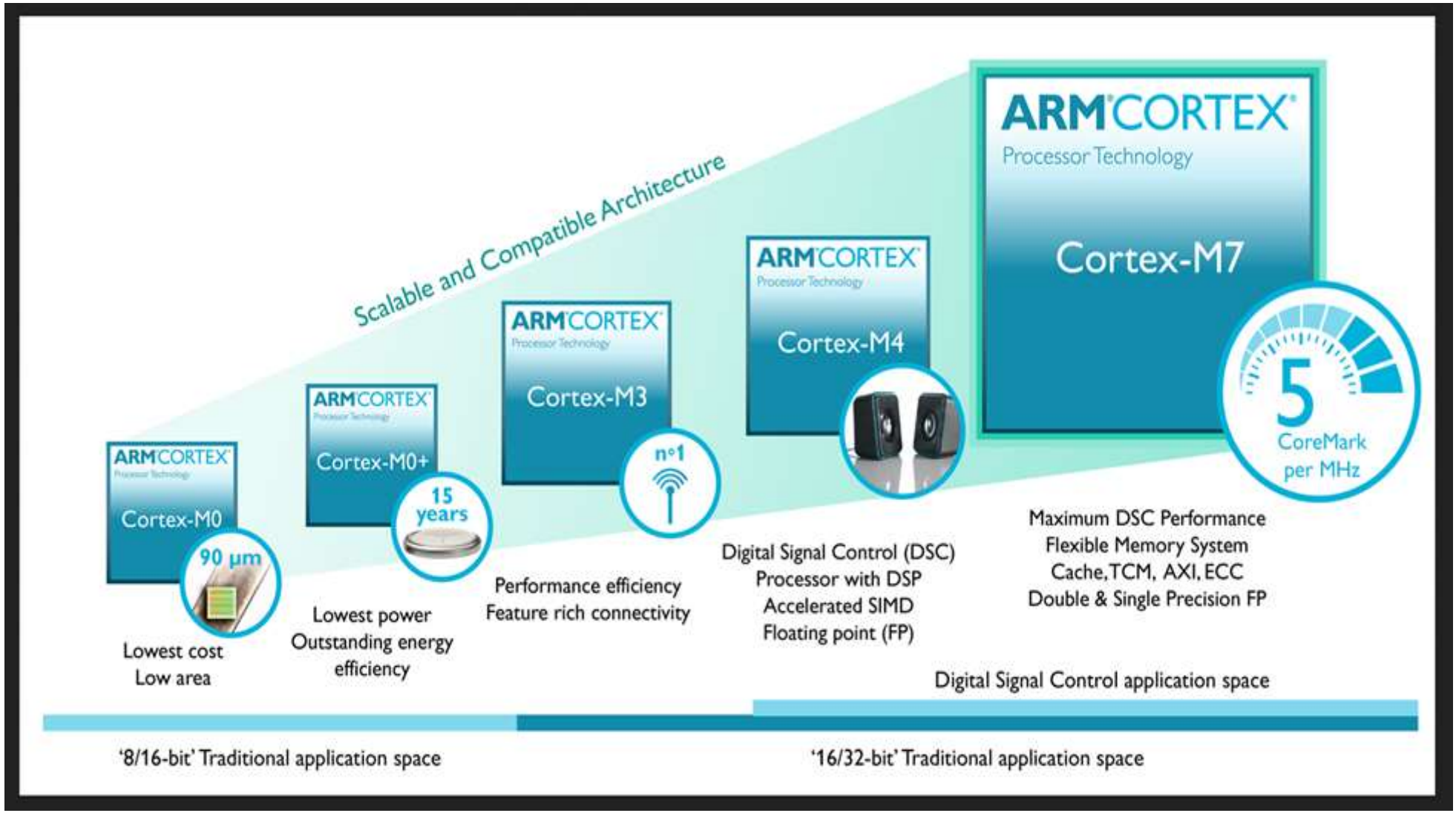
ARM Cortex-M3 Microcontroller

- 18 x 32-bit registers
- Excellent compiler target
- Reduced pin count requirements
- Efficient interrupt handling
- Power management
- Efficient debug and development support features
  - Breakpoints, Watchpoints,
    - Flash Patch support,
  - Instruction Trace
- Strong OS support
  - User/Supervisor model
  - OS support features
- Designed to be fully programmed in C (even reset, interrupts and exceptions)

ARM Cortex-M3 Microcontroller

- ARMv7M Architecture
- No Cache - No MMU
- Debug is optimized for microcontroller applications
- Vector table contains addresses, not instructions1
- DIV instruction
- Interrupts automatically save/restore state
- Exceptions programmed in C (No Coprocessor 15 - All registers are memory-mapped)
- Interrupt controller is part of Cortex-M3 macrocell
- Fixed memory map
- Bit-banding
- Non-Maskable Interrupt (NMI)
- Only one processor status reg
- Thumb-2 processing core
  - Mix of 16 and 32 bit instructions for very high code density
  - Gives complete Thumb compatibility

The Cortex M Processor Family



Advantage of Cortex –M Processor

- Low Power

  Currently, many Cortex-M microcontrollers have power consumption of less than 200 mA/MHz, with some of them well under 100 mA/MHz. In addition, the Cortex-M processors also include support for sleep mode features and can be used with various advanced ultra-low power design technologies.

- Performance

  The Cortex-M3 and Cortex-M4 processors can deliver over 3 CoreMark/MHz & 1.25 DMIPS

- Energy Efficiency
- Code Density
- Interrupts
- Ease of use, C friendly

Features Cont.

- Scalability
- Debug Friendly
- OS Support
- Versatile system features
    - Bit Banding
    - MPU
- Software portability and reusability
    - CMSIS

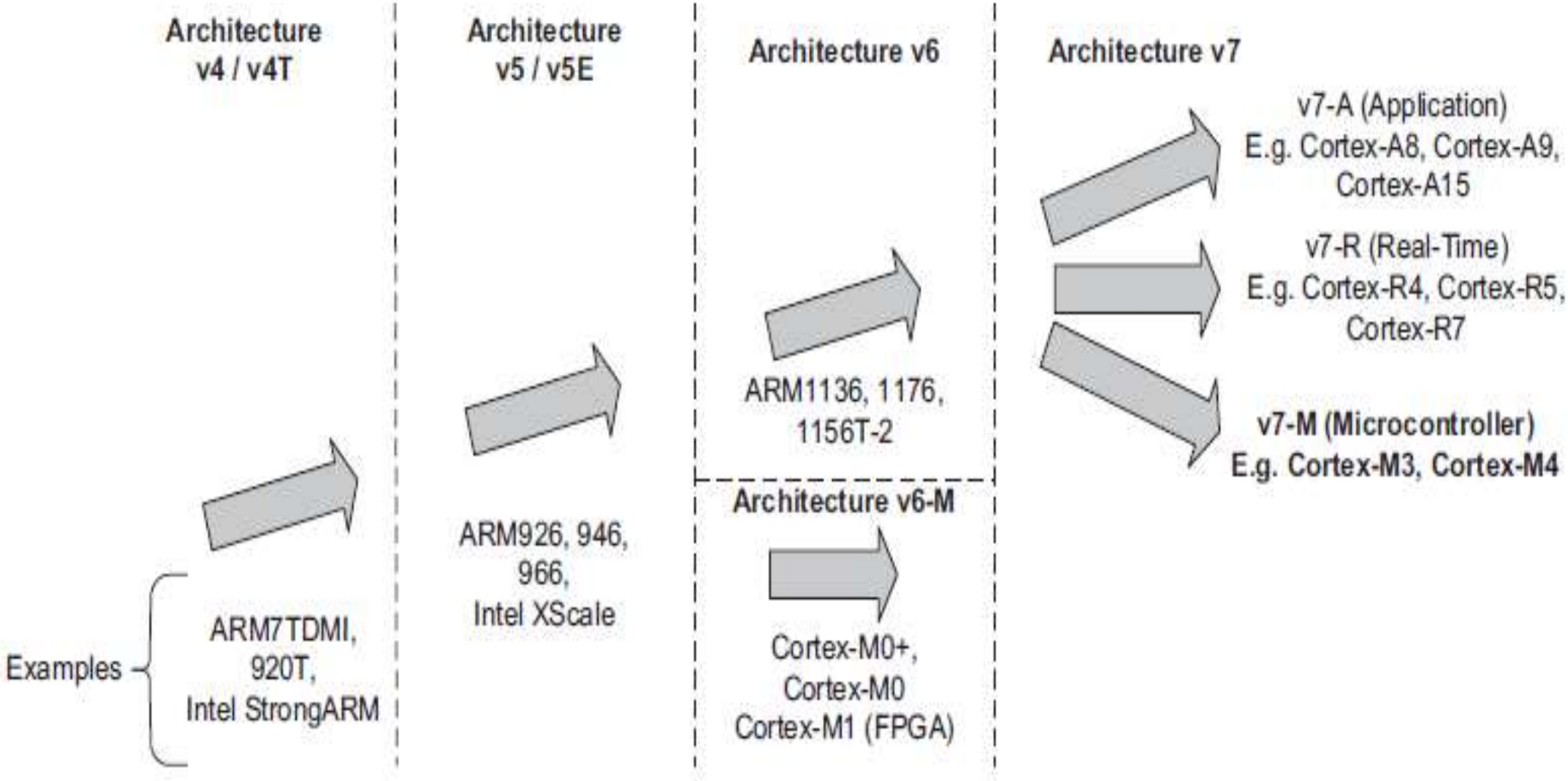
Application of Cortex M Processors

- Microcontrollers
- Automotive
- Industrial Control
- Consumer Products
- DSP

Architecture Version and Thumb ISA

- The successful ARM7TDMI is based on the architecture version ARMv4T (The “T” means Thumb instruction support).
- Note that architecture version numbers are independent of processor names.
- The Architecture version 7 is divided into three profiles
  - Cortex-A Processors: ARMv7-A Architecture
  - Cortex-R Processors: ARMv7-R Architecture
  - **Cortex-M Processors: ARMv7-M & ARMv6-M Architectures**

ISA Enhancement and Evolution



Introduction to Embedded Software Development

CDAC ACTS, Pune

DESIGNING FOR THE FUTURE

What are inside typical ARM µC.

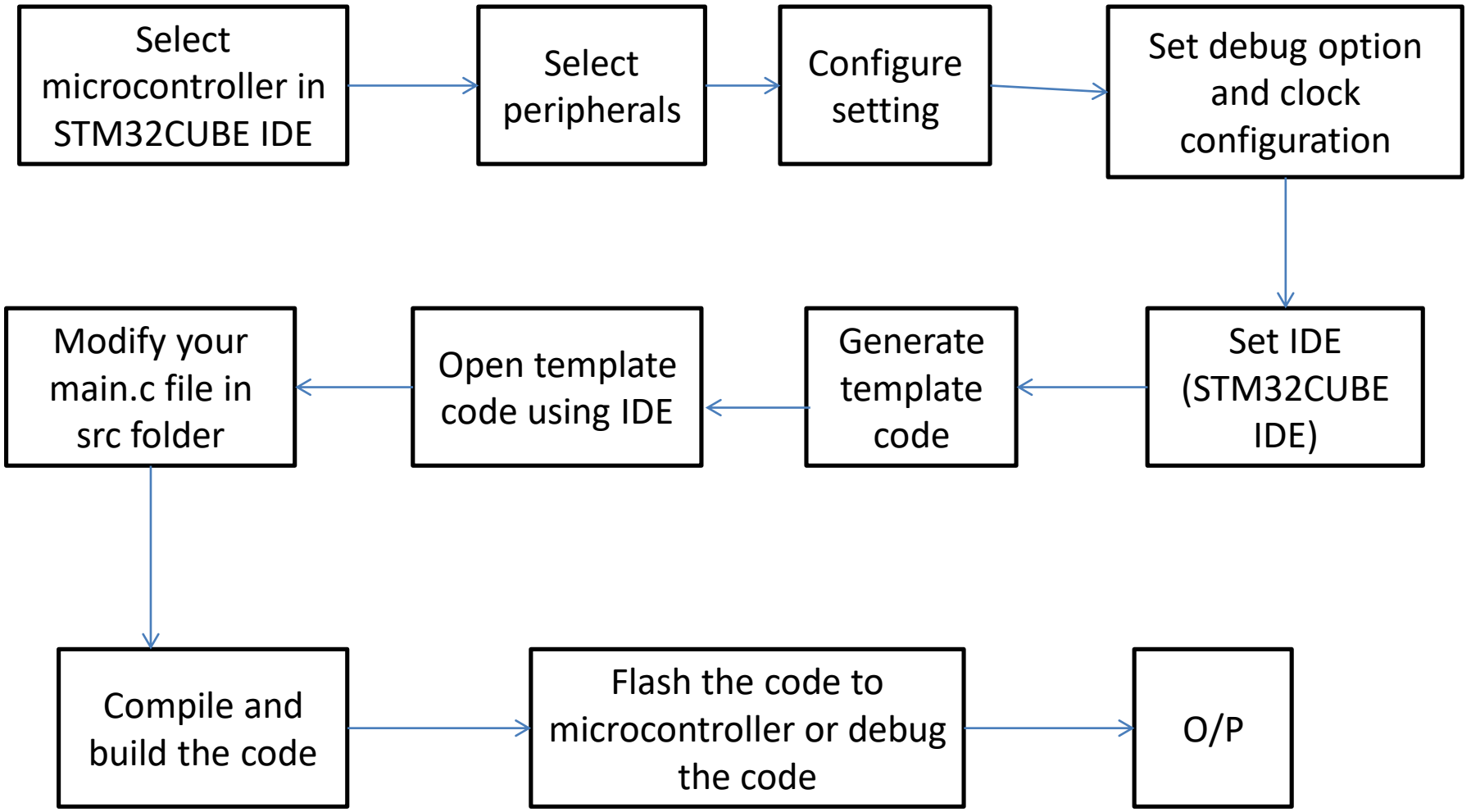
In many microcontrollers, the processor takes less than 10% of the silicon area, and the rest of the silicon die is occupied by other components such as:

- Program memory (e.g., flash memory)
- SRAM
- Peripherals
- Internal bus infrastructure
- Clock generator (including Phase Locked Loop), reset generator, and distribution network for these signals
- Voltage regulator and power control circuits
- Other analog components (e.g., ADC, DAC, voltage reference circuits)
- I/O pads

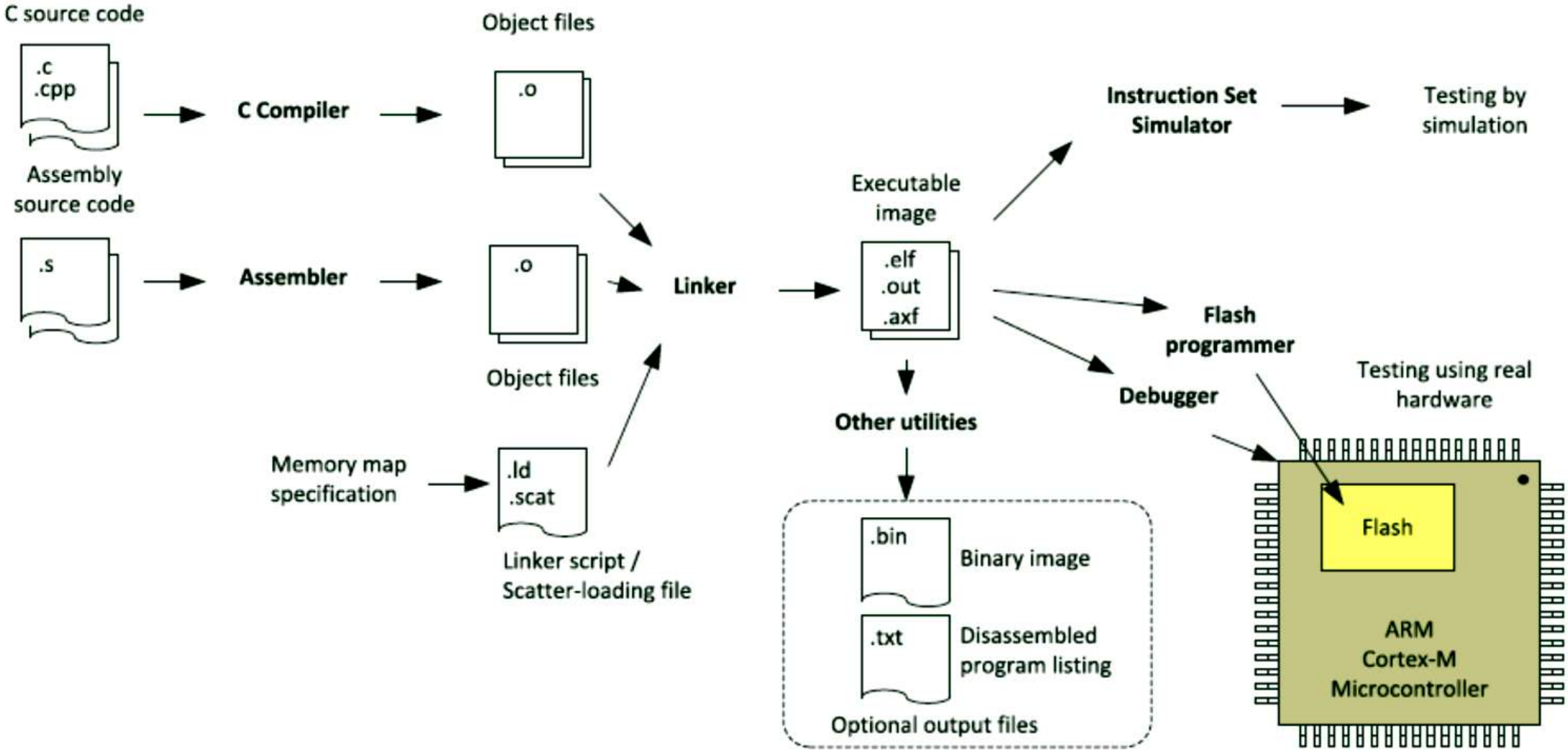
What you need to start

- Development suites
  - STM32CubeIDE
- Development Board
  - STM32F4 Discovery Board- IoT
- Real Time Variable Viewer
  - STM32CUBE Monitor
- Debug Adaptor
  - STLinkv2 -IoT
- Documents and other Resources
  - Link to Resources

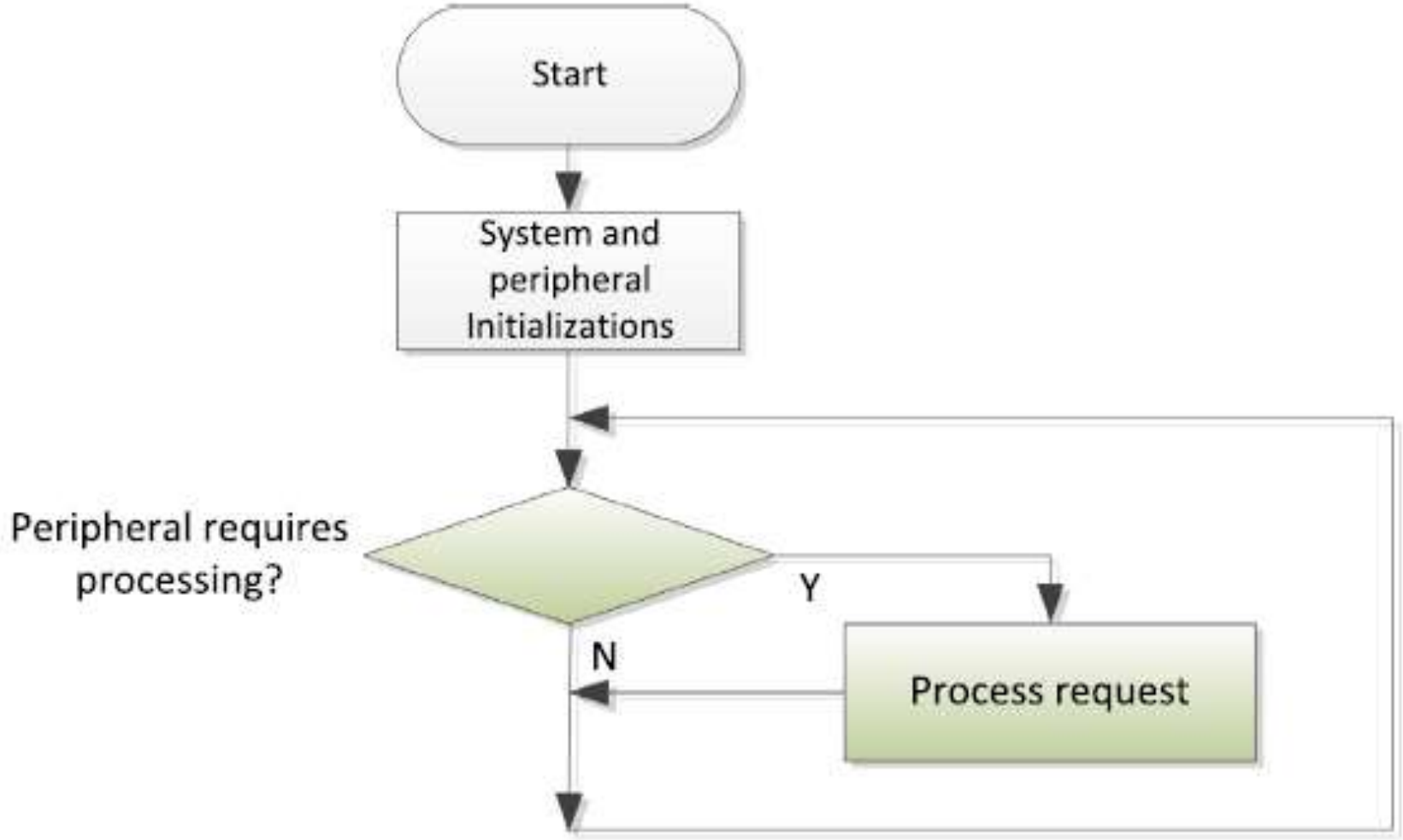
Software Development Flow



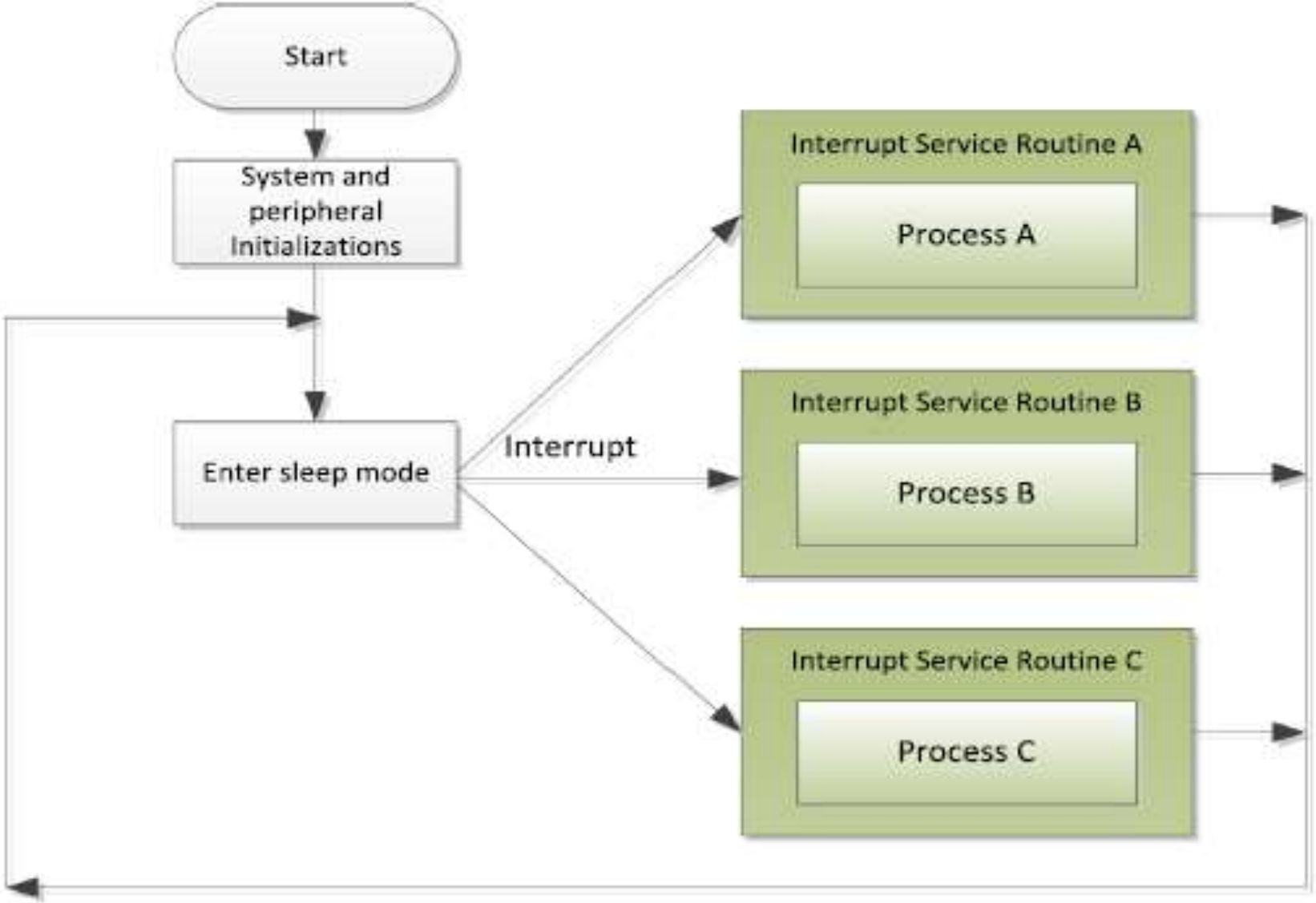
Software Compiling Flow



Polling Flow



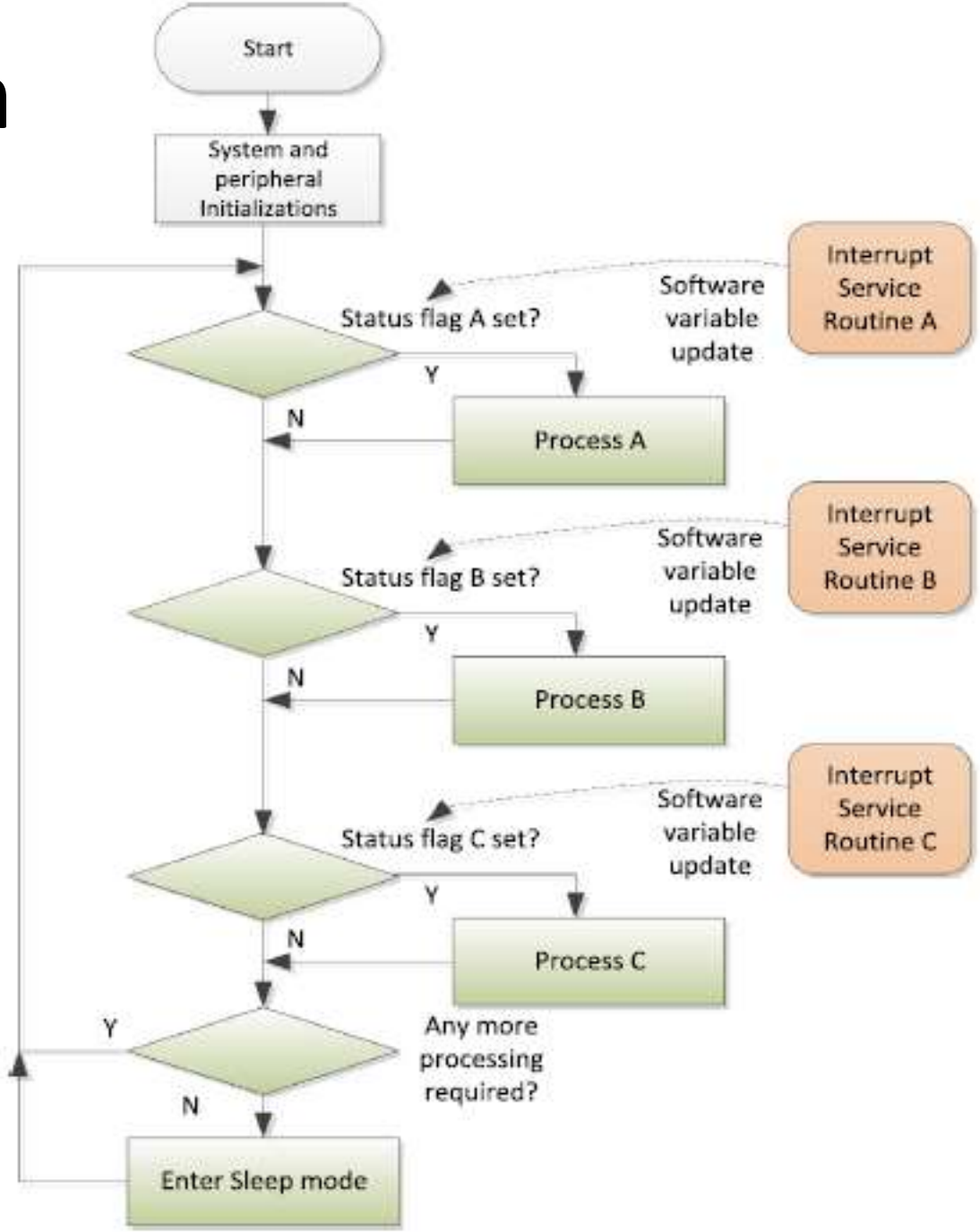
Interrupt Driven



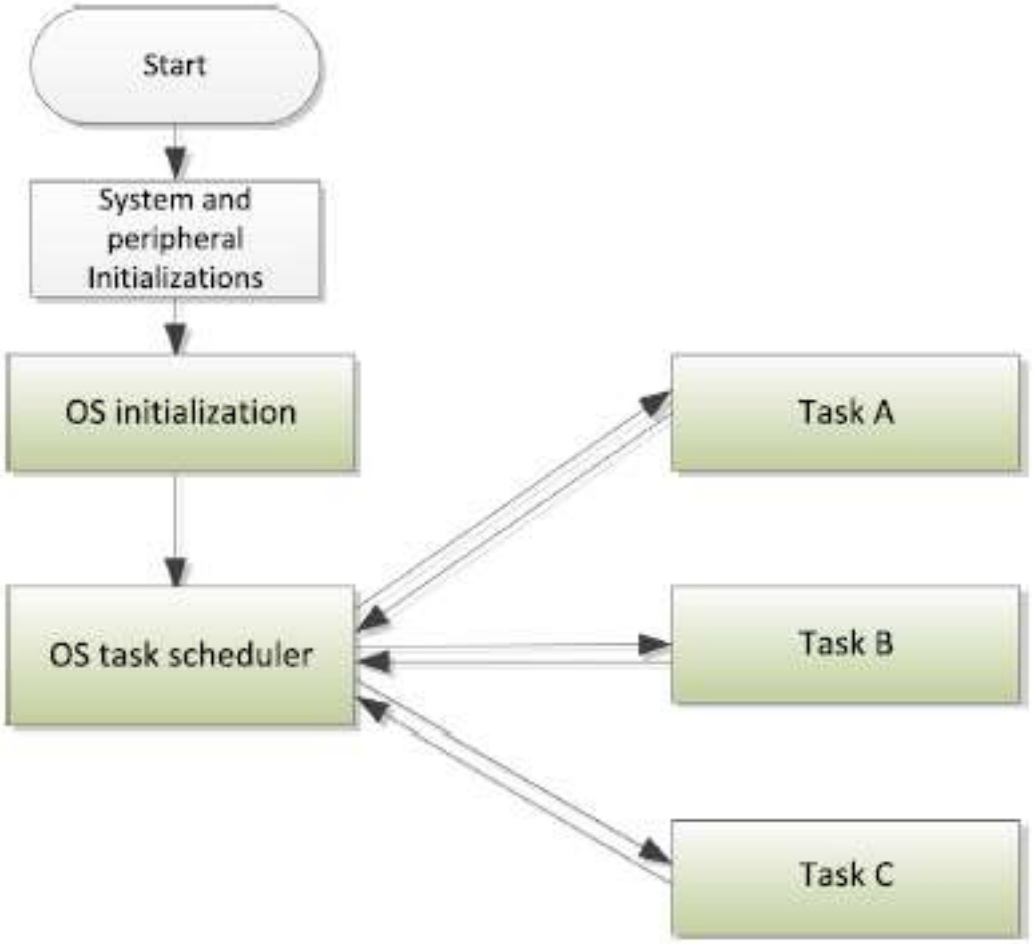
Multi Tasking System

In these applications, a Real-Time Operating System(RTOS) can be used to handle the task scheduling.

An RTOS allows multiple processes to be executed concurrently, by dividing the processor's time into time slots and allocating the time slots to the processes that require services.

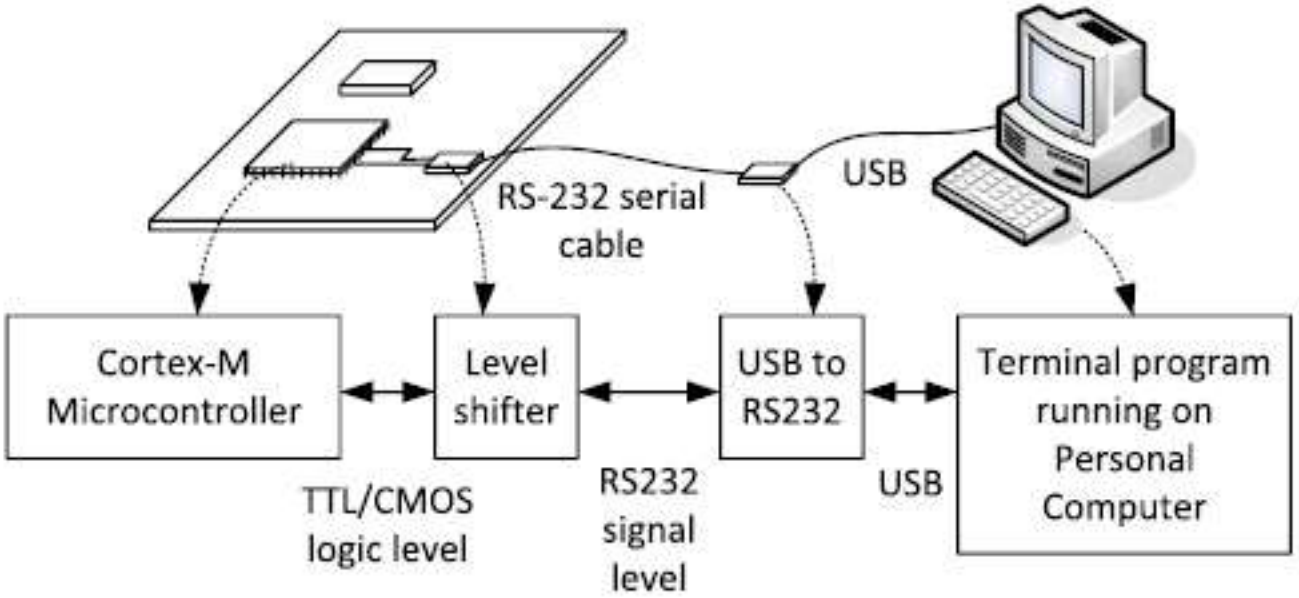


RTOS to Handle Multiple Task



ARM Data Size Definition & Interfaces

Terms	Size
Byte	8-bit
Half word	16-bit
Word	32-bit
Double word	64-bit



Technical Overview

CDAC ACTS, Pune

DESIGNING FOR THE FUTURE

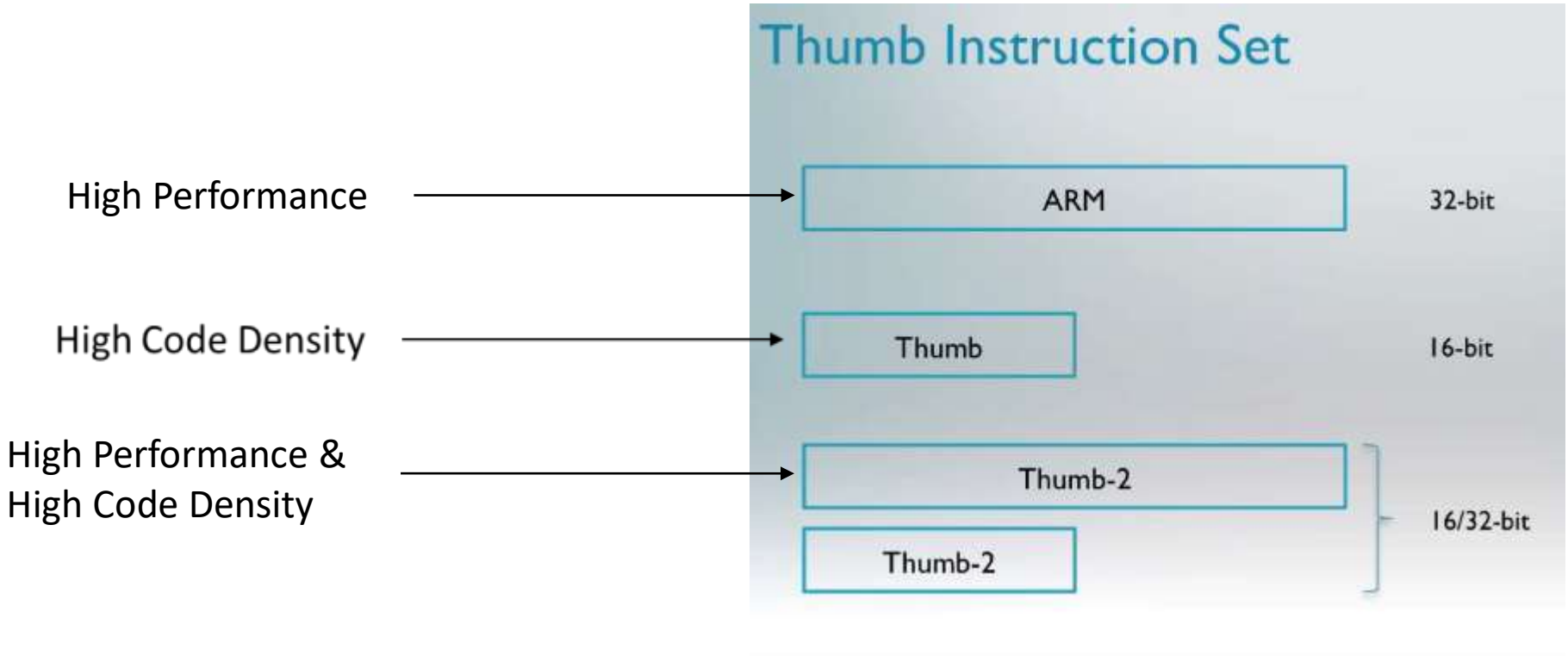
Processor type and Architecture

- All the ARM Cortex-M processors are 32-bit RISC (Reduced Instruction Set Computing) processors. They have:
- 32-bit registers
- 32-bit internal data path
- 32-bit bus interface
- 3 Stage Pipelining (FDE)
- **Harvard Bus Arch**.(Simul. Instrn. Fetch & Data Access)
- 32-bit Addressing which allow 4GB address space
- Load and Store Arch.

(LDM – load data from memory & STM – store data to memory)

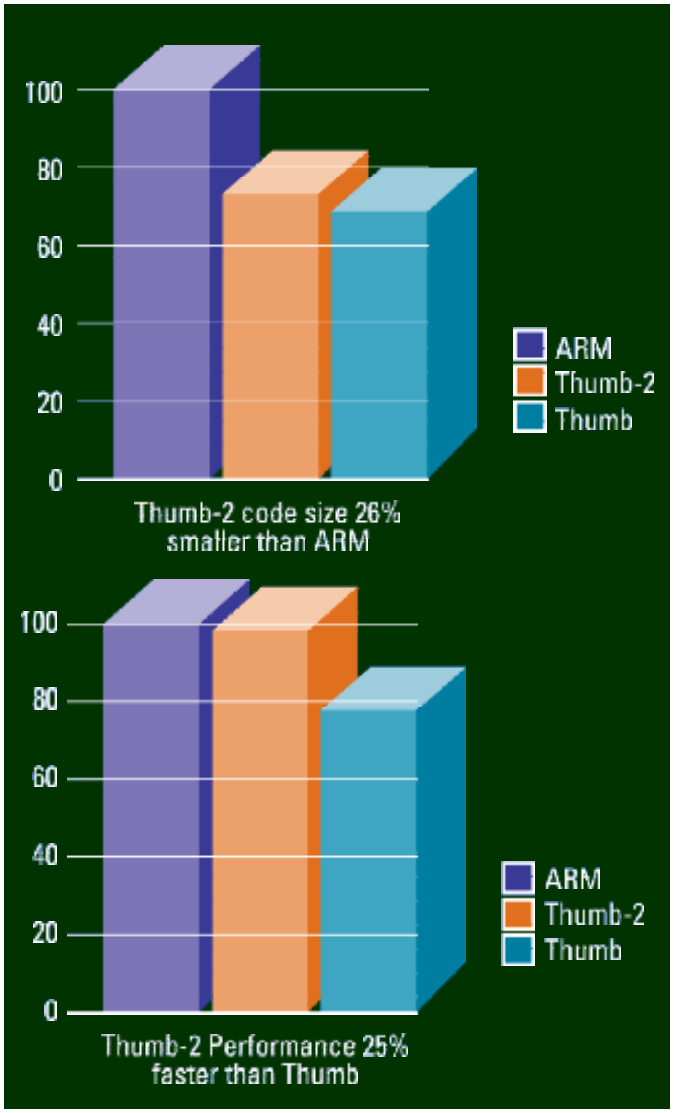
- Processor Arch. ARMv7M

Instruction Set - Thumb2



The Thumb-2 instruction set

- Variable-length instructions
  - ARM instructions are a fixed length of 32 bits
  - Thumb instructions are a fixed length of 16 bits
  - Thumb-2 instructions can be either 16-bit or 32-bit
- Thumb-2 gives approximately **26%** improvement in code density over ARM
- Thumb-2 gives approximately 25% improvement in performance over Thumb



Code density: to achieve the same tasks, you need a smaller program size.

- reduced cost
- Reduced power consumption
- use a microcontroller with smaller flash memory size

ARM and Thumb Mode Switching

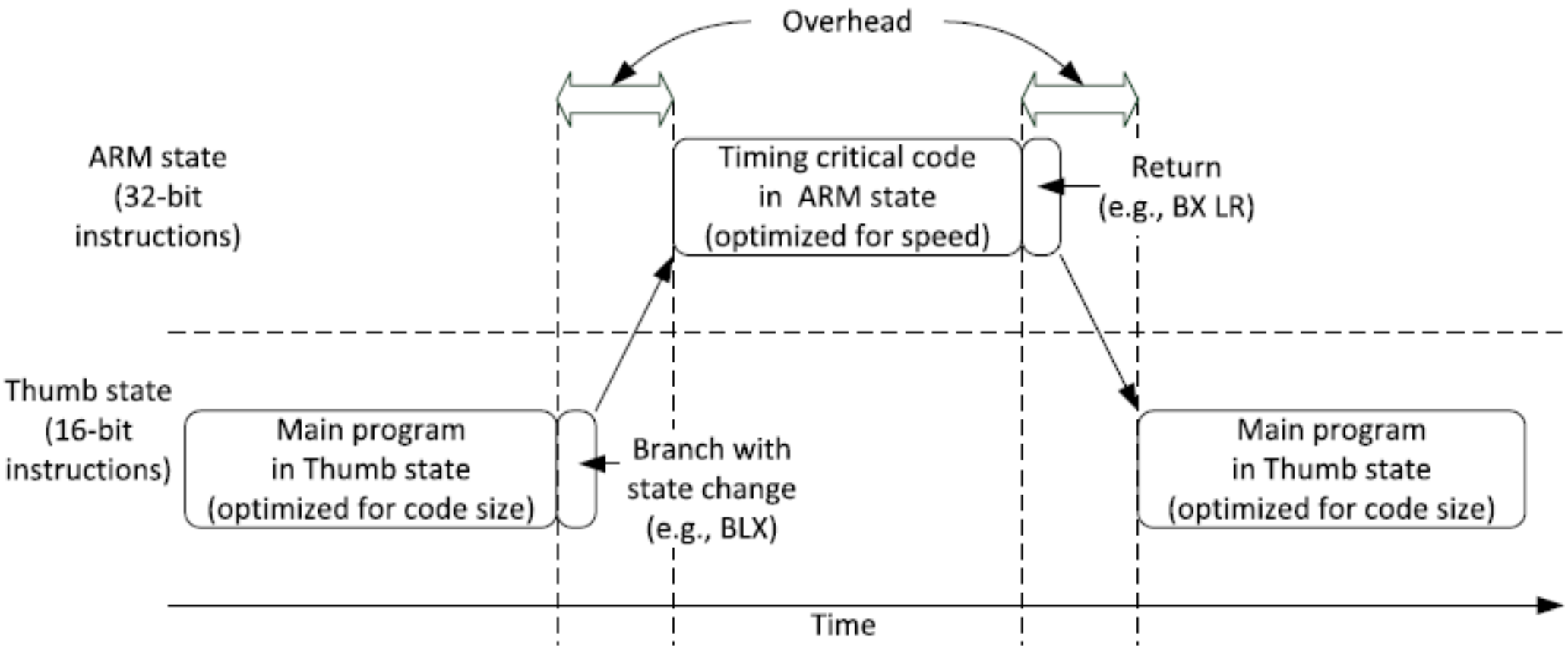


FIGURE 3.1

Switching between ARM code and Thumb code in class ARM processors such as the ARM7TDMI

Thumb2 No Switching Req.

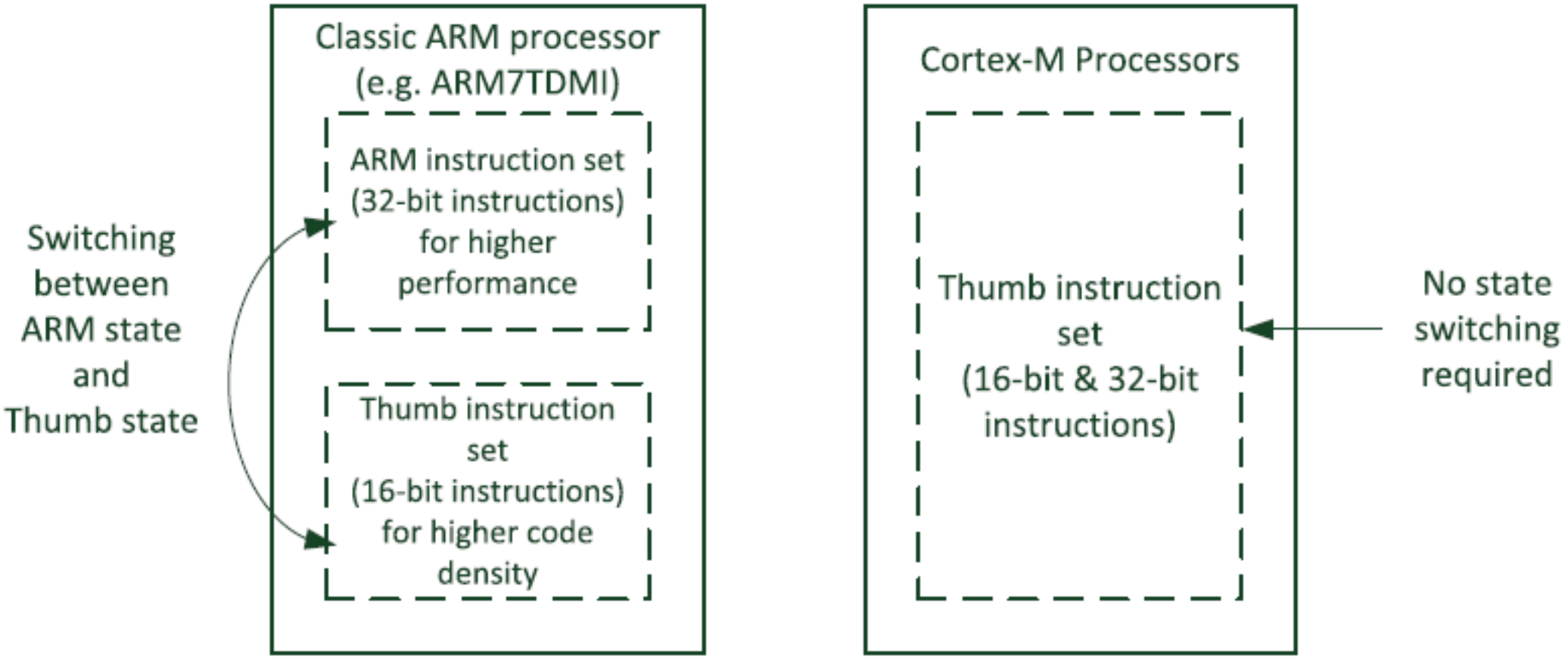
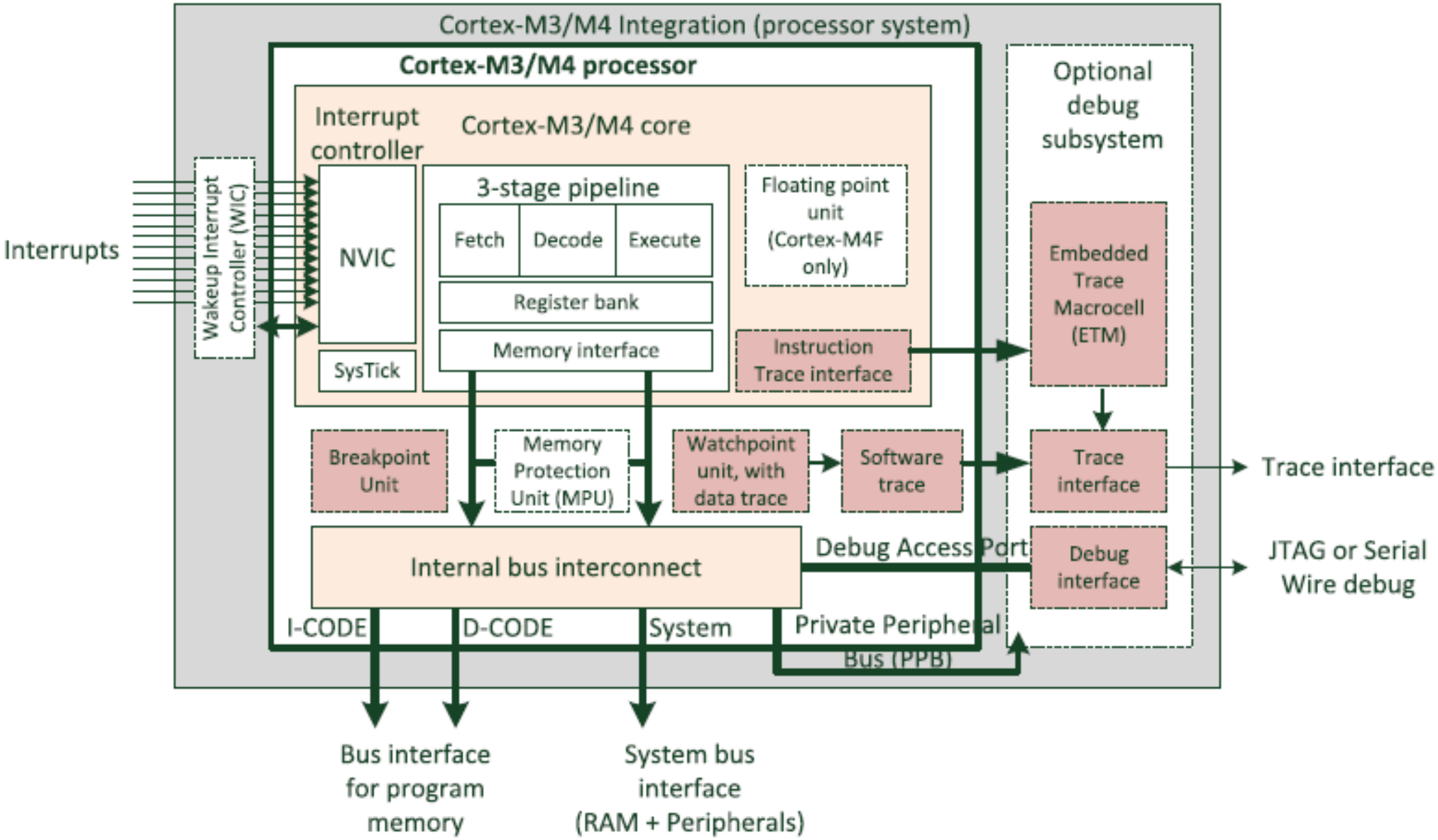


FIGURE 3.2

Instruction set comparison between Cortex-M processors and ARM7TDMI

Block Diagram

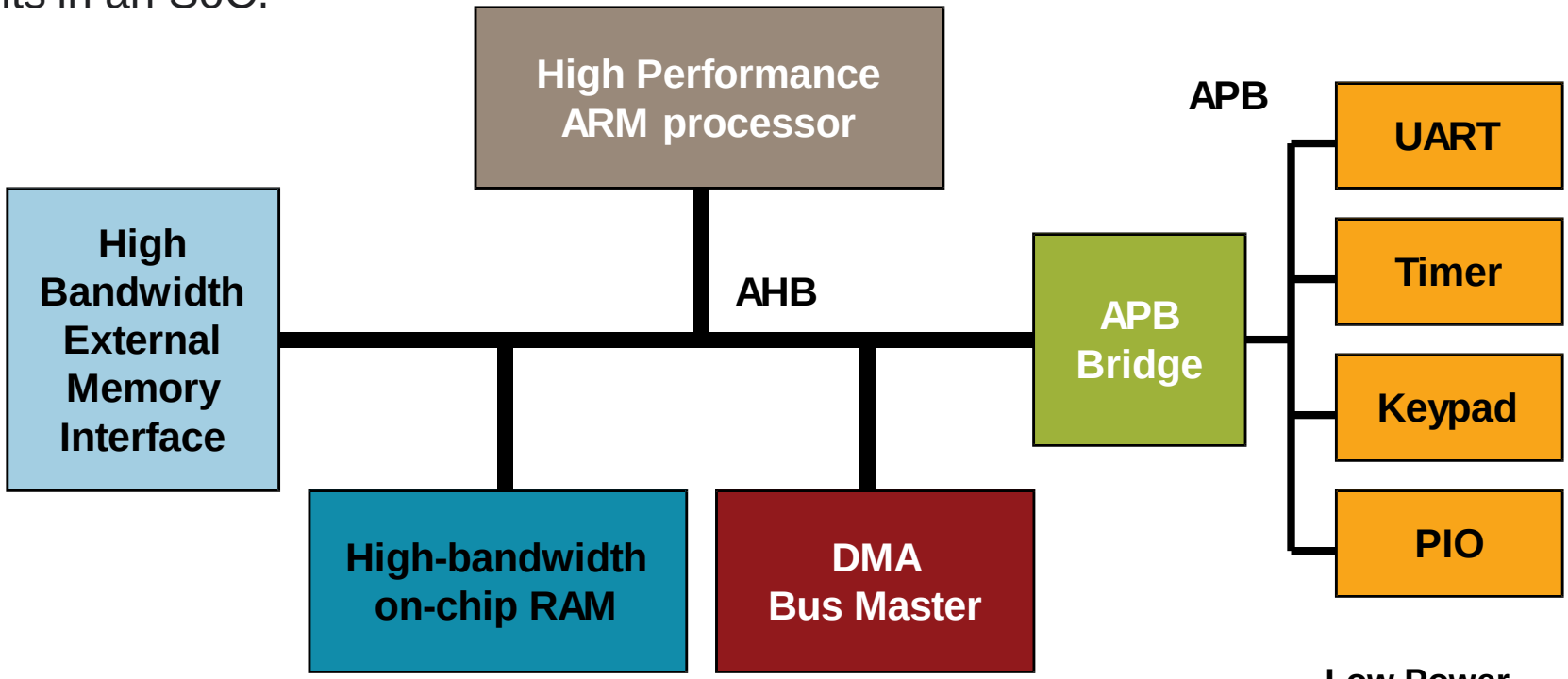


Various Bus Interfaces on Cortex M3

Bus Interface	Descriptions
I-CODE	Primarily for program memory: Instruction fetch and vector fetch for address 0x0 to 0x1FFFFFFF. Based on AMBA 3.0 AHB Lite bus protocol.
D-CODE	Primarily for program memory: Data and debugger accesses for address 0x0 to 0x1FFFFFFF. Based on AMBA 3.0 AHB Lite bus protocol.
System	Primarily for RAM and peripherals: Any accesses from address 0x20000000 to 0xFFFFFFFF (apart from PPB regions). Based on AMBA 3.0 AHB Lite bus protocol.
PPB	External Private Peripheral Bus (PPB): For private debug components on system level from address 0xE0040000 to 0xE00FFFFF. Based on AMBA 3.0 APB protocol.
DAP	Debug Access Port (DAP) interface: For debugger accesses generated from the debug interface module to any memory locations including system memory and debug components. Based on the ARM CoreSight™ debug architecture.

AMBA 3.0 System

The AMBA specification was developed by **ARM** and has become the de facto standard for interfacing components in an SoC.

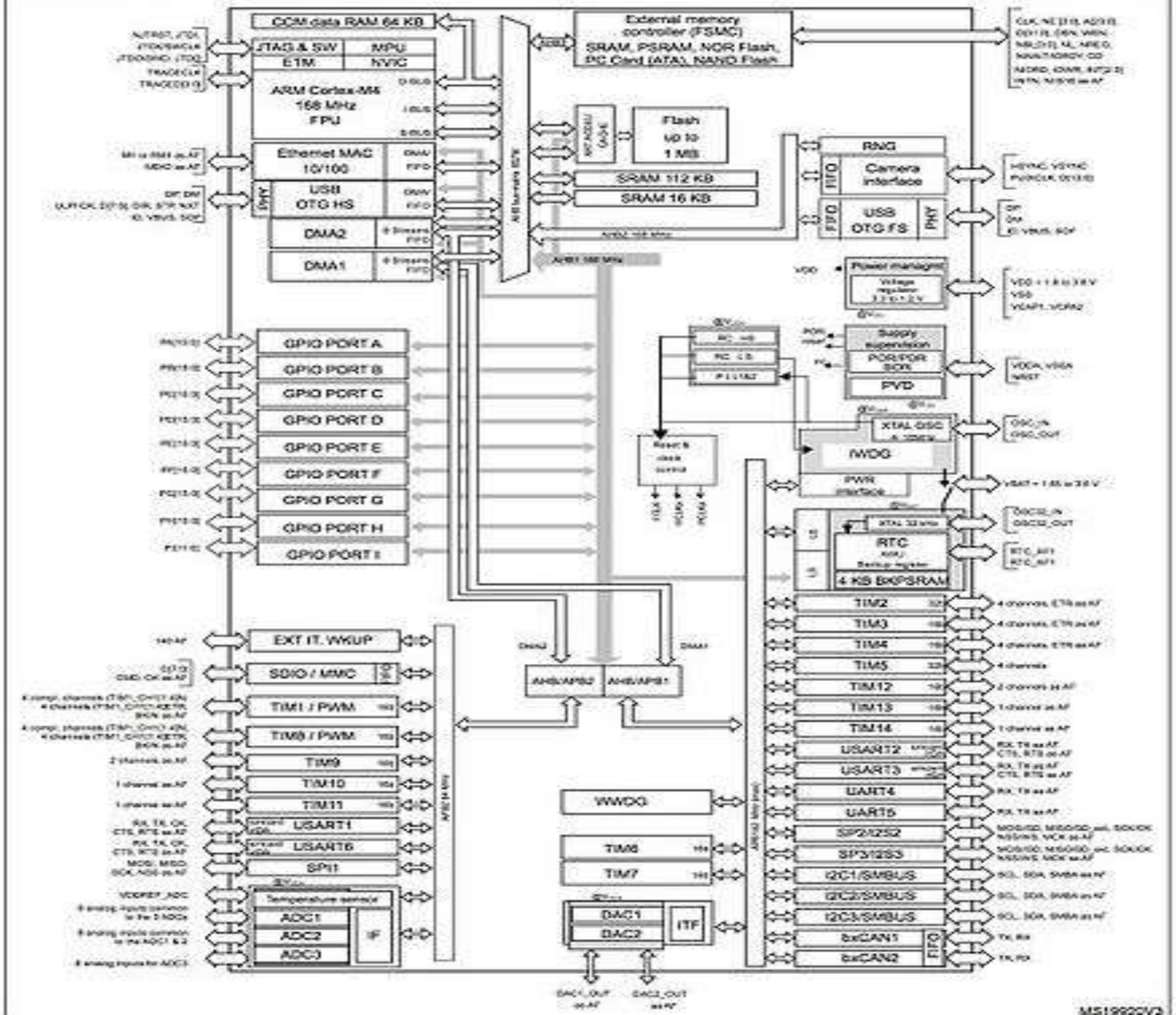


APB Bridge – **to regulate speed between AHB and APB buses**

High Performance
Pipelined
Burst Support
Multiple Bus Masters

Low Power
Non-pipelined
Simple Interface

STM32F407 Block Diagram


CCM data RAM 64 KB
JTAG & SW
ETM
MPU
NVIC
ARM Cortex-M4
168 MHz
FPU
Ethernet MAC
10/100
USB
OTG HS
DMA2
DMA1
External memory
controller (FSMC)
SRAM, PSRAM, NOR Flash,
PC Card (ATA), NAND Flash
Flash
up to
1 MB
SRAM 112 KB
SRAM 16 KB
RNG
Camera
interface
USB
OTG FS
Power managmt
Supply
supervision
POR/PDR
BOR
PVD
XTAL OSC
IWDG
PWR
interface
RTC
4 KB BKPSRAM
GPIO PORT A
GPIO PORT B
GPIO PORT C
GPIO PORT D
GPIO PORT E
GPIO PORT F
GPIO PORT G
GPIO PORT H
GPIO PORT I
EXT IT. WKUP
SDIO / MMC
TIM1 / PWM
TIM8 / PWM
TIM9
TIM10
TIM11
USART1
USART6
SPI1
Temperature sensor
ADC1
ADC2
ADC3
AHB/APB2
AHB/APB1
WWDG
TIM6
TIM7
DAC1
DAC2
ITF
TIM2
TIM3
TIM4
TIM5
TIM12
TIM13
TIM14
USART2
USART3
UART4
UART5
SPI2/I2S2
SPI3/I2S3
I2C1/SMBUS
I2C2/SMBUS
I2C3/SMBUS
bxCAN1
bxCAN2

Architecture

CDAC ACTS, Pune

DESIGNING FOR THE FUTURE

Agenda

Introduction to the Architecture
Programmers Model
- Operating Modes and States
- Registers
- Special registers

Behavior of Application program Status Register(APSR)
- Integer status Flags
- Q status flag
- GE bits

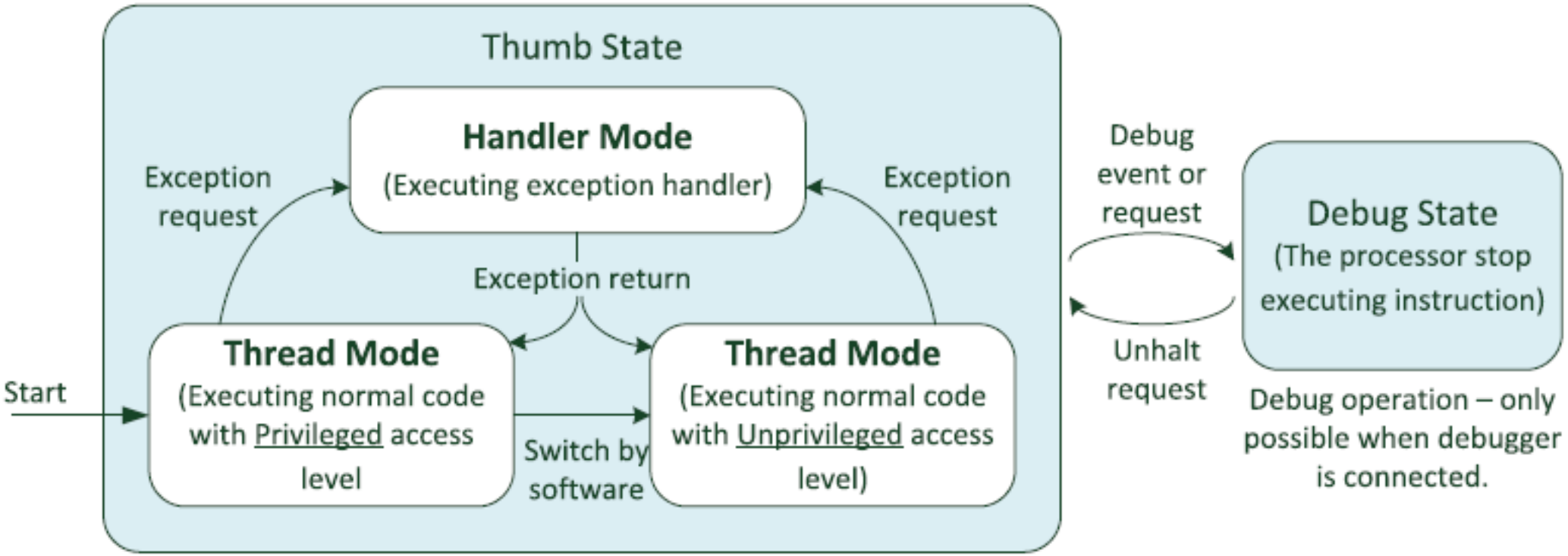
Memory System
- Memory system features and Memory Map
- Stack Memory & MPU

Exception and Interrupt
- What are exceptions?
- Nested vector interrupt controller (NVIC)
- Vector Table and Fault Handling

System Control Block
Debug and Reset Sequence

Programmer's Model

Operation states and Modes



CPU Operating Modes – Cont.

Modes (Thread out of reset)		Operations (privilege out of reset)	Stacks (Main out of reset)
	Handler - An exception is being processed	Privileged execution Full control	Main Stack Used by OS and Exceptions
	Thread - No exception is being processed - Normal code is executing	Privileged/Unprivileged	Main/Process

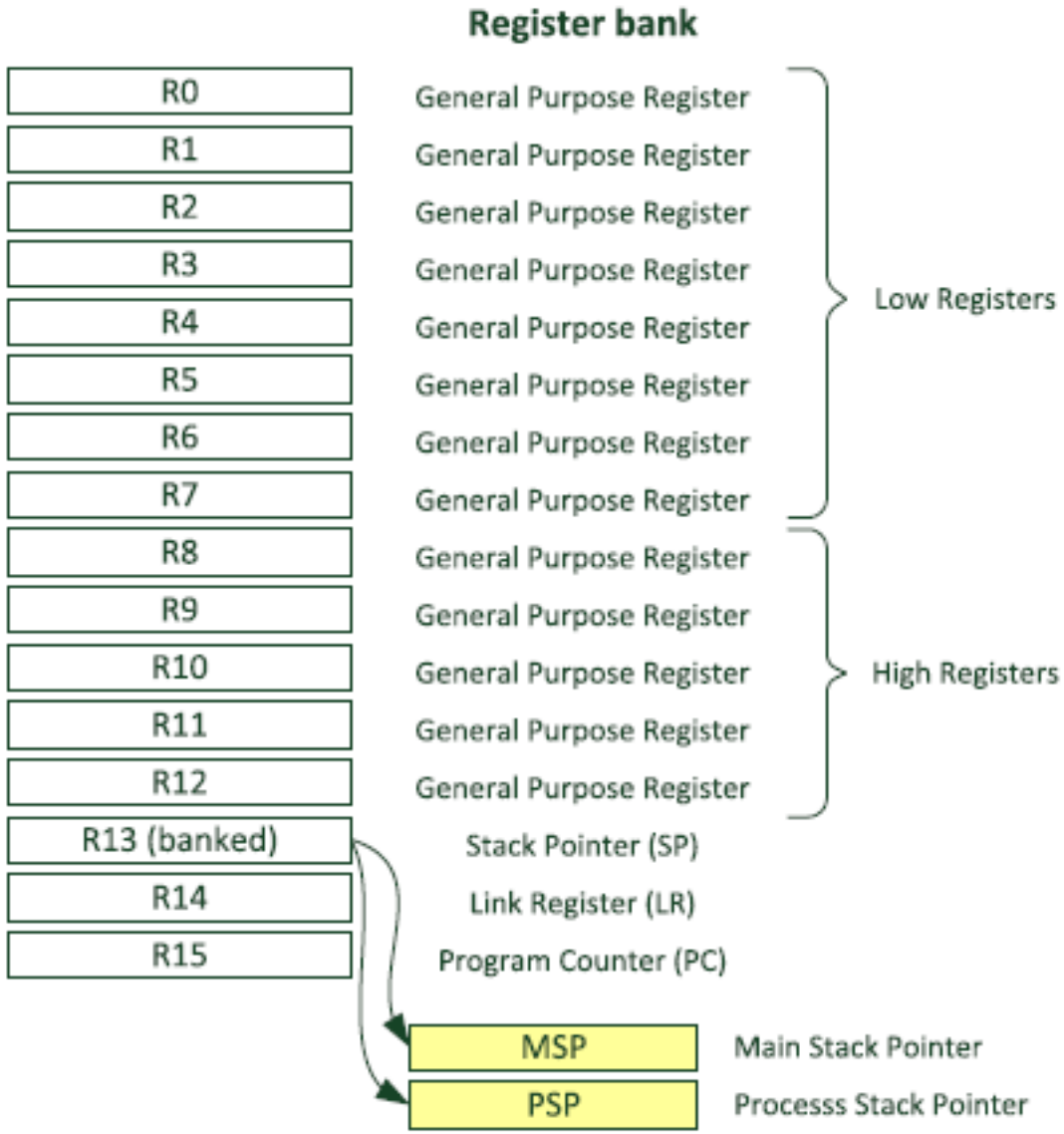
Operation modes

• Handler mode: When executing an exception handler such as an Interrupt Service Routine (ISR). When in handler mode, the processor always has privileged access level.

• Thread mode: When executing normal application code, the processor can be either in privileged access level or unprivileged access level. This is controlled by a special register called “CONTROL.”

Register Set

- R0-R12
  - General Purpose registers
  - R0-R7 low registers due to limited space available in IS , many 16 bit instruction can only access the low registers.
  - R8-R12 high registers can be used by 32-bit instruction
  - Initial Value of R0-R12 are undefined.
  - R13 Stack Pointer
  - R14 Link Register
  - R15 Program Counter



Stack Pointer –R13

Physically there are two different Stack Pointers:

- Main Stack Pointer is the default Stack Pointer. It is selected after reset, or when the processor is in Handler Mode.
- The other Stack Pointer is called the Process Stack Pointer. The PSP can only be used in Thread Mode. The selection of Stack Pointer is determined by a special register called CONTROL.
- The PSP is normally used when an embedded OS is involved, where the stack for the OS kernel and application tasks are separated.

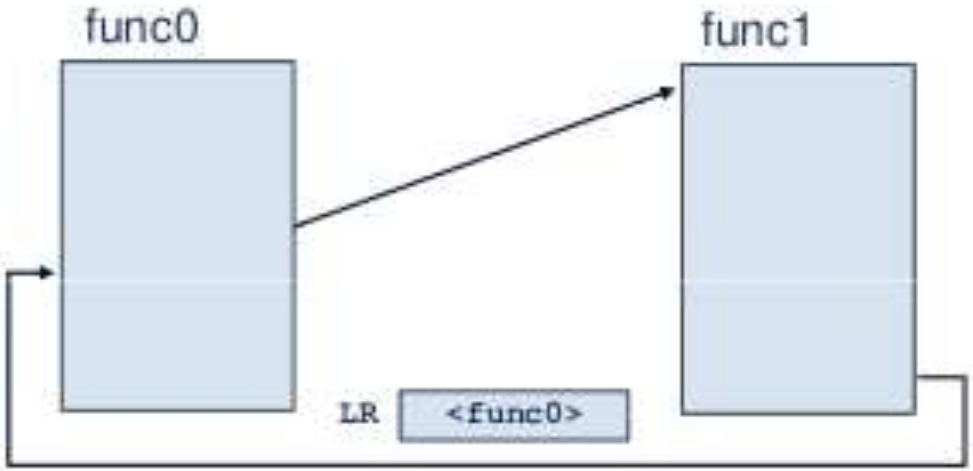
Link Register – R14

- This is used for holding the return address when calling a function or subroutine or ISR.
- At the end of the function or subroutine, the program control can return to the calling program and resume by loading the value of LR into the Program Counter (PC).
- When a function or subroutine call is made, the value of LR is updated automatically.
- If a function needs to call another function or subroutine, it needs to save the value of LR in the stack first. Otherwise, the current value in LR will be lost when the function call is made.

Link Register Flow diagram

- The Link Register (LR) is used to enable returns from subroutines

```
void func0 (void)
{
        :
        func1();
        :
}
```



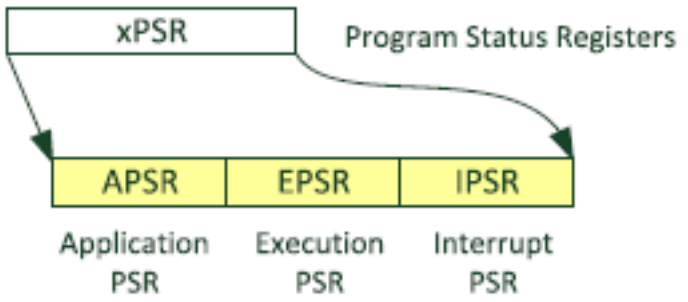
- **Further usage of the LR**
  - It has a special function for exception handling

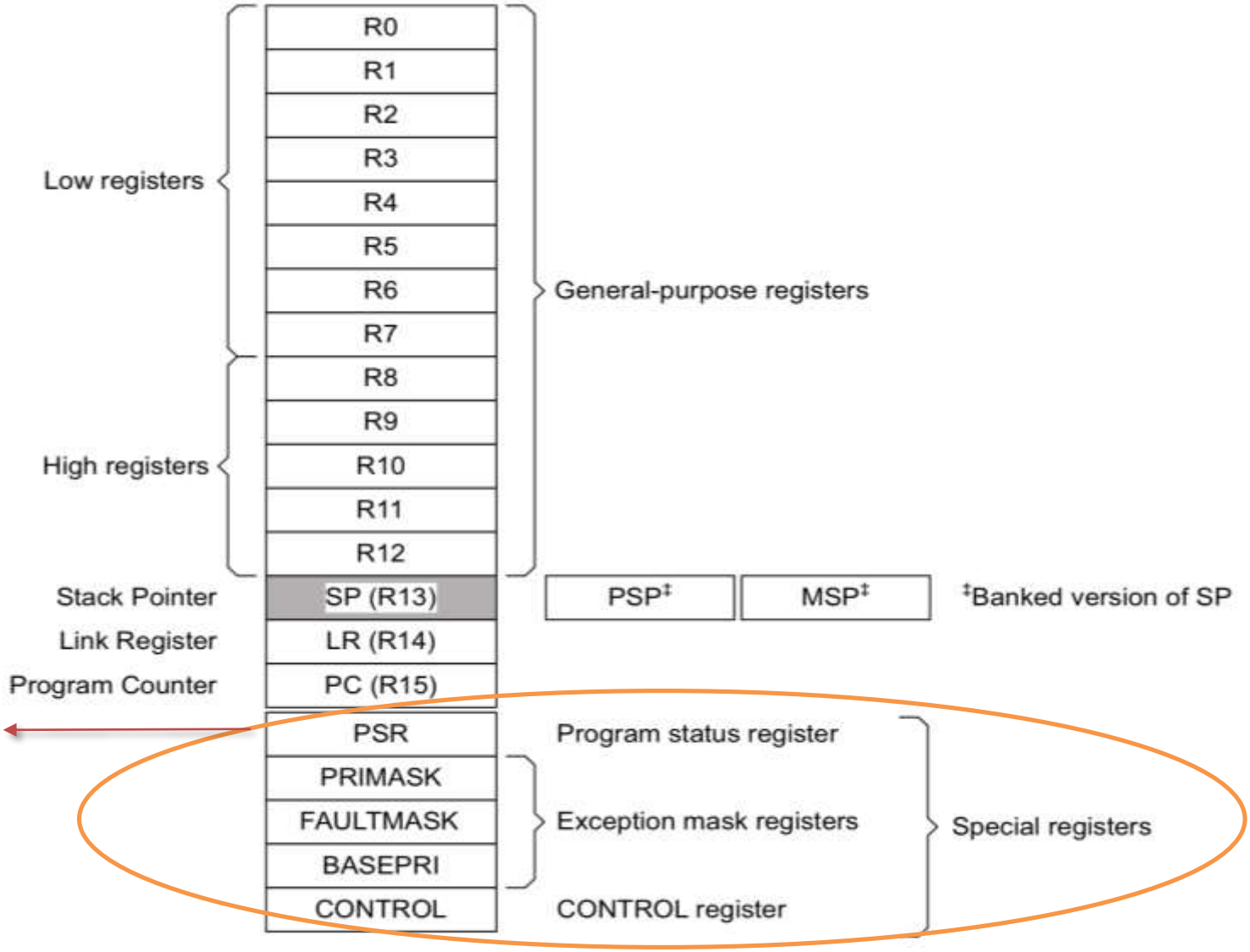
Program Counter – R15

- It is readable and writeable:
- a read returns the current instruction address plus 4 (this is due to the pipeline nature of the design, and compatibility requirement with the ARM7TDMI processor).
- Writing to PC (e.g., using data transfer/processing instructions) causes a branch operation.

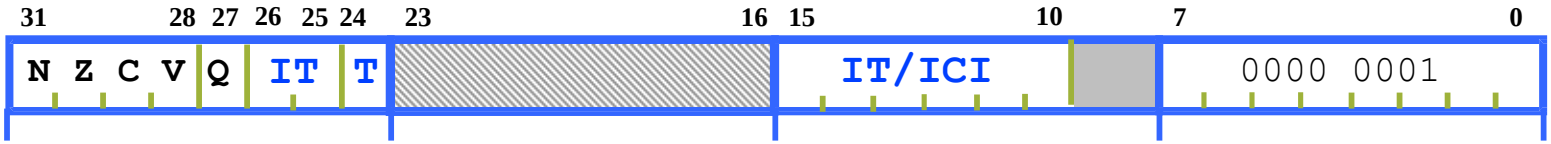
Special Registers

They are needed for development
of an embedded OS, or when advanced interrupt masking features are needed.
Special registers are not memory mapped, and can be accessed using special register access instructions such as MRS and MSR(Move to ARM register from system coprocessor register).





Program Status Register



- One Status Register consisting of
  - APSR - Application Program Status Register – ALU flags
  - IPSR - Interrupt Program Status Register – Interrupt/Exception No.
  - EPSR - Execution Program Status Register

    IT field – If/Then block information

    ICI field – Interruptible-Continuable Instruction information

  Please note:

  • The ERSR cannot be accessed by software code directly using MRS (read as zero) or MSR

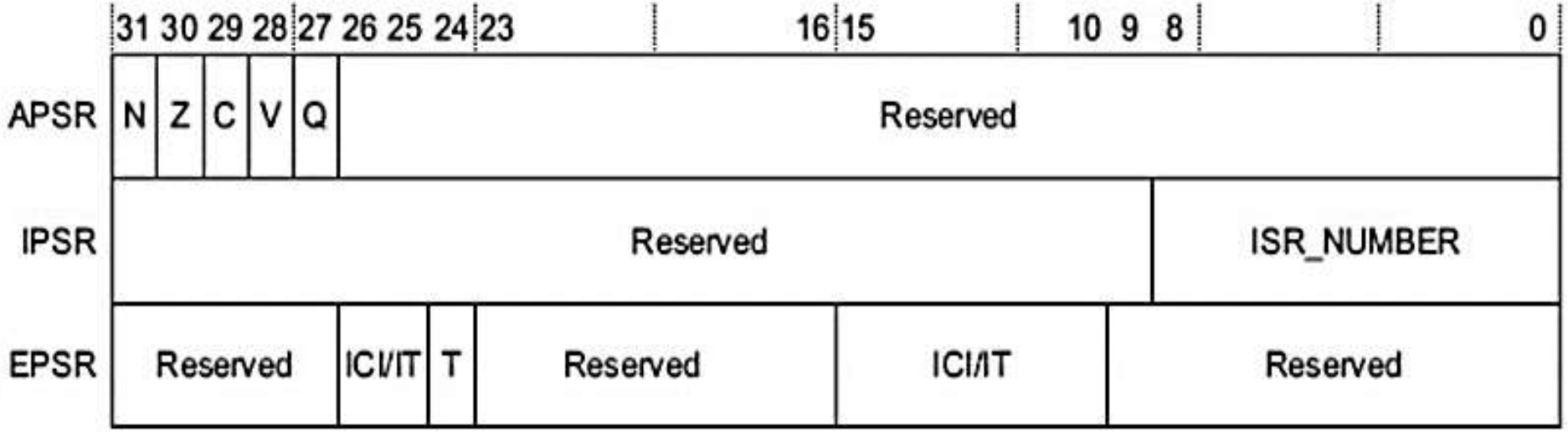
  • The IPSR is read only and can be read from combined PSR (xPSR).

Program Status Register: xPSR

The *Program Status Register* (PSR) combines:

- *Application Program Status Register* (APSR)
- *Interrupt Program Status Register* (IPSR)
- *Execution Program Status Register* (EPSR).

These registers are mutually exclusive bitfields in the 32-bit PSR. The bit assignments are:



APSR

The APSR contains the current state of the condition flags from previous instruction executions.

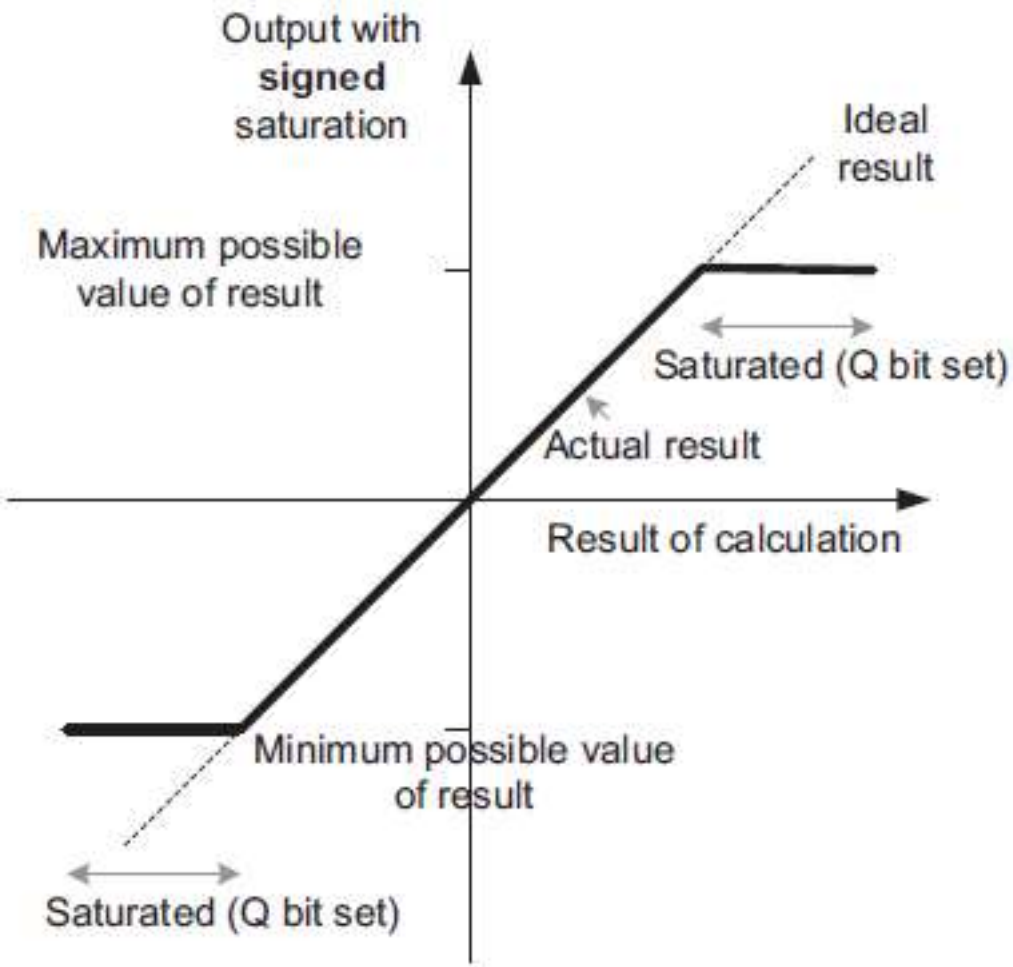
Bits 28 to 31 are alu condition code flags.

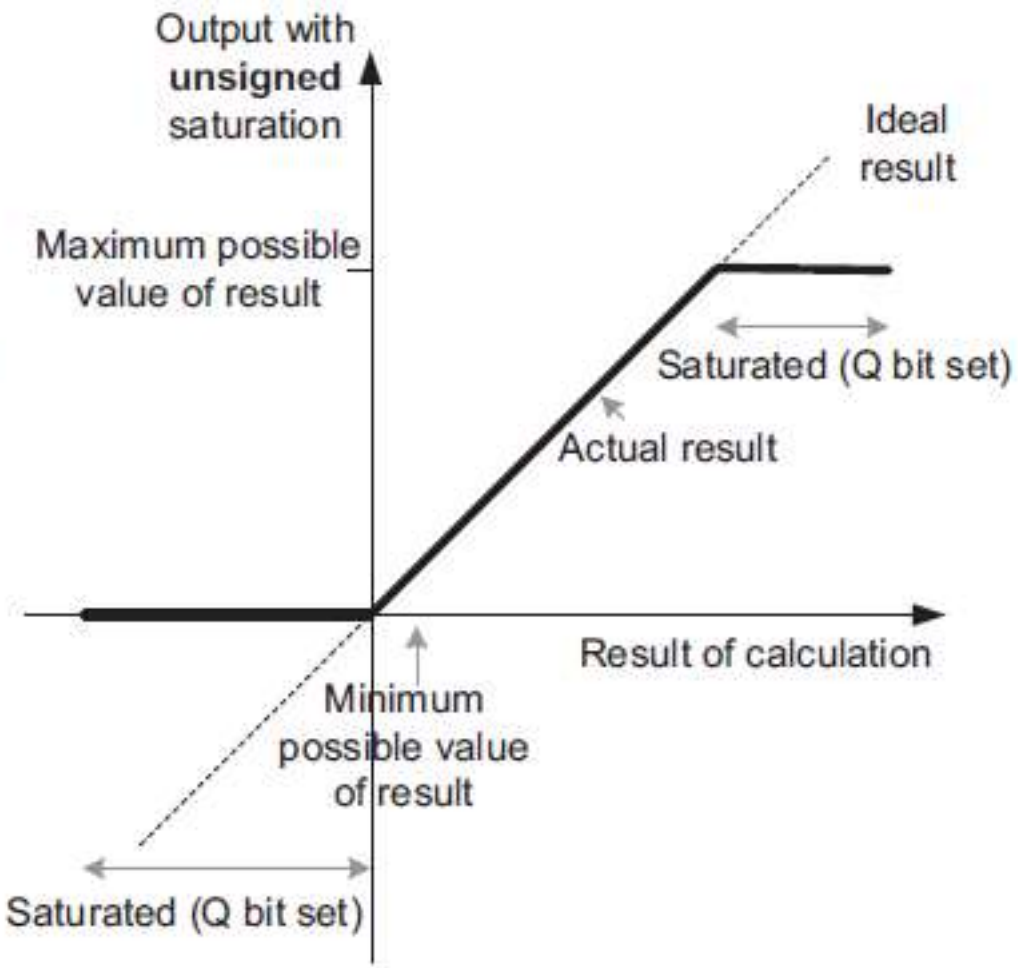
Q bit – sticky overflow flag , used by saturating instructions.

These alu are the only one you can modify in thread mode

Bits	Name	Function
[31]	N	Negative flag
[30]	Z	Zero flag
[29]	C	Carry or borrow flag
[28]	V	Overflow flag
[27]	Q	Saturation flag
[26:0]	-	Reserved

Q Flag – Signed & Unsigned Saturation





IPSR

The IPSR contains the exception type number of the current *Interrupt Service Routine* (ISR).

Bits	Name	Function
[31:9]	-	Reserved.
[8:0]	ISR_NUMBER	This is the number of the current exception: 0 = Thread mode 1 = Reserved 2 = NMI 3 = HardFault 4 = MemManage 5 = BusFault 6 = UsageFault 7-10 = Reserved 11 = SVCall 12 = Reserved for Debug 13 = Reserved 14 = PendSV 15 = SysTick 16 = IRQ0 . . . n+15 = IRQ(n-1)[a]. See *Exception types* on page 2-21 for more information.

a. The number of interrupts, n, is implementation-defined, in the range 1-240.

CONTROL register

The CONTROL register controls the stack used and the privilege level for software execution when the processor is in Thread mode.

		31:3	2	1	0
Cortex-M3 Cortex-M4	CONTROL			SPSEL	nPRIV

Bit	Function
nPRIV (bit 0)	Defines the privileged level in Thread mode: When this bit is 0 (default), it is privileged level when in Thread mode. When this bit is 1, it is unprivileged when in Thread mode. In Handler mode, the processor is always in privileged access level.
SPSEL (bit 1)	Defines the Stack Pointer selection: When this bit is 0 (default), Thread mode uses Main Stack Pointer (MSP). When this bit is 1, Thread mode uses Process Stack Pointer (PSP). In Handler mode, this bit is always 0 and write to this bit is ignored.

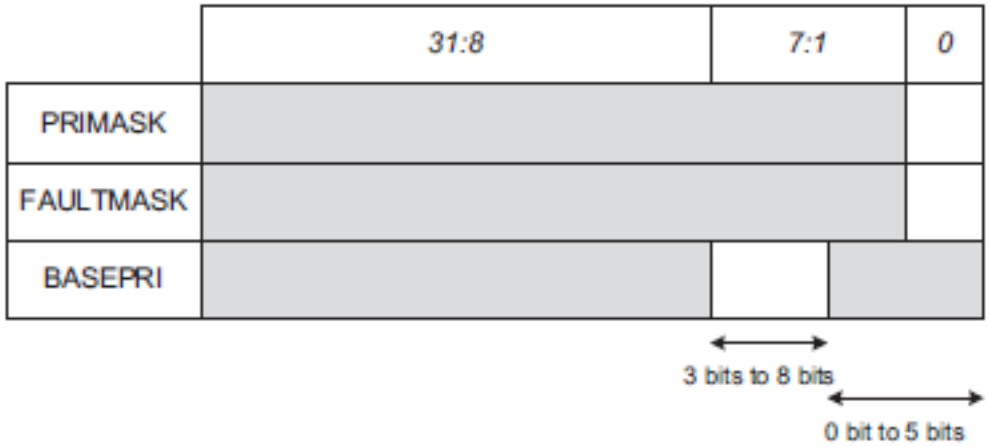
CONTROL register

- The CONTROL register can only be modified in the privileged access level and can be read in both privileged and unprivileged access levels.
- After reset, the CONTROL register is 0.

Cortex-M3
Cortex-M4

	31:3	2	1	0
CONTROL			SPSEL	nPRIV

Mask registers



- PRIMASK
  - 1-bit Register
  - changes the execution priority to 0(Highest Priority)
- FAULTMASK
  - 1-bit Register
  - Changes the execution priority to -1(Hard Fault)
- BASEPRI
  - 8-bit Register
  - Changes the priority level required for preemption

- What if we quickly want to disable all interrupts?
- Write 1 into **PRIMASK** to disable all interrupt except NMI
  - MOV R0, #1
  - MSR PRIMASK, R0
- Write 0 into **PRIMASK** to enable all interrupts
- FAULTMASK is the same as PRIMASK, but also blocks hard fault (priority -1)

- What if we want to disable all interrupts below a certain priority?
- Write priority into **BASEPRI**
  - MOV R0, #0x60
  - MSR BASEPRI, R0

Stacks

Two run-time models supported

- Single Stack Pointer – MSP for entire application
- Two Stack Pointers
  - MSP for Handler Mode (Exception Handling)
  - PSP for Thread Mode (Application Code)

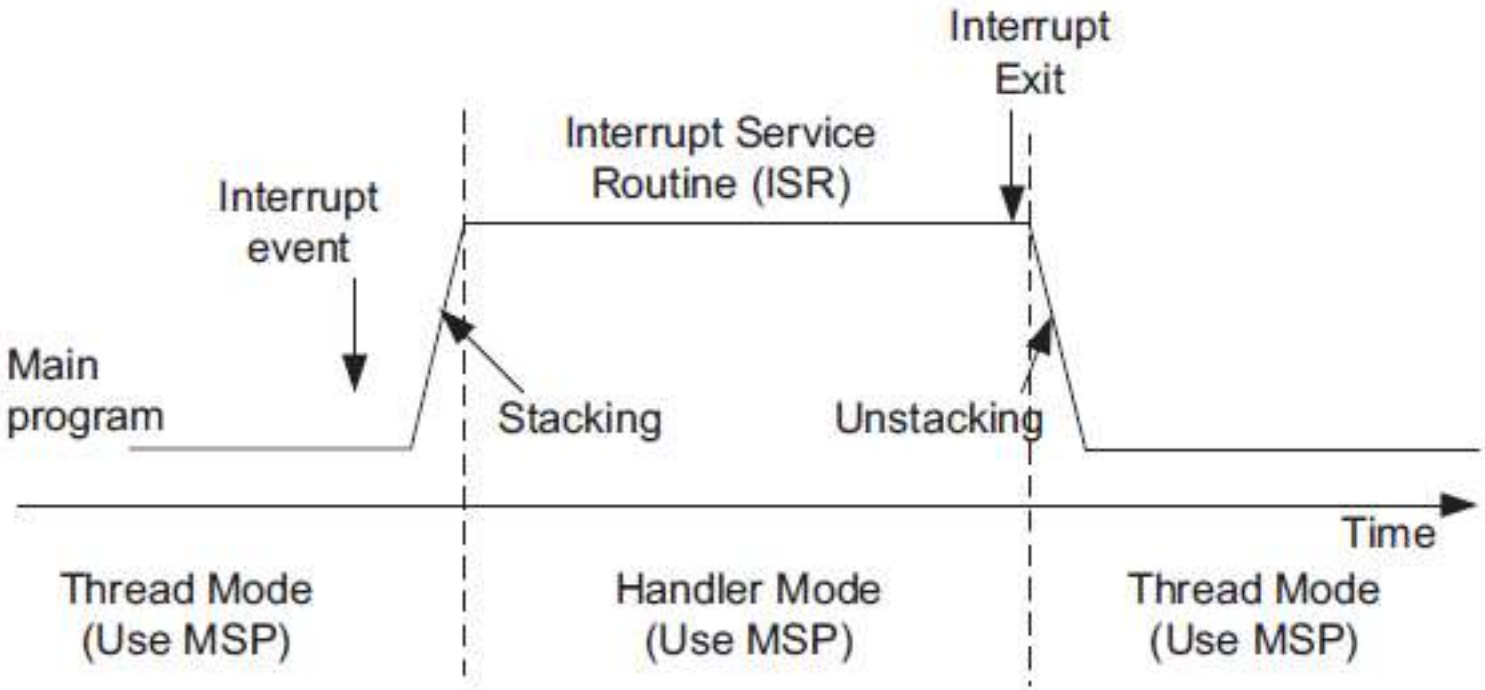
Main Stack Pointer (MSP)

- Used by Thread Mode out of reset
  - Initial MSP value is taken from first entry of Vector Table
- Always used by Handler Mode

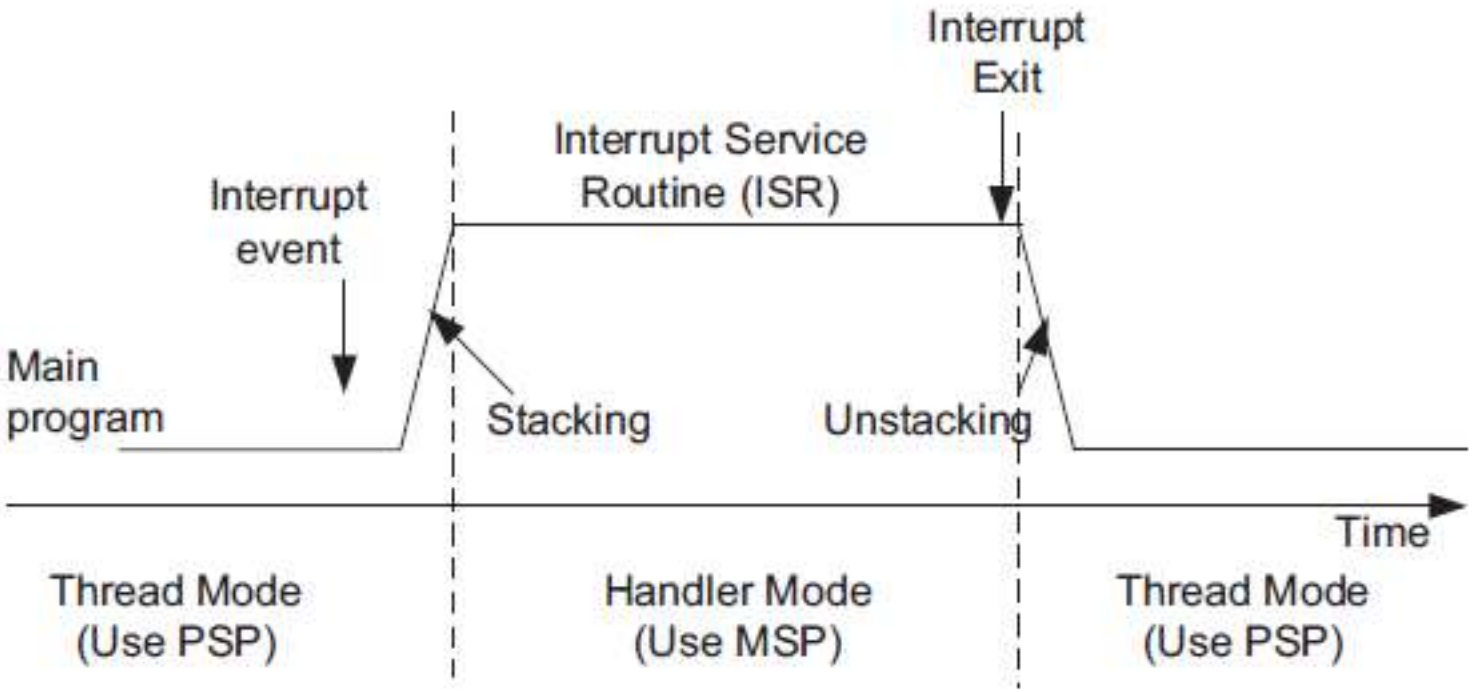
Process Stack Pointer (PSP)

- Optionally used for Thread Mode
- PSP is enabled using CONTROL.SPSEL
  - Must be initialized by user before being used

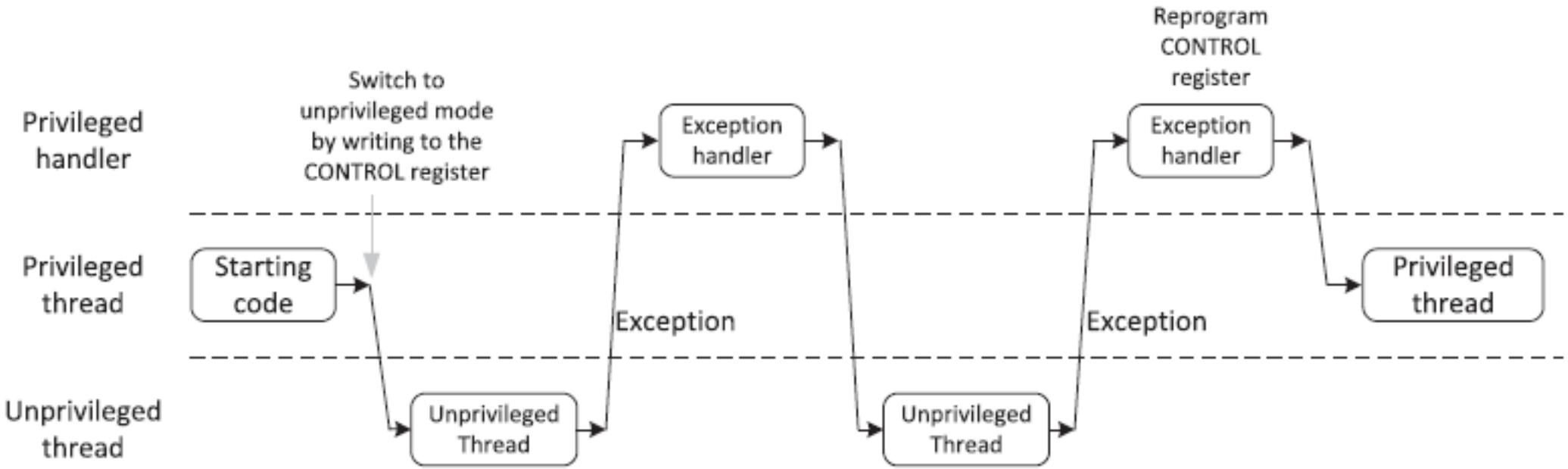
Both Thread and Handler using MSP



Thread uses PSP & Handler uses MSP



Switching Privileged & Unprivileged



Privilege, Modes and Stacks

Thread mode and handler mode

- Handler mode is for an exception or interrupt
- Thread mode is for normal application code execution

Privileged/non-privileged operation

- Handler mode is always privileged
- Thread mode can be in privileged or non-privileged mode

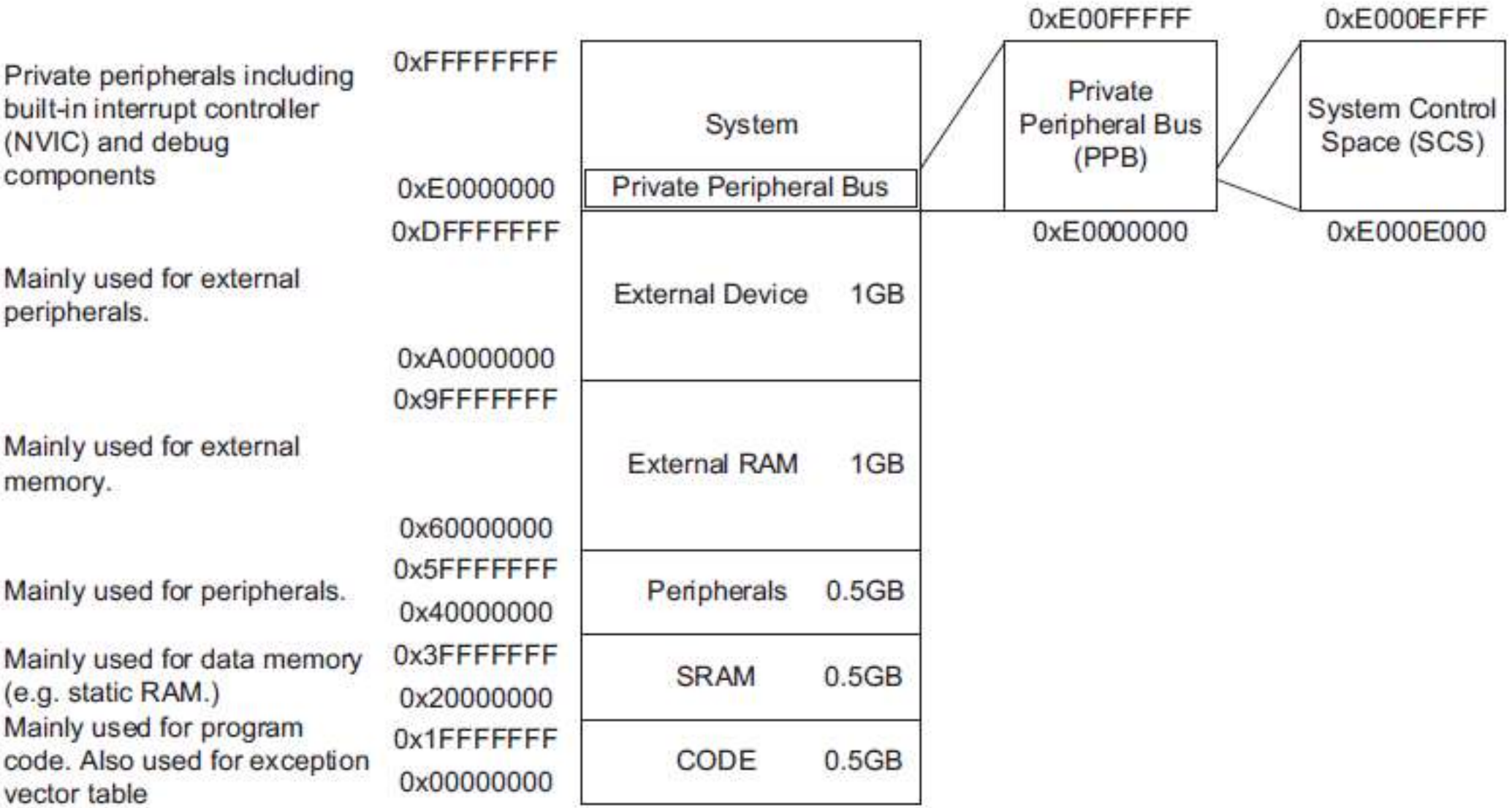
Stacks - Main stack and Process stack

- Both stacks have their own stack pointer (r13) register
- Exceptions always use main stack in privileged mode
- Applications (thread mode) can use either main or process stack

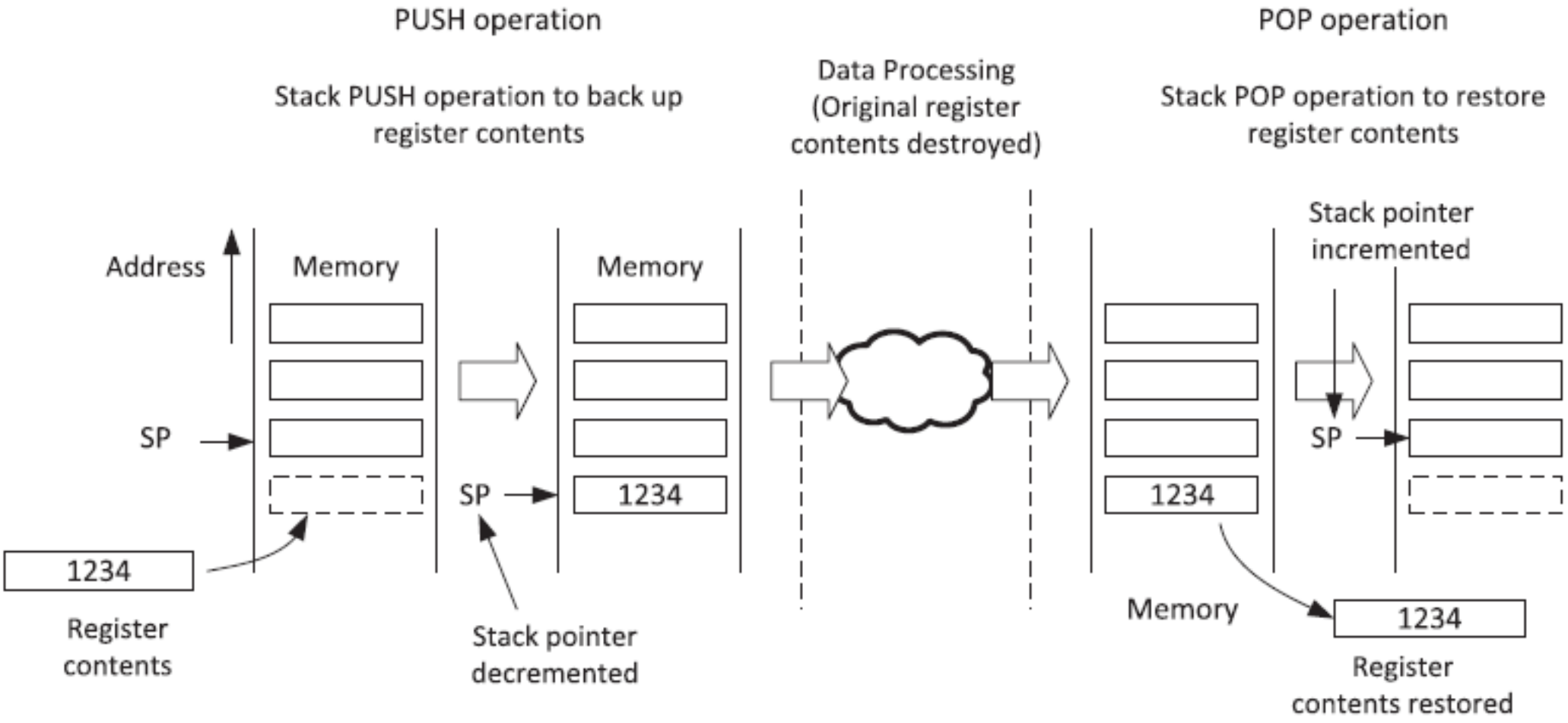
Memory System

- Memory System features
  - 4GB linear address space
  - Architecturally defined memory map
  - Support for little endian and big endian memory systems
  - Bit band accesses (optional)
  - Write buffer - When a write transfer to a buffered memory region will take multiple cycles, the transfer can be buffered by the internal write buffer in the Cortex-M3 or Cortex-M4 processor so that the processor can continue to execute the next instruction, if possible. This allows higher program execution speed.
  - Memory Protection Unit (Optional)
  - Unaligned transfer support

Memory Map



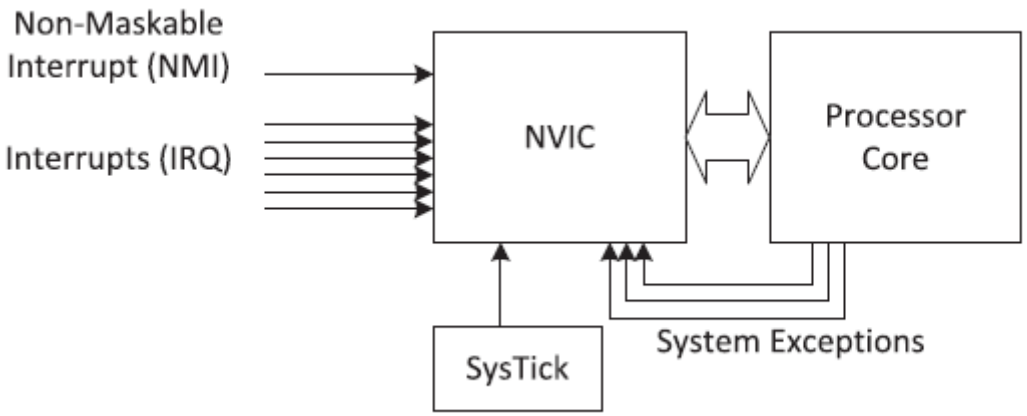
Stack Memory



Exceptions and Interrupts

- What are exceptions?
  - Exceptions are events that cause changes to program flow. When one happens, the processor suspends the current executing task and executes a part of the program called the exception handler.
  - After the execution of the exception handler is completed, the processor then resumes normal program execution.
  - In the ARM architecture, interrupts are one type of exception. Interrupts are usually generated from peripheral or external inputs, and in some cases they can be triggered by software.
  - The exception handlers for interrupts are also referred to as Interrupt Service Routines (ISR).

Various Exception sources



Exception Types

Exception Number	CMSIS Interrupt Number	Exception Type	Priority	Function
1	—	Reset	–3 (Highest)	Reset
2	–14	NMI	–2	Non-Maskable interrupt
3	–13	HardFault	–1	All classes of fault, when the corresponding fault handler cannot be activated because it is currently disabled or masked by exception masking
4	–12	MemManage	Settable	Memory Management fault; caused by MPU violation or invalid accesses (such as an instruction fetch from a non-executable region)
5	–11	BusFault	Settable	Error response received from the bus system; caused by an instruction prefetch abort or data access error
6	–10	Usage fault	Settable	Usage fault; typical causes are invalid instructions or invalid state transition attempts (such as trying to switch to ARM state in the Cortex-M3)
7–10	—	—	—	Reserved
11	–5	SVC	Settable	Supervisor Call via SVC instruction
12	–4	Debug monitor	Settable	Debug monitor – for software based debug (often not used)
13	—	—	—	Reserved
14	–2	PendSV	Settable	Pendable request for System Service
15	–1	SYSTICK	Settable	System Tick Timer
16–255	0–239	IRQ	Settable	IRQ input #0–239

Nested Vectored Interrupt Controller

- Features
  - Flexible exception and interrupt management
    - Each interrupt (apart from the NMI) can be enabled or disabled and can have its pending status set or cleared by software.
  - Nested exception/interrupt support
    - Each exception has a priority level. Some exceptions, such as interrupts, have programmable priority levels and some others (e.g., NMI) have a fixed priority level. When an exception occurs, the NVIC will compare the priority level of this exception to the current level. If the new exception has a higher priority, the current running task will be suspended.
  - Vectored exception/interrupt entry
    - The Cortex-M processors automatically locate the starting point of the exception handler from a vector table in the memory. As a result, the delays from the start of the exception to the execution of the exception handlers are reduced.
  - Interrupt masking

Starting Address of Exception Handler

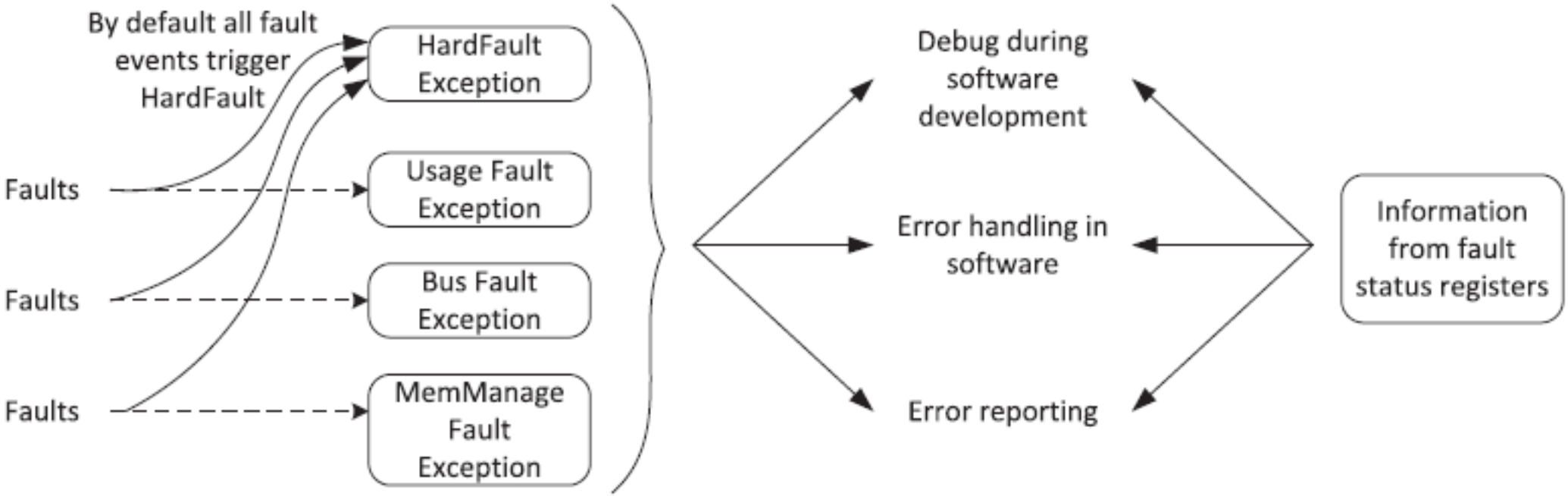
- To determine the starting address of the exception handler, a vector table mechanism is used.
- The vector table is an array of word data inside the system memory, each representing the starting address of one exception type.
- Example :The vector table is located at address 0x0 after reset.
- if the reset is exception type 1, the address of the reset vector is 1 times 4 (each word is 4 bytes), which equals 0x00000004, and the NMI vector (type 2) is located at 2 x 4 = 0x00000008.
- The address 0x00000000 is used to store the starting value of the MSP.
- The LSB of each exception vector indicates whether the exception is to be executed in the Thumb state. Since the Cortex-M processors can support only Thumb instructions, the LSB of all the exception vectors should be set to 1.

Vector Table

LSB of exception vectors should be set to 1 to indicate Thumb state

Exception Type	CMSIS Interrupt Number	Address Offset	Vectors
18 - 255	2 - 239	0x48 – 0x3FF	IRQ #2 - #239 1
17	1	0x44	IRQ #1 1
16	0	0x40	IRQ #0 1
15	-1	0x3C	SysTick 1
14	-2	0x38	PendSV 1
NA	NA	0x34	Reserved
12	-4	0x30	Debug Monitor 1
11	-5	0x2C	SVC 1
NA	NA	0x28	Reserved
NA	NA	0x24	Reserved
NA	NA	0x20	Reserved
NA	NA	0x1C	Reserved
6	-10	0x18	Usage fault 1
4	-11	0x14	Bus Fault 1
4	-12	0x10	MemManage Fault 1
3	-13	0x0C	HardFault 1
2	-14	0x08	NMI 1
1	NA	0x04	Reset 1
NA	NA	0x00	Initial value of MPS

Fault Handling



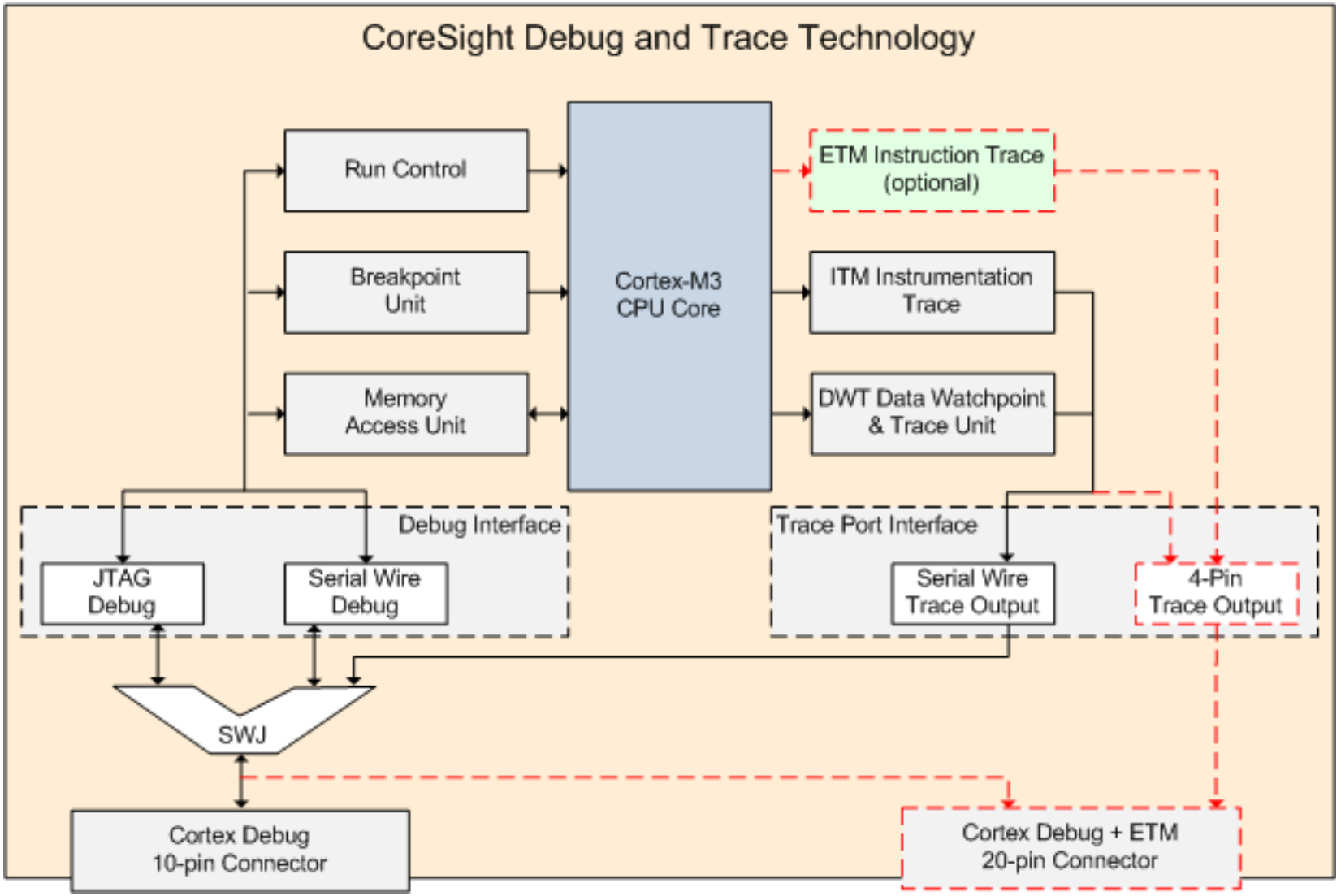
System Control Block (SCB)

One part of the processor that is merged into the NVIC unit is the SCB. The SCB contains various registers for:

- Controlling processor configurations (e.g., low power modes)
- Providing fault status information (fault status registers)
- Vector table relocation (VTOR)

- The SCB is memory-mapped. Similar to the NVIC registers, the SCB registers are accessible from the System Control Space (SCS).

Core Sight Debug and Trace Technology



Core Sight features

- Core Sight features can be accessed through a JTAG or Serial Wire interface. Debugging in JTAG and Serial Wire mode at the same time is not possible. Cortex-M processor-based devices can include a:
- **Debug Interface**
- The debug interface offers two modes:
- **JTAG Debug** is the industry-standard interface that allows device chaining.
- Serial Wire Debug is a 2-pin interface with an optional Serial Wire Trace Output. In contrast to JTAG, devices cannot be chained.
- The **Debug Interface** communicates with the following units:
- **Run Control**: allows the user to start, stop, and single-step through the source code.
- **Breakpoint Unit**: allows the user to set breakpoints even while the processor is running.
- **Memory Access Unit**: allows the user to read or write to memory and peripheral registers even while the program is running.

Cortex Reset Sequence & Startup

Processor will be in thread mode with privileged operation

Processor will use main stack

Core will fetch the MSP and PC from the vector table

- Vector table will be located at address 0x0 (generally non-volatile memory)
- PSP (if used) can be set later in reset handler using MRS instruction

All interrupts are disabled

- Vector table must contain valid value for NMI Handler and Hard Fault Handler
- PRIMASK, FAULTMASK and BASEPRI are cleared

MPU is disabled

- Default memory map is used

Vector Table at Startup

First entry contains initial Main SP

All other entries are addresses for exception handlers

- Must always have LSBit = 1 (for Thumb)

Table has up to 496 external interrupts

- Implementation-defined
- Maximum table size is 2048 bytes

Table may be relocated

- Use Vector Table Offset Register
- Still require minimal table entries at 0x0 for booting the core

Each exception has an exception number

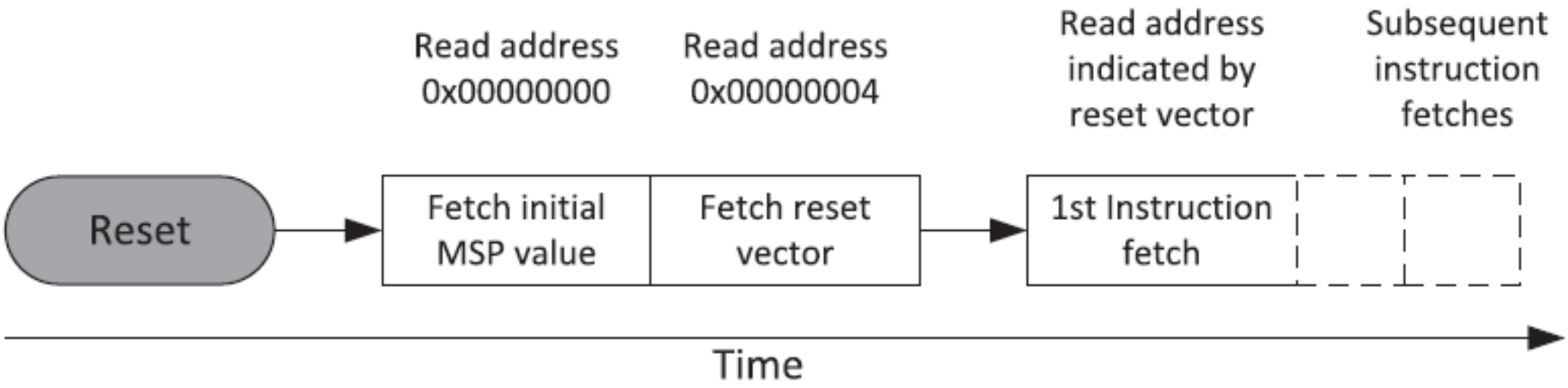
- Used in Interrupt Control and State Register to indicate the active or pending exception type

Table can be generated using C code

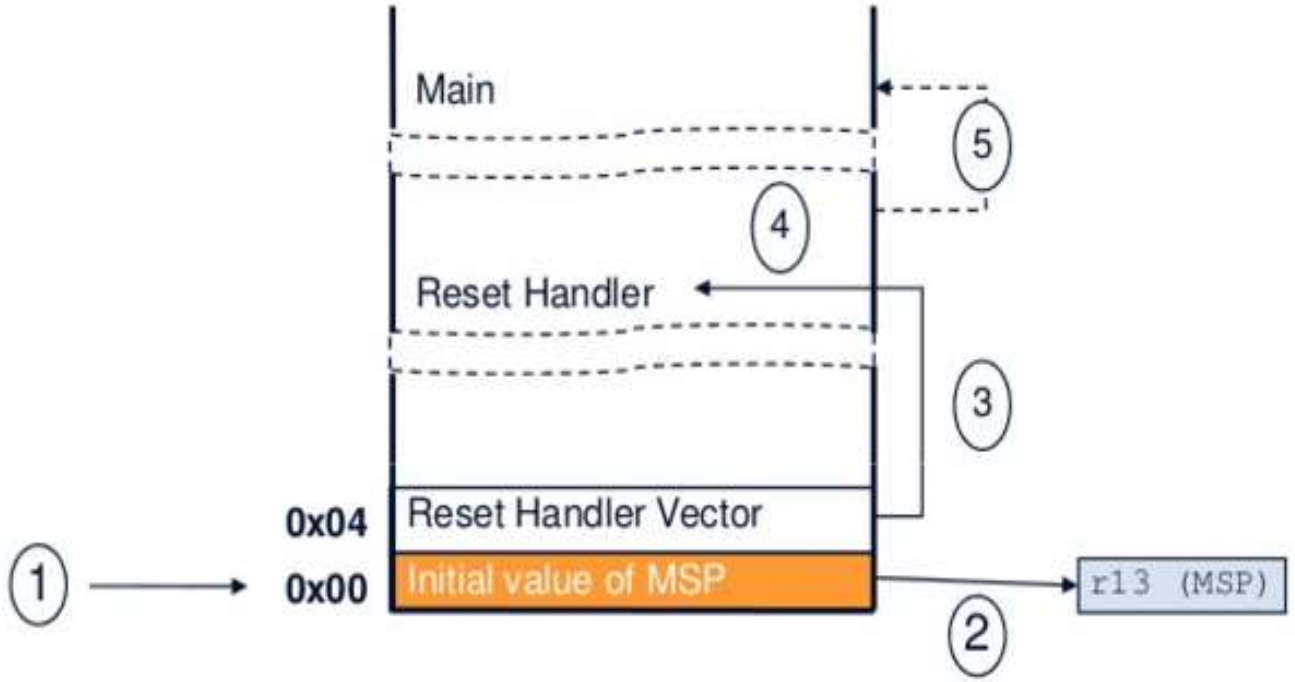
- Example provided later

Address		Exception
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

Reset Sequence

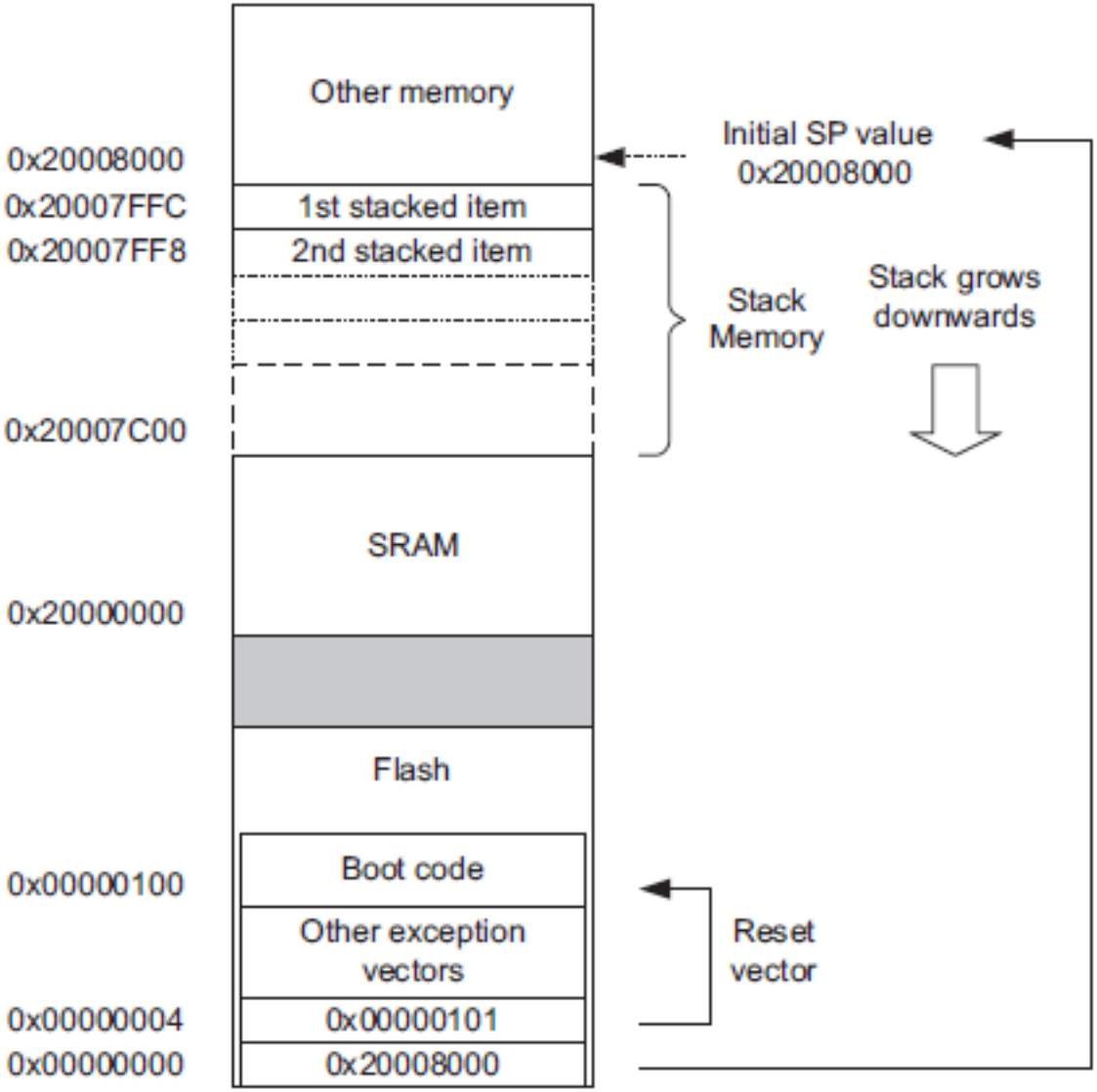


Reset Behavior



1. A reset occurs (Reset input was asserted)
2. Load MSP (Main Stack Pointer) 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

Status of SP and PC during reset



Initialization Summary

Cortex-M series cores start up in Privileged Thread mode

For ARMv7-M architecture-based cores, you need to consider which mode your main application will run in

- Privileged vs. Unprivileged mode
- Note that ARMv6-M architecture-based cores (Cortex-M0 and Cortex-M1) do not have Unprivileged mode

System initialization can only be executed in Privileged mode

- Need to carry out privileged operations e.g. setup MPU, enable interrupts

Before entering main application, typically need to:

- Setup and enable MPU (ARMv7-M cores only)
- Enable hardware enforcement of 8-byte stack alignment for exception handlers
- Complete any memory (Flash/RAM) remapping
- Enable interrupts
- (Last) Change Thread mode to Unprivileged – if required

Memory System

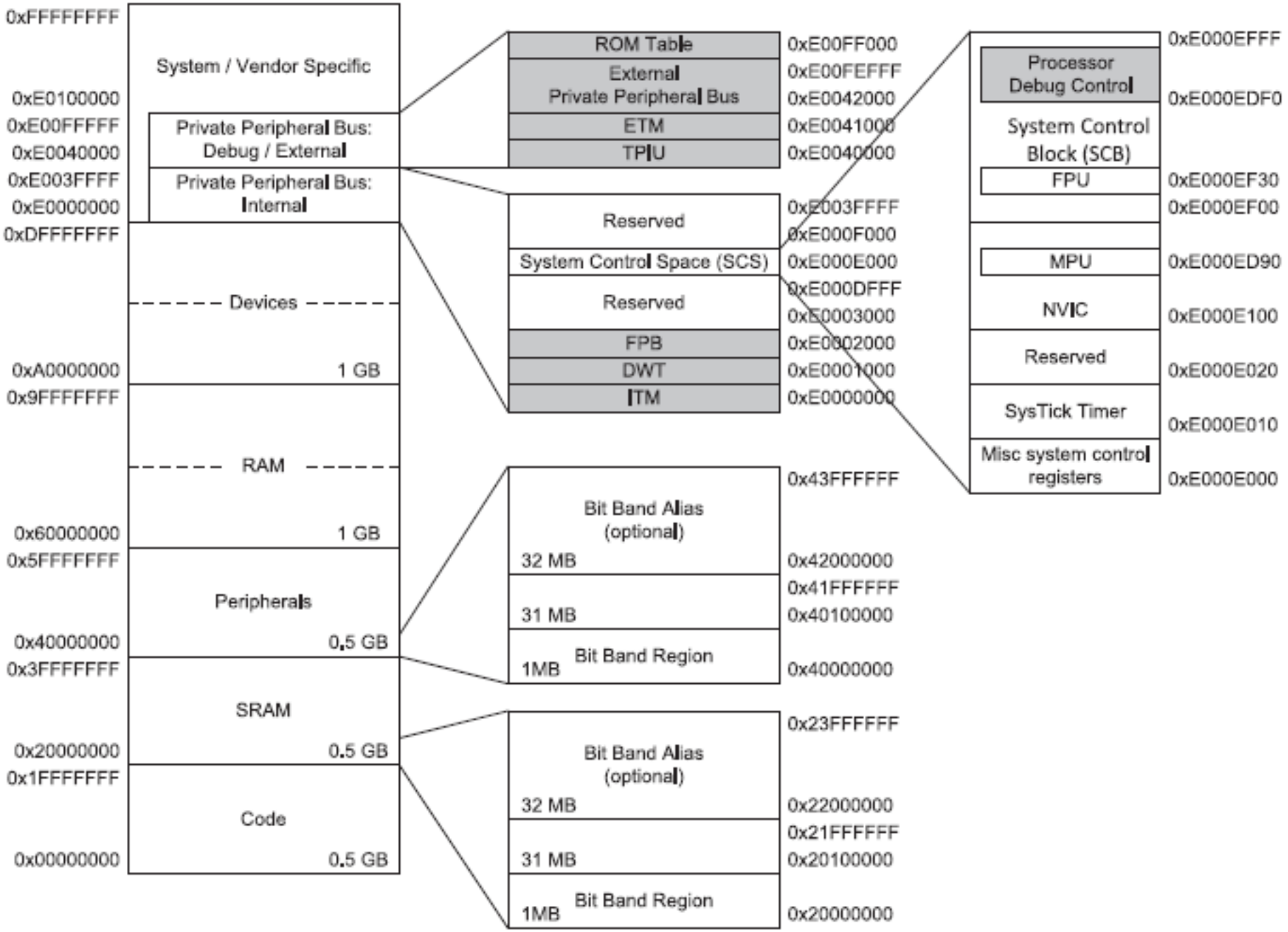
CDAC ACTS, Pune

DESIGNING FOR THE FUTURE

Agenda

Memory Map
Connecting the processor to memory and peripheral
Memory Requirements
Memory Endianness
Data alignment and Unaligned data access
Bit-band operations
Default Memory Access permissions
Exclusive access
Memory Barrier
Memory System in a microcontroller

Memory Map



Bit Banding

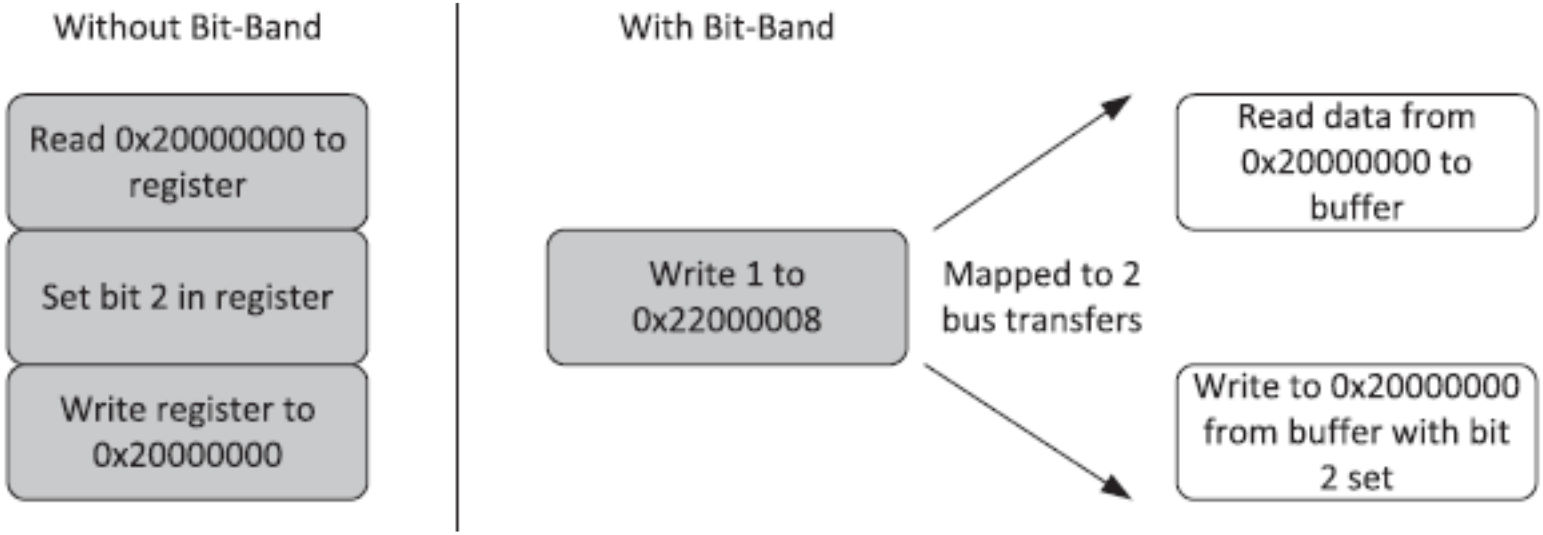
The earlier ARM7 and ARM9 CPUs were only able to perform bit manipulations on SRAM and peripheral memory locations by using AND and OR operations. This requires a READ MODIFY WRITE operation which is expensive in terms of the number of cycles taken to set and clear individual bits and the overall code space required for each bit manipulation.

To overcome this limitation it would be possible to introduce a dedicated bit set and clear instructions, or a full Boolean processor, but this would increase the size and complexity of the Cortex CPU. Instead, a technique called bit banding allows direct bit manipulation on sections of the peripheral and SRAM memory spaces, without the need for any special instructions. The bit addressable regions of the Cortex memory map are composed of the bit band region (which is up to 1Mbyte of real memory or peripheral registers) and the bit band Alias region which takes up to 32Mbyte of the memory map. Bit banding works by mapping each bit in the bit band region to a word address in the Alias region. So by setting and clearing the aliased word address we can set and clear bits in the real memory.

The memory map has two 32MB alias regions that map to two 1MB bit-band regions:

- accesses to the 32MB SRAM alias region map to the 1MB SRAM bit-band region, as shown in Table 2-13
- accesses to the 32MB peripheral alias region map to the 1MB peripheral bit-band region, as shown in Table 2-14.

Read Modify Write vs Bit Banding



Assembler sequence to write a bit with and without bit-band

Without Bit-Band

```
LDR   R0,=0x20000000 ; Setup address
LDR   R1, [R0]       ; Read
ORR.W R1, #0x4       ; Modify bit
STR   R1, [R0] ; Write back result
```

With Bit-Band

```
LDR   R0,=0x22000008 ; Setup address
MOV   R1, #1         ; Setup data
STR   R1, [R0]       ; Write
```

Read from the bit-band alias

Without Bit-Band

```
LDR    R0,=0x20000000 ; Setup address
LDR    R1, [R0]       ; Read
UBFX.W R1, R1, #2, #1 ; Extract bit[2]
```

With Bit-Band

```
LDR   R0,=0x22000008 ; Setup address
LDR   R1, [R0]       ; Read
```

Advantages of Bit-Band

- Bit-Band vs. Bit-Bang
  - In the Cortex-M3, we use the term bit-band to indicate that the feature is a special memory band (region) that provides bit accesses.
  - Bit-bang commonly refers to driving I/O pins under software control to provide serial communication functions. The bit-band feature in the Cortex-M3 can be used for bit-banging implementations, but the definitions of these two terms are different.
- Reading the whole register
- Masking the unwanted bits
- Comparing and branching
- You can simplify the operations to:
- Reading the status bit via the bit-band alias (get 0 or 1)
- Comparing and branching

Memory Endianness

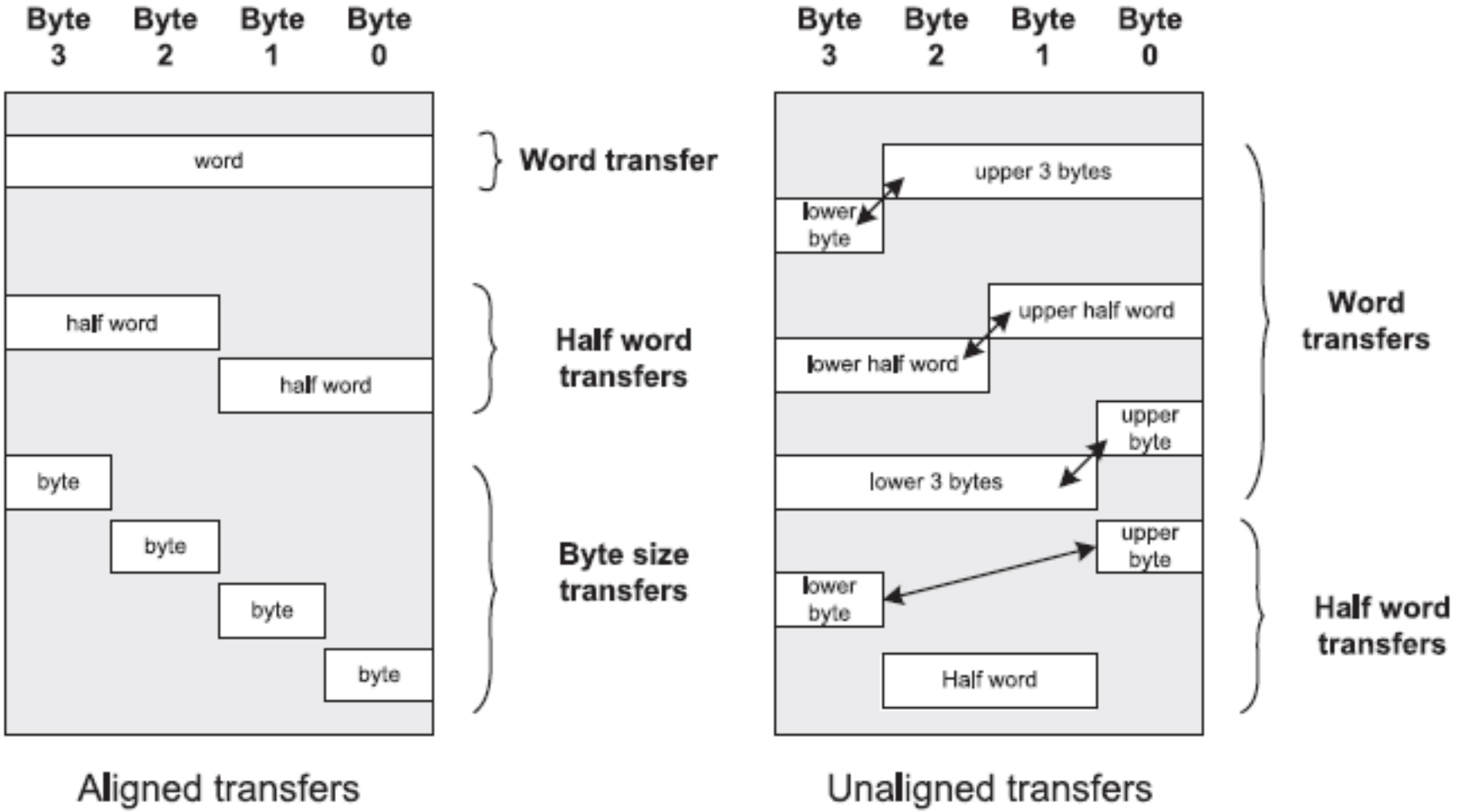
The Cortex-M3 (byte-invariant big-endian, BE-8) – Data on the AHB Bus

Address, Size	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x1000, word	Data bit[7:0]	Data bit [15:8]	Data bit [23:16]	Data bit [31:24]
0x1000, half word	-	-	Data bit [7:0]	Data bit [15:8]
0x1002, half word	Data bit [7:0]	Data bit [15:8]	-	-
0x1000, byte	-	-	-	Data bit [7:0]
0x1001, byte	-	-	Data bit [7:0]	-
0x1002, byte	-	Data bit [7:0]	-	-
0x1003, byte	Data bit [7:0]	-	-	-

Little Endian – Data on the AHB Bus

Address, Size	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x1000, word	Data bit [31:24]	Data bit [23:16]	Data bit [15:8]	Data bit [7:0]
0x1000, half word	-	-	Data bit [15:8]	Data bit [7:0]
0x1002, half word	Data bit [15:8]	Data bit [7:0]	-	-
0x1000, byte	-	-	-	Data bit [7:0]
0x1001, byte	-	-	Data bit [7:0]	-
0x1002, byte	-	Data bit [7:0]	-	-
0x1003, byte	Data bit [7:0]	-	-	-

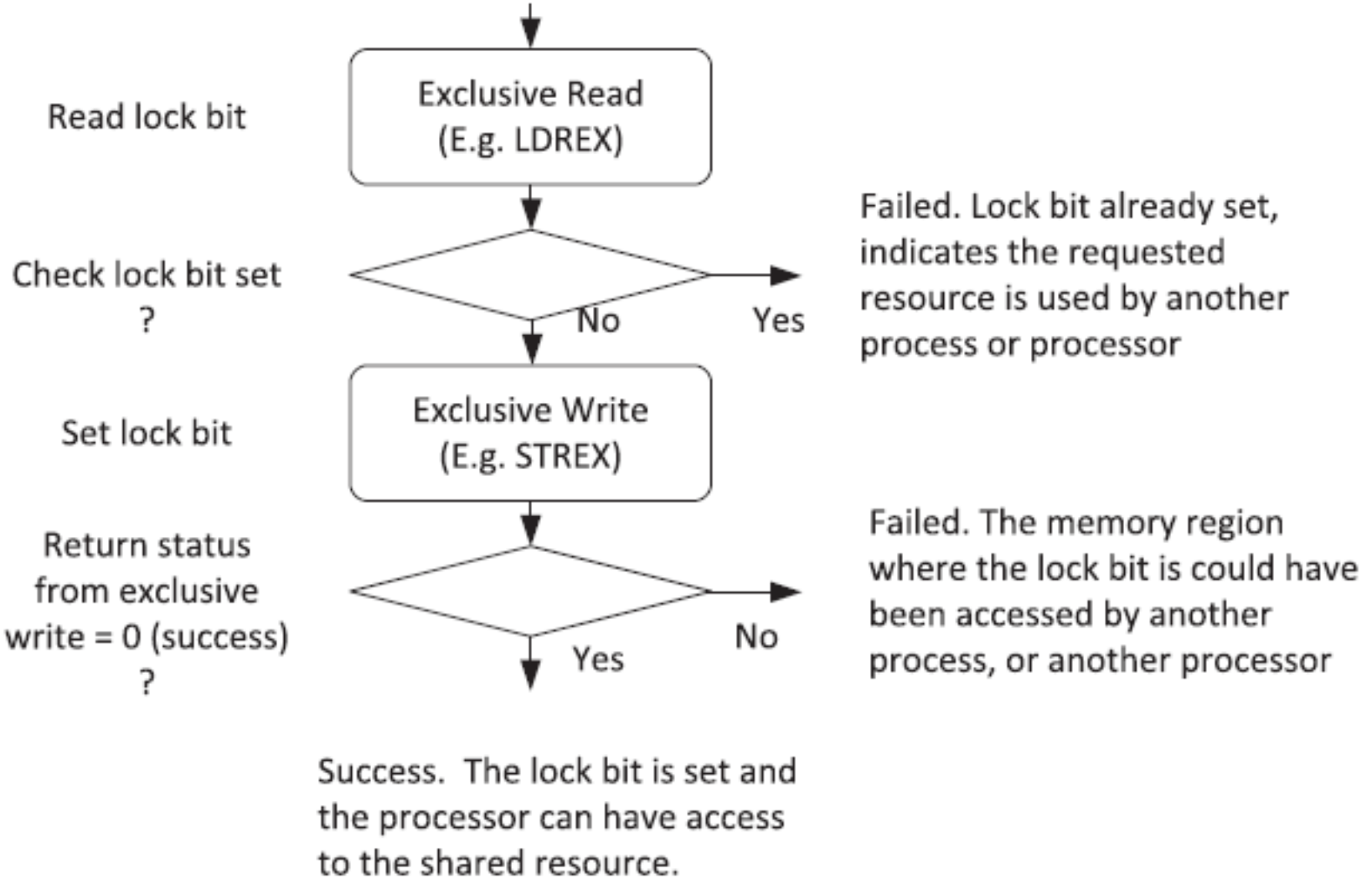
Data alignment and unaligned data access



Memory Access attributes

- **Bufferable**: Write to memory can be carried out by a write buffer while the processor continues on to next instruction execution.
- **Cacheable**: Data obtained from memory read can be copied to a memory cache so that next time it is accessed the value can be obtained from the cache to speed up program execution.
- **Executable**: The processor can fetch and execute program code from this memory region.
- **Sharable**: Data in this memory region could be shared by multiple bus masters. The memory system needs to ensure coherency of data between different bus masters in the shareable memory region.

Exclusive Access in MUTEX semaphore



Memory system in a Microcontroller

In many microcontroller devices, the designs integrate additional memory system features such as:

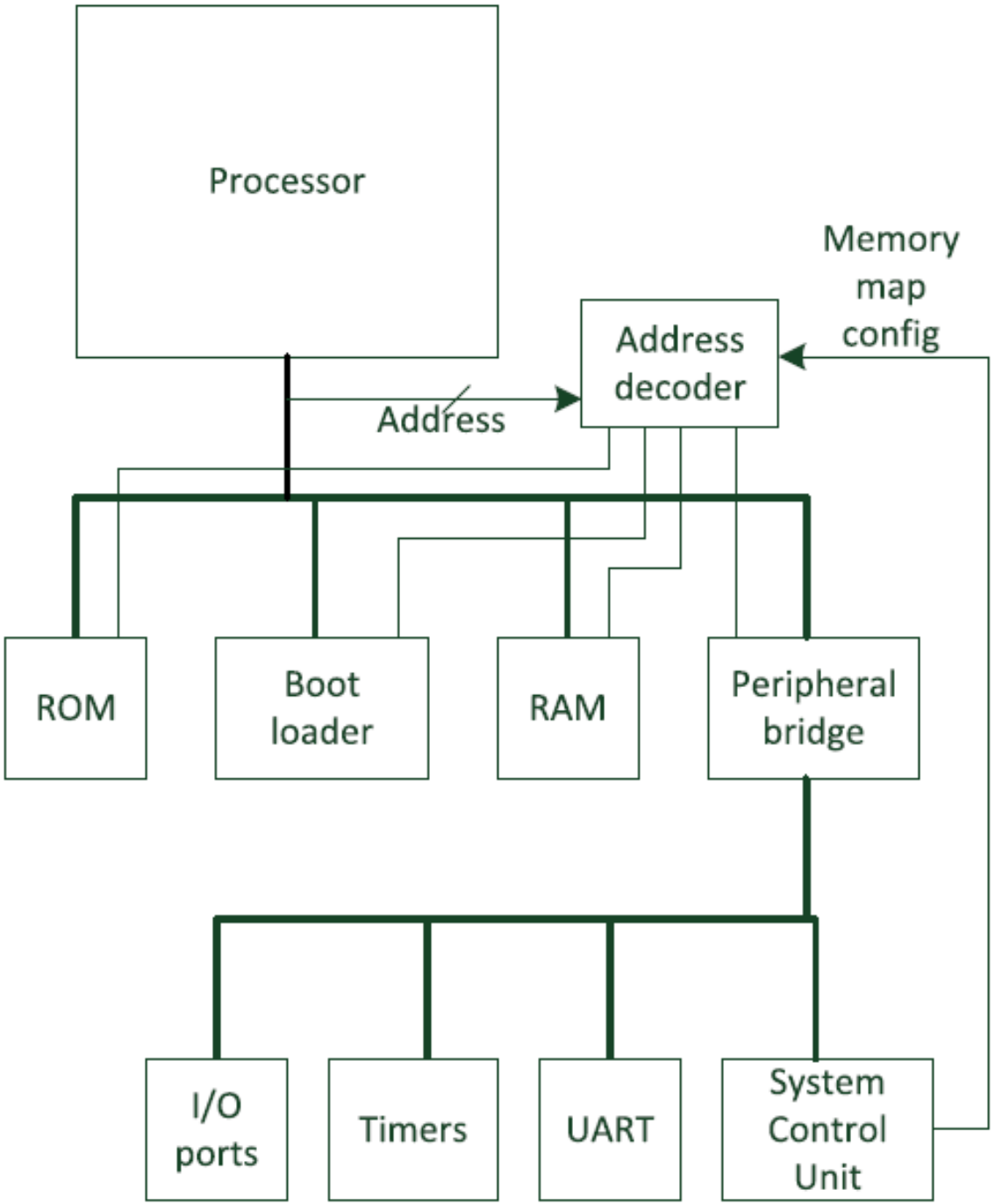
- Boot loader
- Memory remapping
- Memory alias

There are many different reasons why chip designers put a boot loader into the system. For example, to:

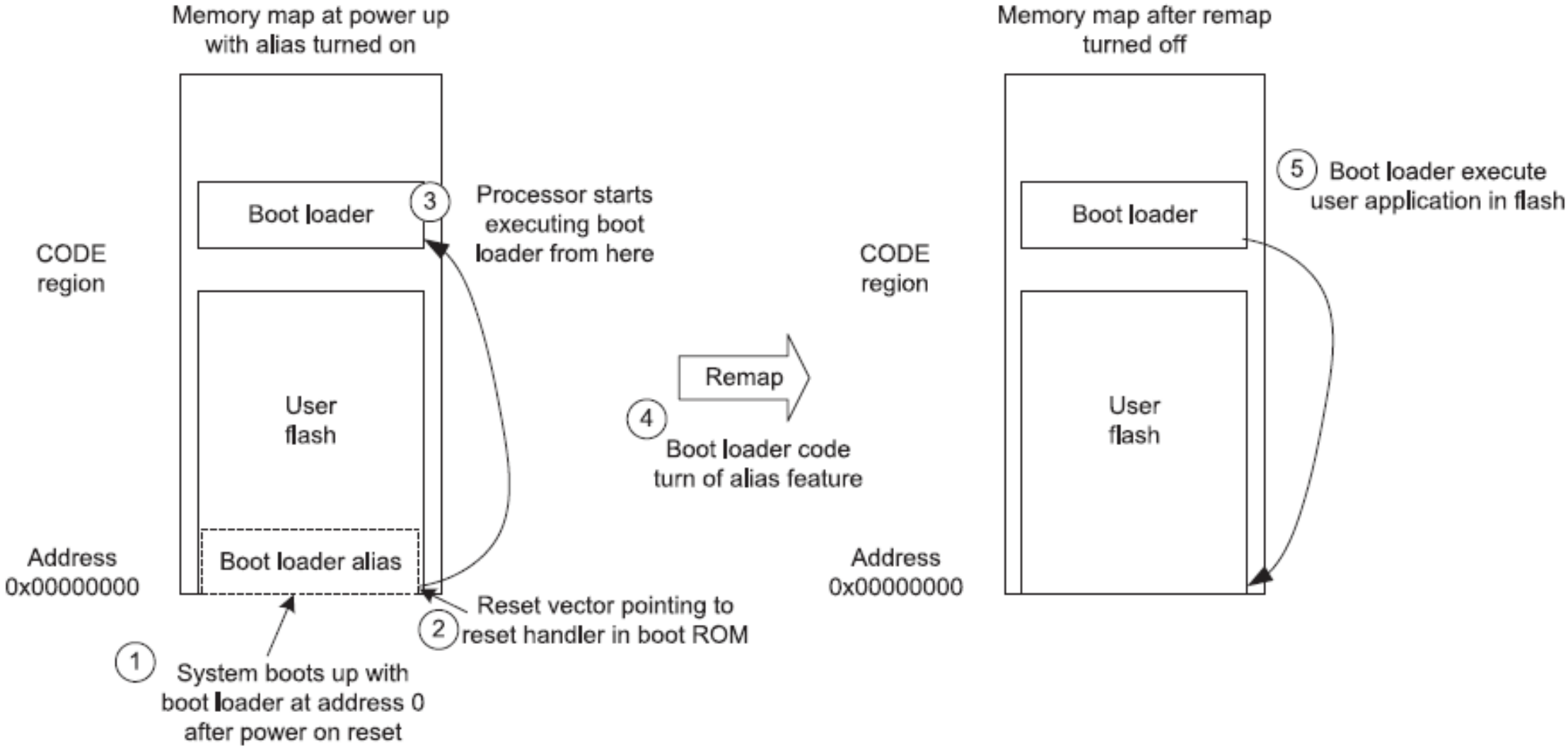
- Provide a flash programming utility, so that you can program the flash using a simple UART interface, or even program some parts of the flash memory dynamically.
- Provide Built-In Self Test (BIST) for the chip.

Bootloaders and Memory Remapping

- For chips with a boot loader ROM, the boot loader is executed when the system is started, so it has to be located in address 0 when the system starts at power up.
- However, the next time the system starts, it might not need to execute the boot loader again and can run the application in the flash directly, so the memory map needs to be changed.
- In order to do this, the address decoder needs to be programmable. A hardware register (e.g., a peripheral register in a system control unit) can be used.
- The operation to switch the memory map is called "Memory Remap." This operation is done by the boot loader.



Memory remap implementation with boot loader



Possibilities of Re-mapping

- Normally the boot loader is accessible from address 0 using a memory address alias, and the alias can be turned off. There are many possible memory system configurations. What is shown in prev. slide is just one of the possibilities.
- In some microcontrollers, memory remapping is not needed as the boot loaders present in address 0 are executed every time the system starts up.
- The vector table is then relocated using the vector table relocation feature provided by the processor, so there is no need to use any remap to handle vector fetches.

Exceptions and Interrupts

CDAC ACTS, Pune

DESIGNING FOR THE FUTURE

Exception Type

Exception Number	Exception Type	Priority	Descriptions
1	Reset	–3 (Highest)	Reset
2	NMI	–2	Non-Maskable Interrupt (NMI), can be generated from on chip peripherals or from external sources.
3	Hard Fault	–1	All fault conditions, if the corresponding fault handler is not enabled
4	MemManage Fault	Programmable	Memory management fault; MPU violation or program execution from address locations with XN (eXecute Never) memory attribute.
5	Bus Fault	Programmable	Bus error; usually occurs when AHB interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access). Can also be caused by other illegal accesses.

6	Usage Fault	Programmable	Exceptions due to program error or trying to access co-processor (the Cortex-M3 and Cortex-M4 processor do not support co-processors).
7–10	Reserved	NA	–
11	SVC	Programmable	SuperVisor Call; usually used in OS environment to allow application tasks to access system services.
12	Debug Monitor	Programmable	Debug monitor; exception for debug events like breakpoints, watchpoints when software based debug solution is used.
13	Reserved	NA	–
14	PendSV	Programmable	Pendable service call; An exception usually used by an OS in processes like context switching.
15	SYSTICK	Programmable	System Tick Timer; Exception generates by a timer peripheral which is included in the processor. This can be used by an OS or can be used as a simple timer peripheral.

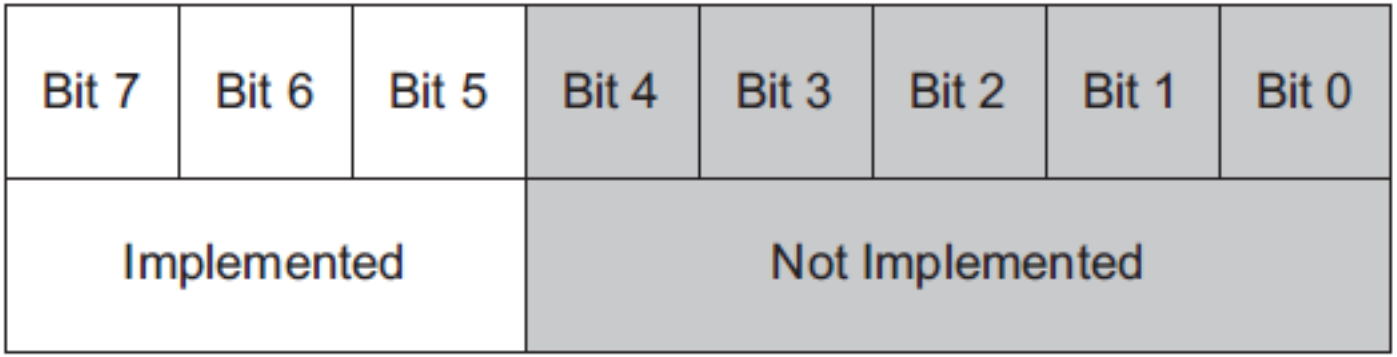
List of Interrupts

Exception Number	Exception Type	Priority	Descriptions
16	Interrupt #0	Programmable	It can be generated from on chip peripherals or from external sources.
17	Interrupt #1	Programmable	
…	…	…	
255	Interrupt #239	Programmable	

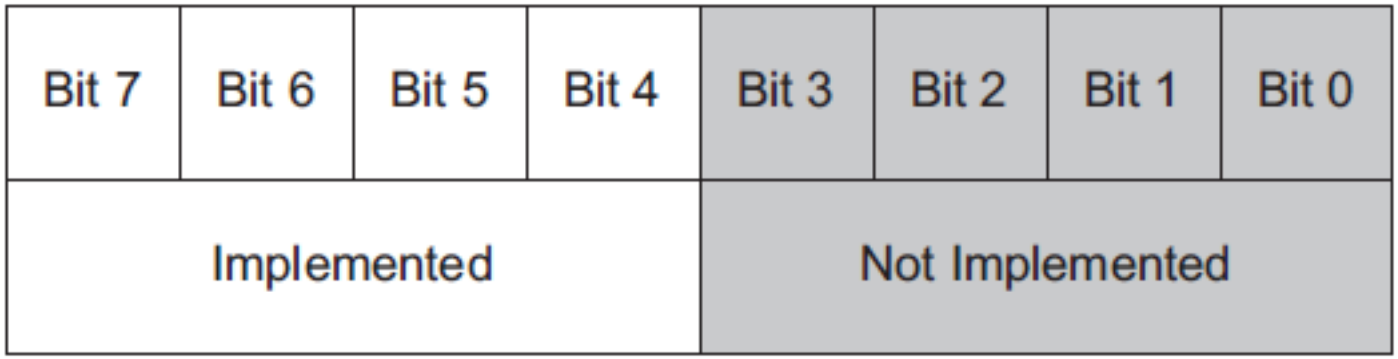
Commonly Used CMSIS-Core Fxns

Functions	Usage
void NVIC_EnableIRQ (IRQn_Type IRQn)	Enable an external interrupt
void NVIC_DisableIRQ (IRQn_Type IRQn)	Disable an external interrupt
void NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)	Set the priority of an interrupt
void __enable_irq(void)	Clear PRIMASK to enable interrupts
void __disable_irq(void)	Set PRIMASK to disable all interrupts
void NVIC_SetPriorityGrouping(uint32_t PriorityGroup)	Set priority grouping configuration

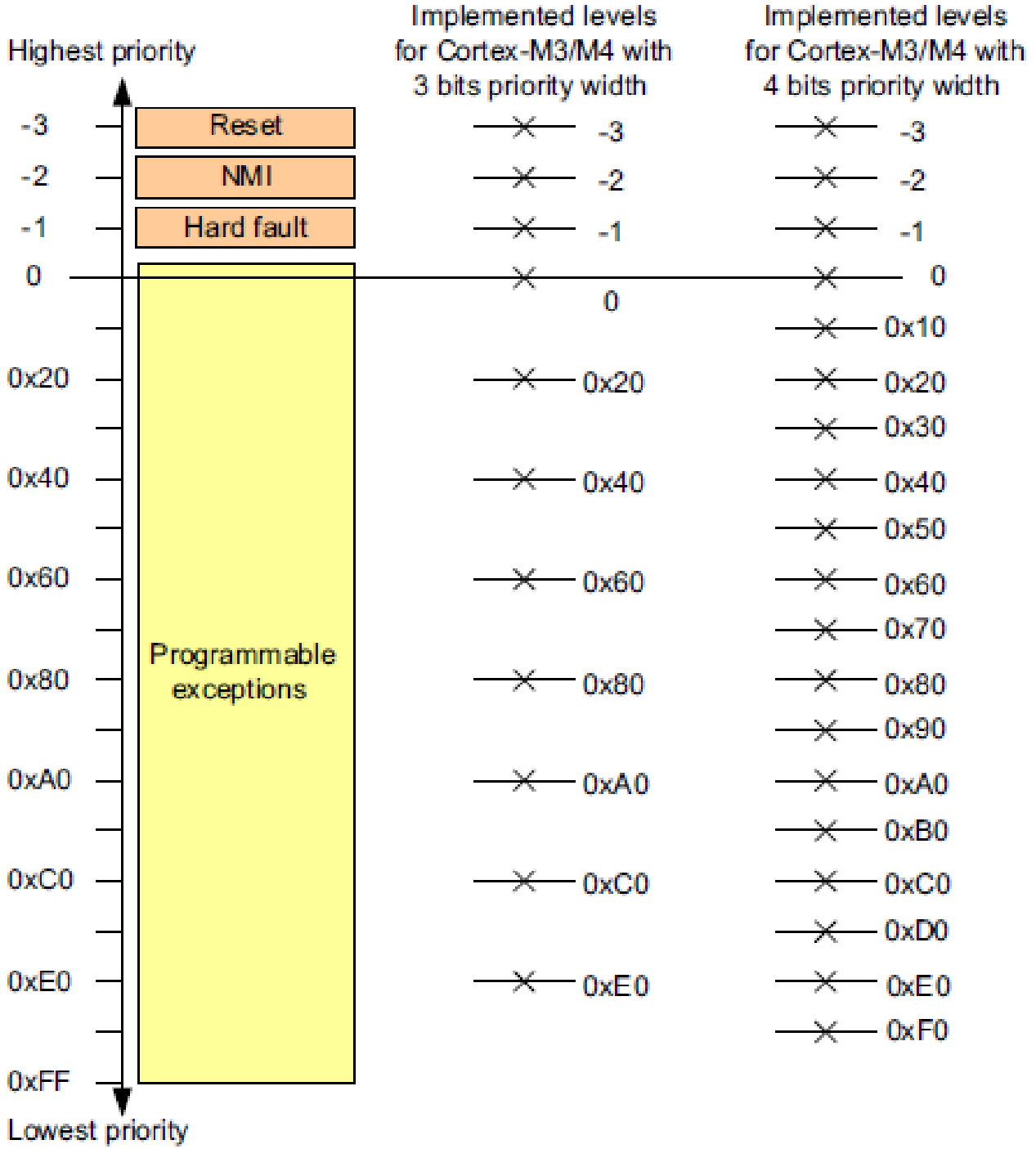
Definition of Priority



A priority-level register with 3 bits implemented (8 programmable priority levels)



A priority-level register with 4 bits implemented (16 programmable priority levels)



Group Priority

- if the priority-level configuration registers are 8-bits wide, there are only 128 pre-emption levels?
- This is because the 8-bit register is further divided into two parts: group priority and sub-priority.
- Using a configuration register in the System Control Block (SCB) called Priority Group
- the priority-level configuration registers for each exception with programmable priority levels is divided into two halves.
- The group priority level defines whether an interrupt can take place when the processor is already running another interrupt handler.
- The sub-priority level is used only when two exceptions with same group-priority level occur at the same time.

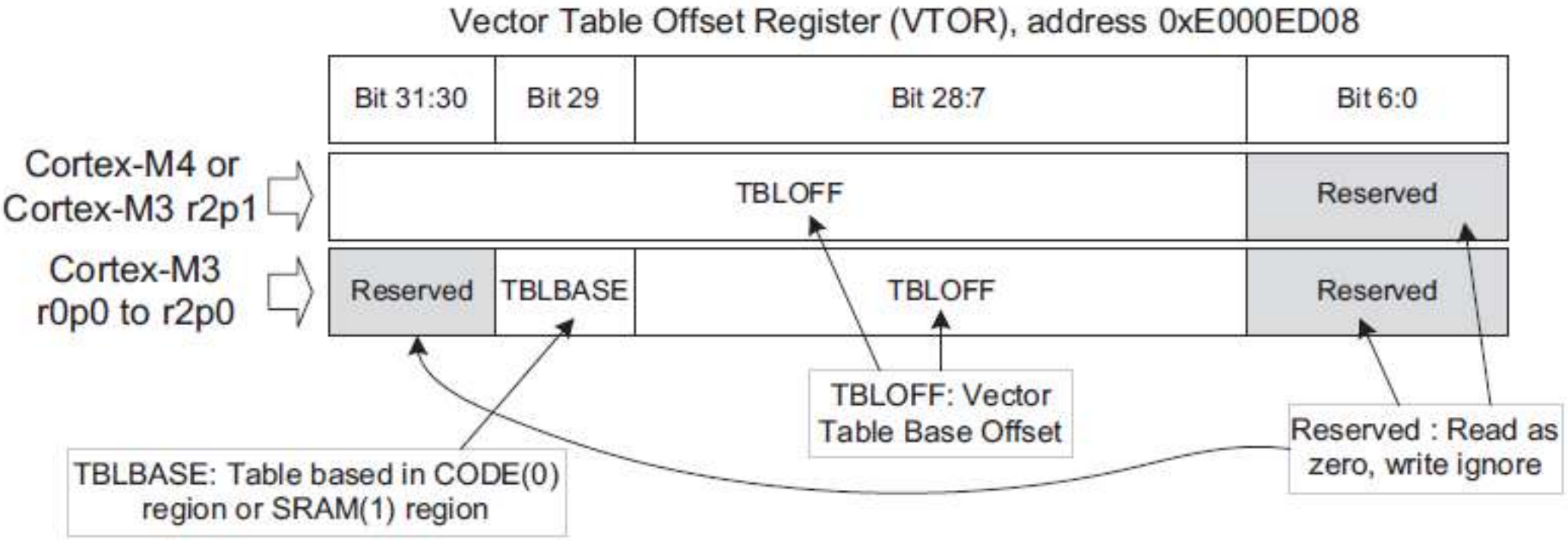
Group Priority

Priority Group	Pre-empt Priority Field	Sub-priority Field
0 (default)	Bit [7:1]	Bit [0]
1	Bit [7:2]	Bit [1:0]
2	Bit [7:3]	Bit [2:0]
3	Bit [7:4]	Bit [3:0]
4	Bit [7:5]	Bit [4:0]
5	Bit [7:6]	Bit [5:0]
6	Bit [7]	Bit [6:0]
7	None	Bit [7:0]

Vector Table and its Relocation

- When the Cortex-M processor accepts an exception request, the processor needs to determine the starting address of the exception handler (or ISR if the exception is an interrupt).
- This information is stored in the vector table in the memory. By default, the vector table starts at memory address 0.
- The vector table is normally defined in the startup codes provided by the microcontroller vendors.
- Usually, the starting address (0x00000000) should be boot memory, and it will usually be either flash memory or ROM devices.
- The Vector Table Relocation feature provides a programmable register called the Vector Table Offset Register (VTOR)

Vector Table Offset Register



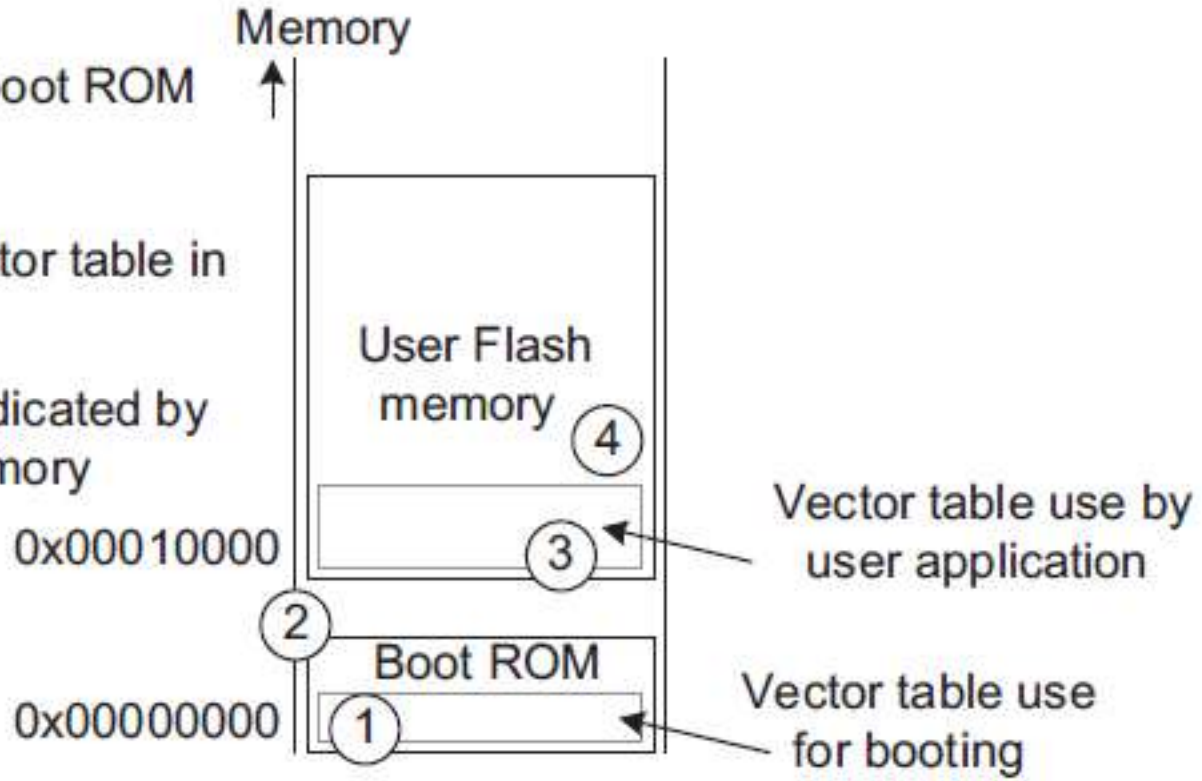
Vector Table Relocation for Boot Rom

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



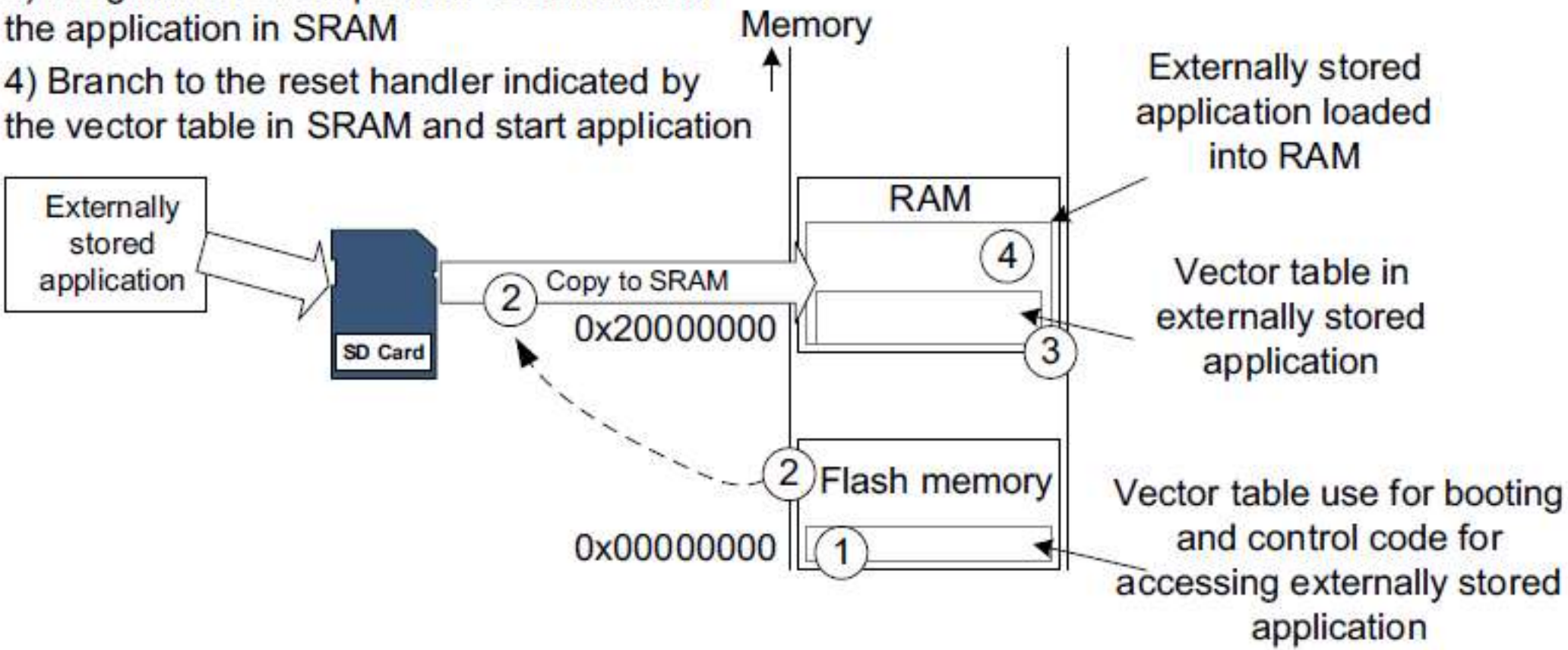
Booting from Pen Drive or SD Card

1) Boot up using vector table in flash memory

2) Initialize hardware and copy externally stored application to RAM

3) Program VTOR to point to vector table in the application in SRAM

4) Branch to the reset handler indicated by the vector table in SRAM and start application



Interrupt Inputs and Pending Behavior

There are various status attributes applicable to each interrupt:

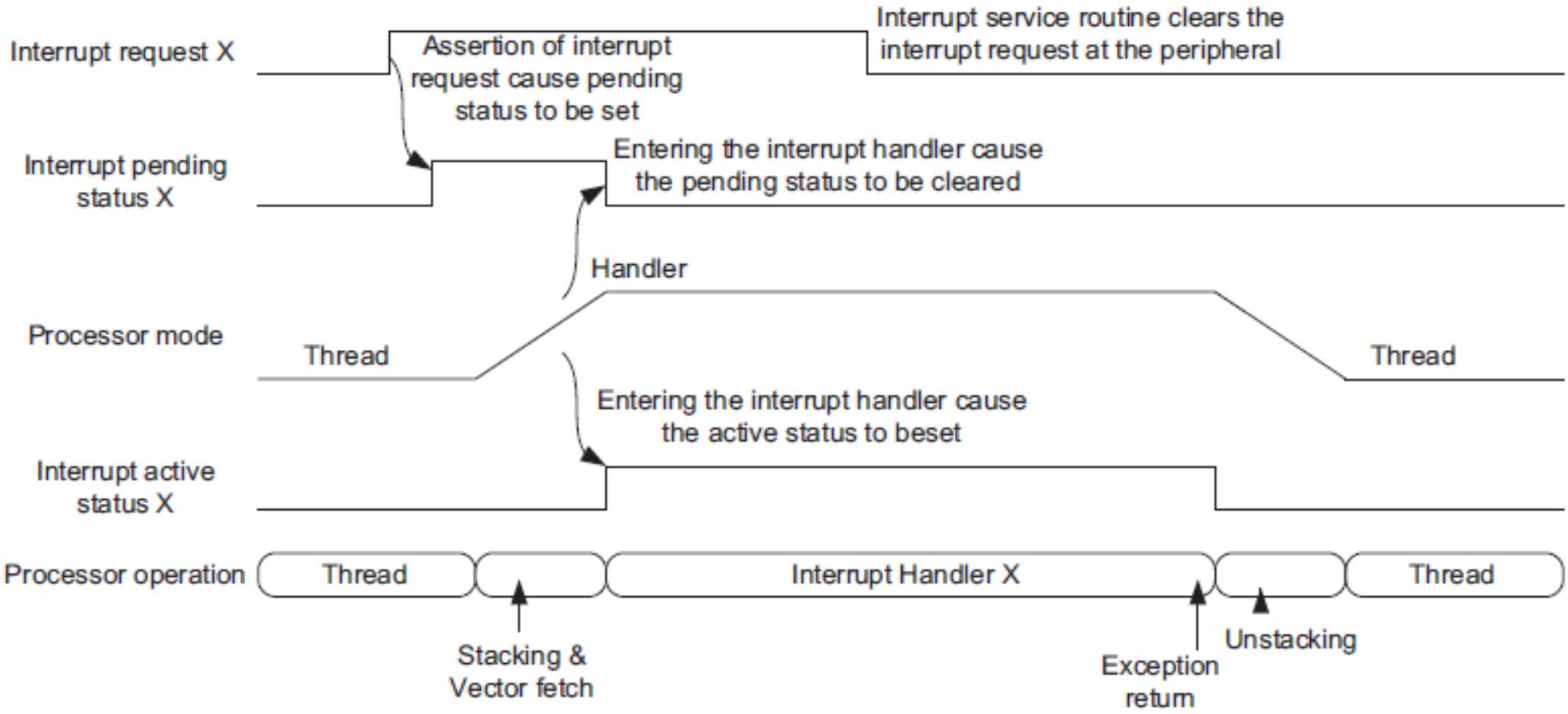
- Each interrupt can either be disabled (default) or enabled
- Each interrupt can either be pending (a request is waiting to be served) or not pending
- Each interrupt can either be in an active (being served) or inactive state

An interrupt request can be accepted

by the processor if:

- The pending status is set,
- The interrupt is enabled, and
- The priority of the interrupt is higher than the current level

Interrupt pending and activation behavior

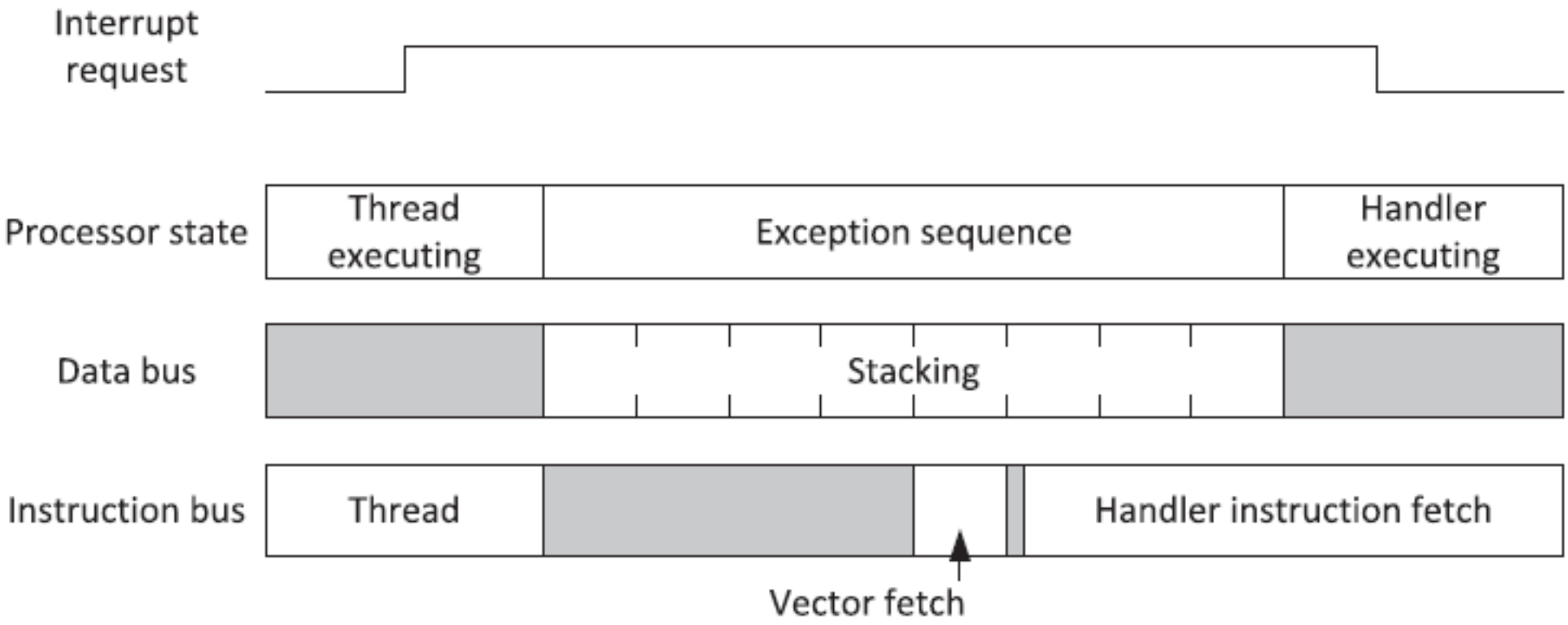


Exceptions Handling In Detail

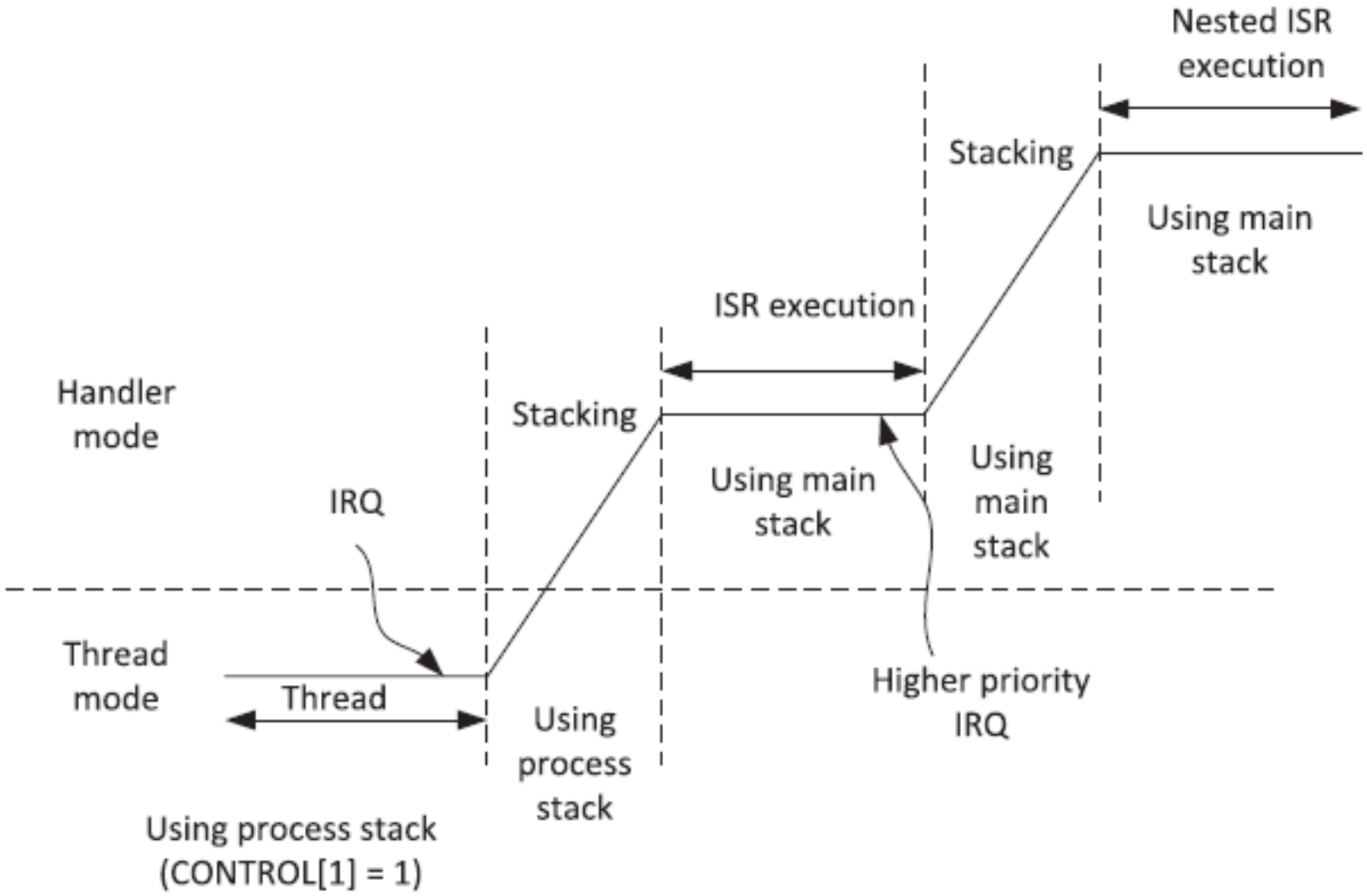
CDAC ACTS, Pune

DESIGNING FOR THE FUTURE

Exception entrance and stacking

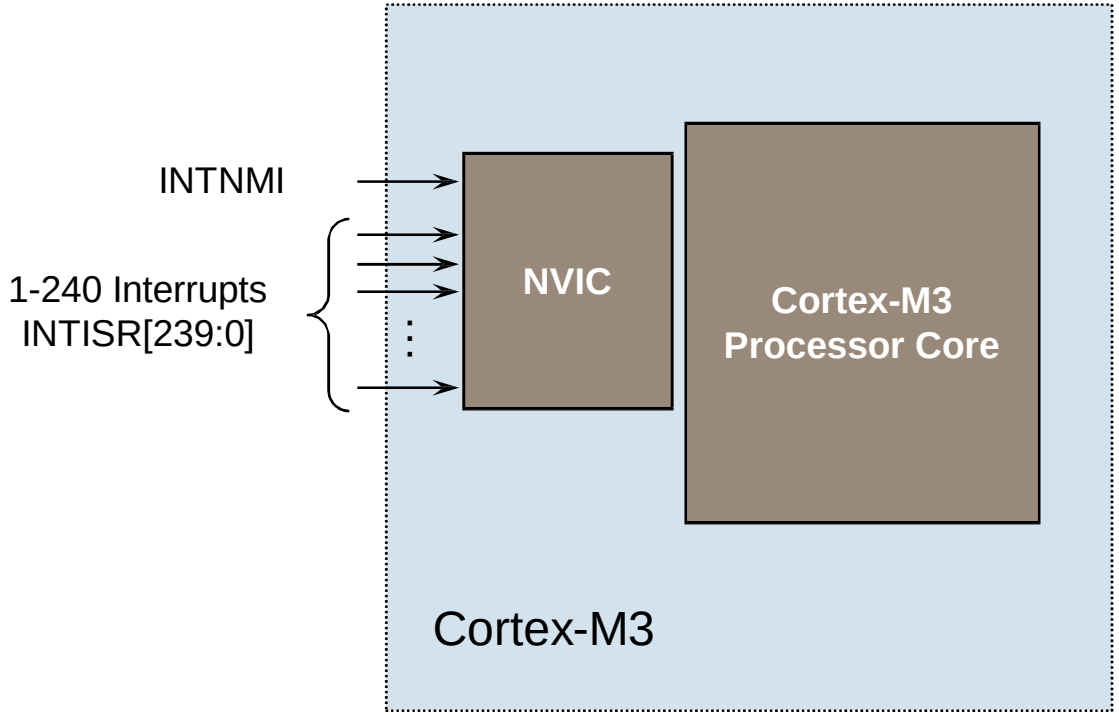


Nested Interrupt Stacking

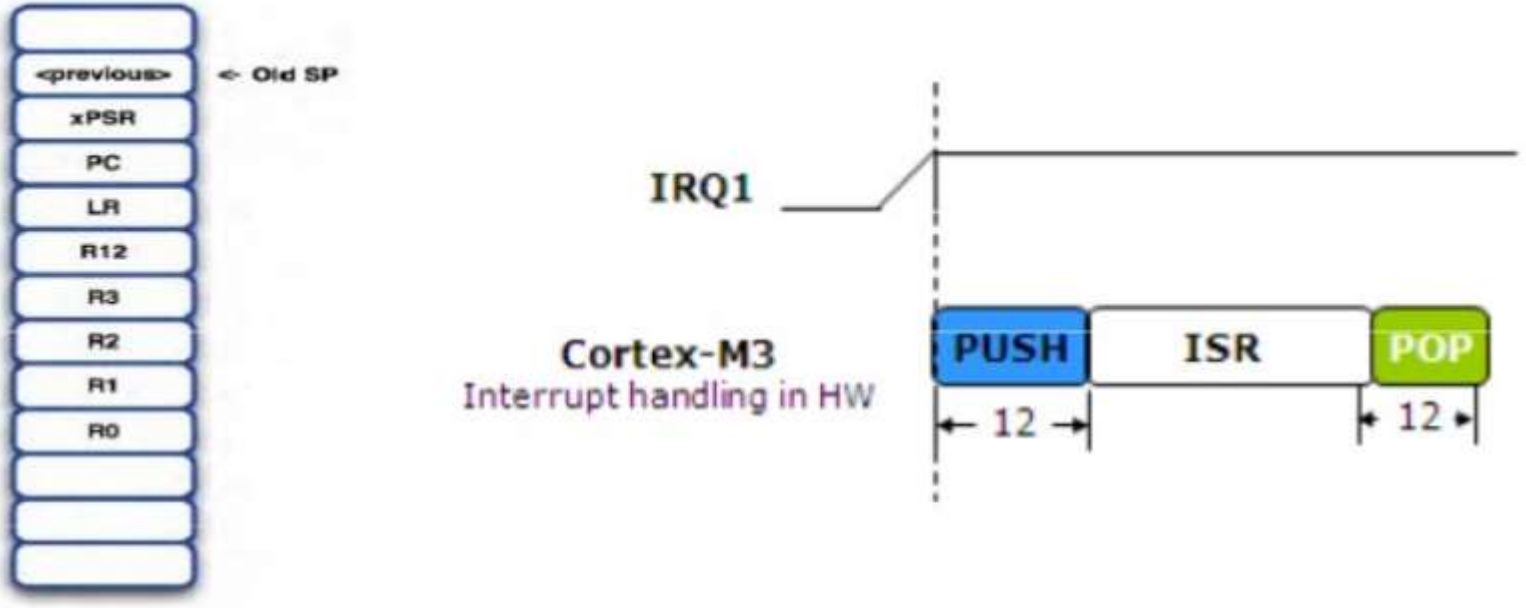


Interrupt Handling

- One Non-Maskable Interrupt (INTNMI) supported
- 1-240 prioritizable interrupts supported
  - Interrupts can be masked
  - Implementation option selects number of interrupts supported
- Nested Vectored Interrupt Controller (NVIC) is tightly coupled with processor core
- Interrupt inputs are active HIGH



NVIC Operations Exception Entry/Exit



When an interrupt is raised by a peripheral, the NVIC will start the Cortex CPU serving the interrupt. As the Cortex CPU enters its interrupt mode, it will push a set of registers onto the stack. Importantly this is done in microcode, so there is no instruction overhead in the application code. While the stack frame is being saved, the starting address of the interrupt service routine is fetched on the instruction bus. Thus the time taken from the interrupt being raised to reaching the first instruction in the interrupt routine is just 12 cycles.

Interrupt Preemption

The stack frame consists of the Program Status Register, the program counter and the link register. This saves the context that the Cortex CPU was running in. In addition registers R0 – R3 are also saved. In the ARM binary interface standard these registers are used for parameter passing, so saving these gives us a set of CPU registers that can be used by the ISR. Finally R12 is also saved; this register is the intracall scratch register. This register is used by any code that runs when function calls are made. For example, if you have enabled stack checking in the compiler, the additional code generated will use R12 if it need a CPU register. When the interrupt ends the process is reversed, the stack frame is restored automatically by microcode and in parallel the return address is fetched, so that the background code can resume execution in 12 cycles.

Interrupt Preemption

As well as being able to handle a single interrupt very quickly, the NVIC is designed to efficiently handle multiple interrupts in a very real time application. The NVIC has several clever methods of handling multiple interrupt sources the minimum delay between interrupts and to ensure that the highest priority interrupt is served first.

The NVIC is also designed to allow high priority interrupts to pre-empt a currently running low priority interrupt. In this case the running interrupt is halted and a new stack frame is saved in the standard 12 cycles after which the high priority interrupt runs. When the high priority interrupt is finished, the stack is automatically POPed and the low priority interrupt can resume execution.

Low Power and System Control Features

CDAC ACTS, Pune

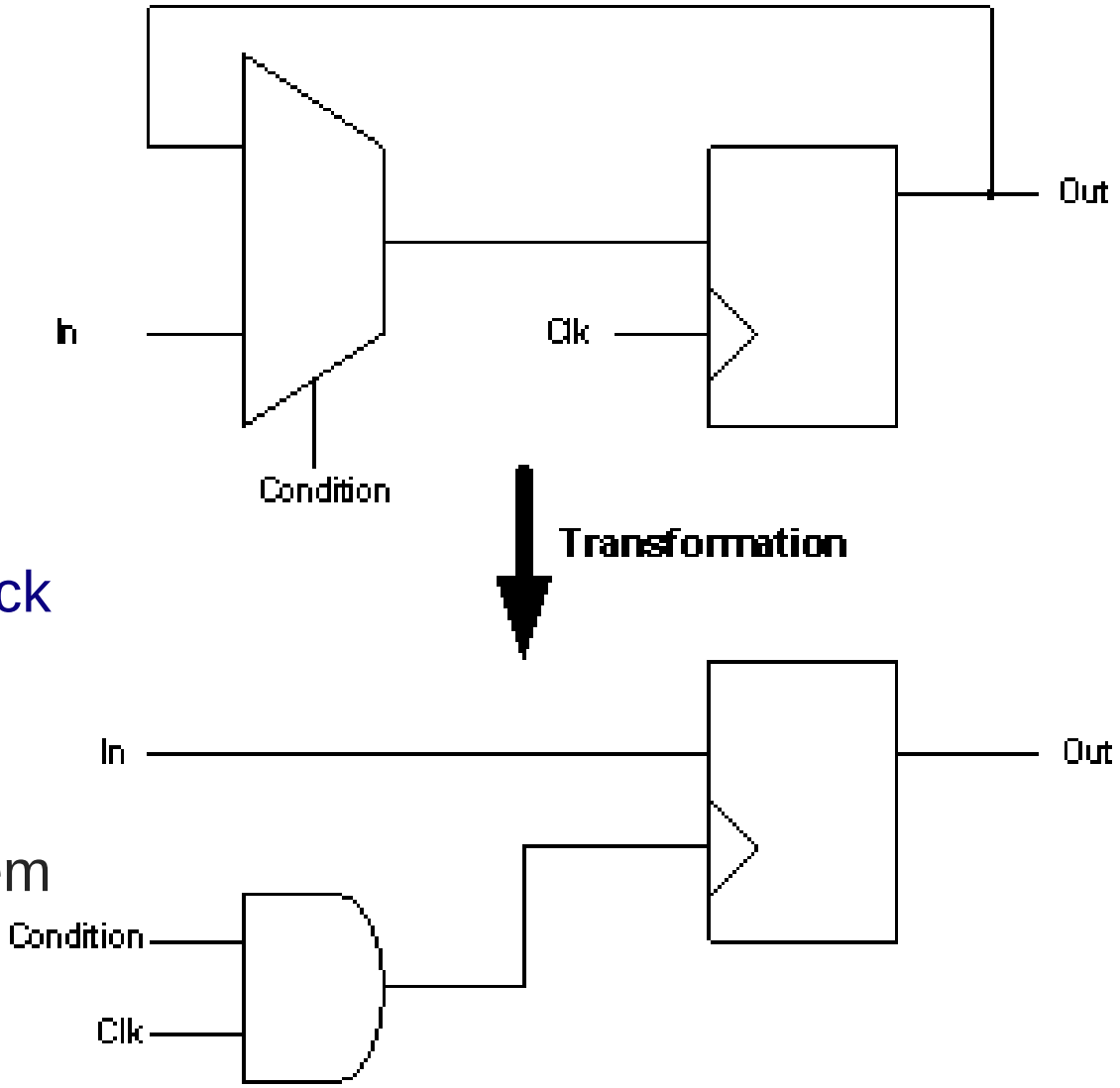
DESIGNING FOR THE FUTURE

Power Management

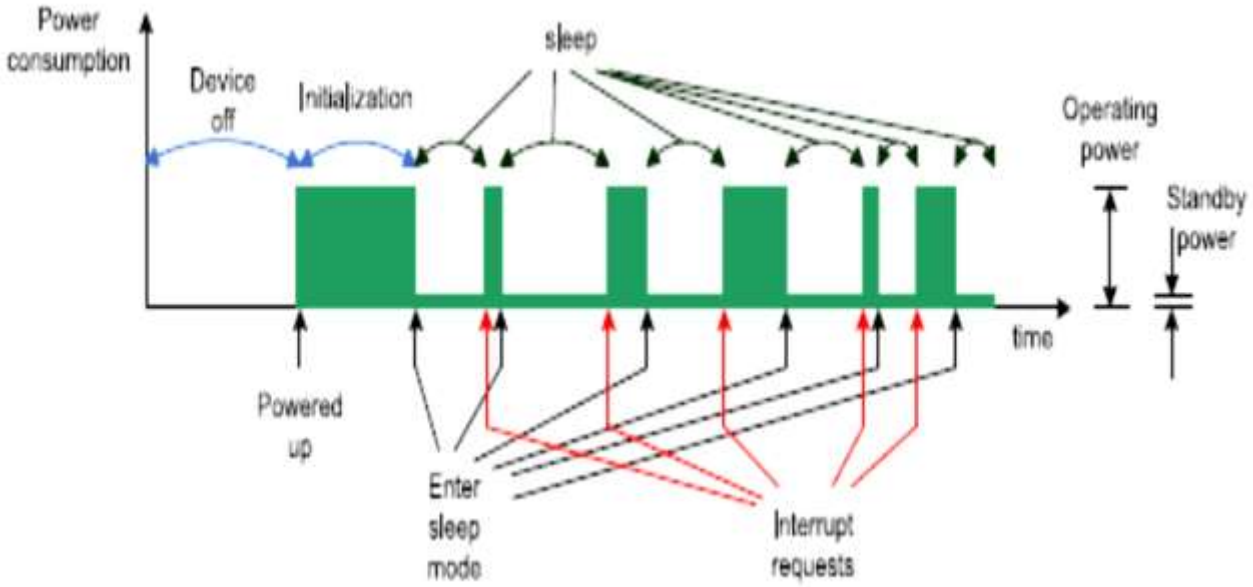
- Multiple sleep modes supported
  - **Controlled by NVIC**
  - **Sleep Now – Wait for Interrupt/Event instructions**
  - **Sleep On Exit – Sleep immediately on return from last ISR**
  - **Deep Sleep**
    - Long duration sleep, so PLL can be stopped
    - Exports additional output signal SLEEPDEEP
- Cortex-M3 system is clock gated in all sleep modes
  - Sleep signal is exported allowing external system to be clock gated also
  - NVIC interrupt Interface stays awake
- Wake-Up Interrupt Controller (WIC)
  - External wake-up detector allows Cortex-M3 to be fully powered down
  - Effective with State-Retention / Power Gating (SRPG) methodology

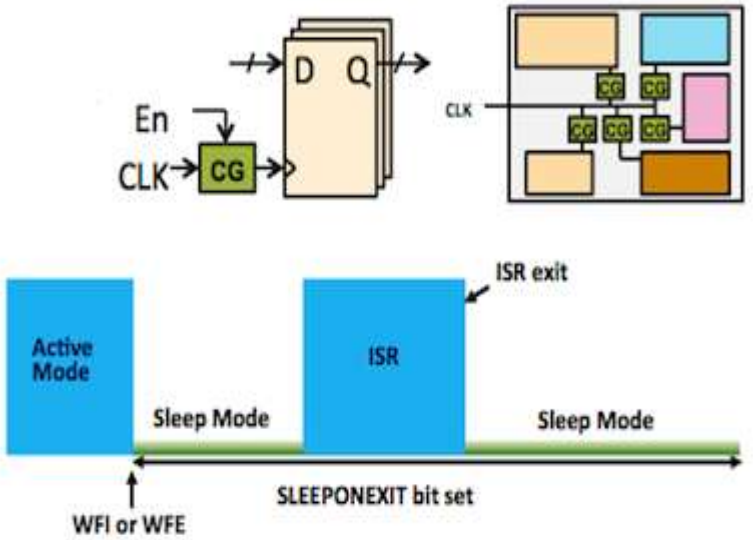
Clock Gating

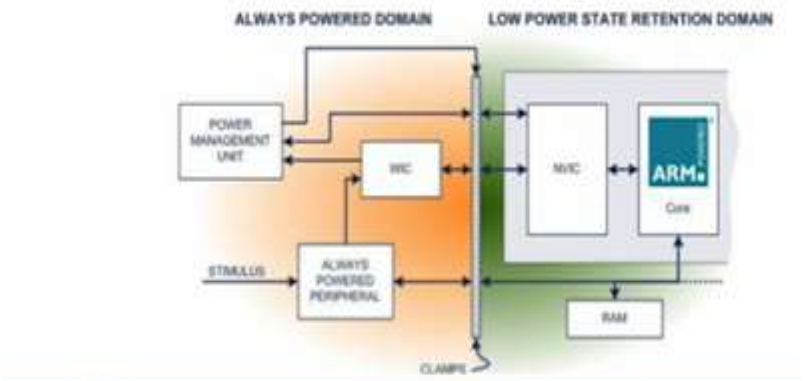
- **Clock gating** is a popular technique used in many synchronous circuits for reducing dynamic power dissipation.
- Clock gating saves power by adding more logic to a circuit to prune the clock tree.
- Pruning the clock disables portions of the circuitry so that the flip-flops in them do not have to switch states because Switching states consumes power.
- When not being switched, the switching power consumption goes to zero, and only leakage currents are incurred.



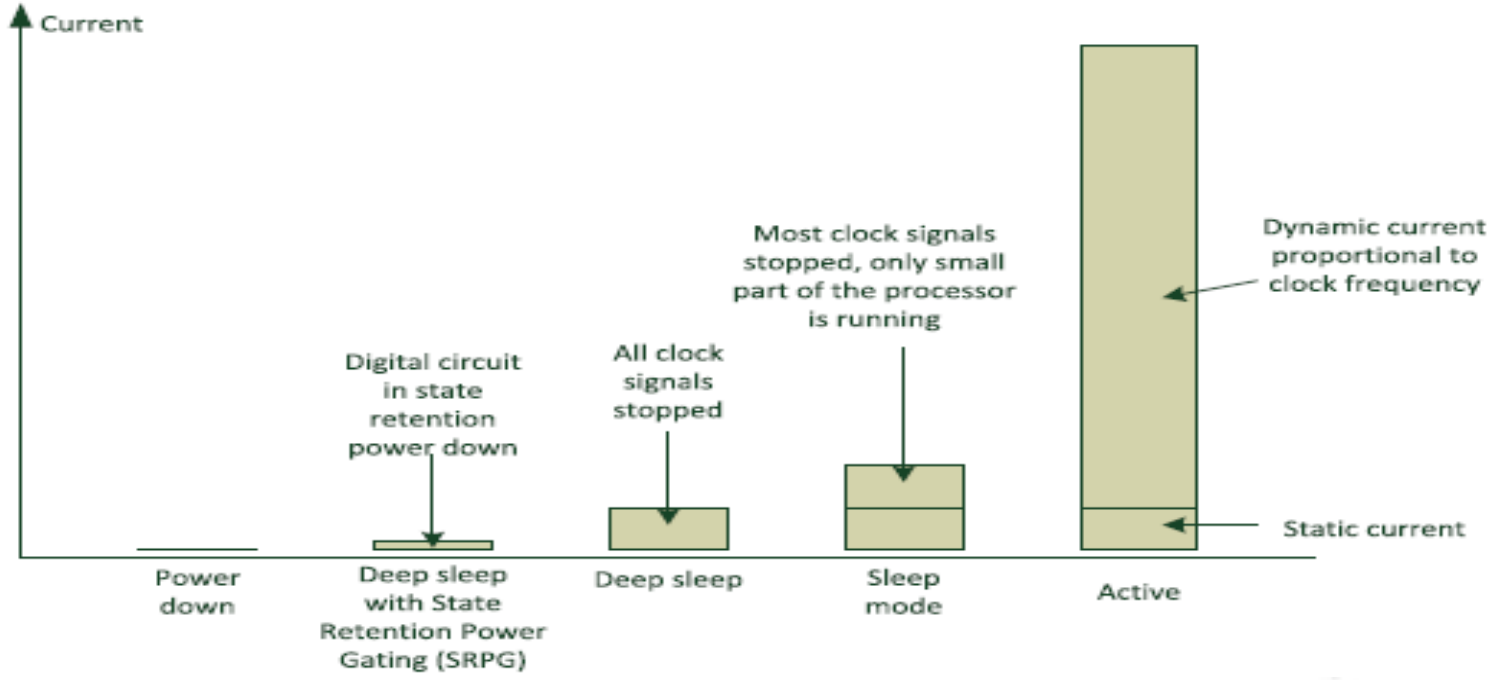
Activities in an Interrupt-Driven Application

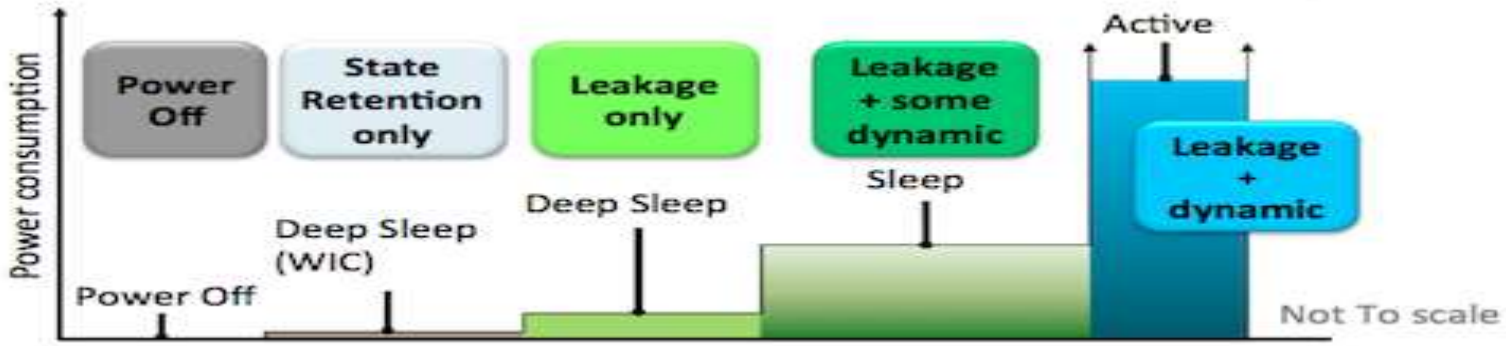






Various Power Modes

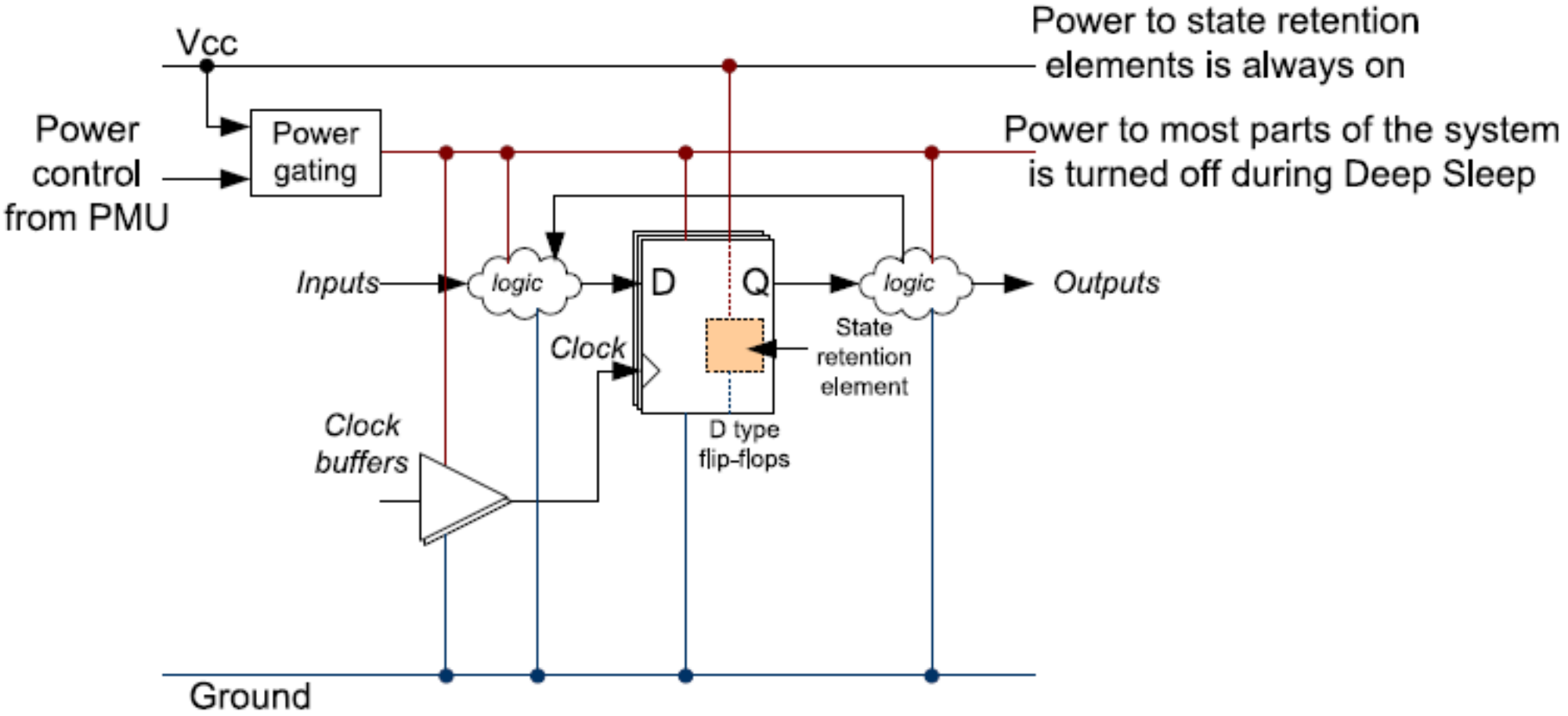




State Retention Power Gating

- In some designs, advanced power-saving techniques called State Retention Power Gating (SRPG) can be used to reduce the leakage current of the chip by a wide margin.
- In SRPG designs, the registers (often called flip-flops in IC design terminology) have a separate power supply for state retention elements inside the registers .
- When the system is in Deep Sleep mode, the normal power supply can be turned off, leaving only the power to the state retention elements on.
- The leakage in this type of design is greatly reduced because the combinatorial logic, clock buffers, and most parts of the registers are powered down.

SRPG



The Cortex microcontroller software interface standard (CMSIS)

- CMSIS was developed by ARM to allow microcontroller and software vendors to use a consistent software infrastructure to develop software solutions for Cortex-M microcontrollers.
- Many software products for Cortex-M microcontrollers are CMSIS-compliant. Currently the Cortex-M microcontroller market comprises:
- More than 15 microcontroller vendors shipping Cortex-M microcontroller products, with some other silicon vendors providing Cortex-M based FPGA and ASICs
- More than 10 toolchain vendors
- More than 30 embedded operating systems
- Additional Cortex-M middleware software providers for codecs, communication protocol stacks, etc.

With such a large ecosystem, some form of standardization of the way the software infrastructure works becomes necessary to ensure software compatibility with various development tools and between different software solutions.

CMSIS

To Increase the interoperability of various software components , ARM worked with various microcontroller vendors, tools vendors, and software solution providers to develop CMSIS, a software framework covering most Cortex-M processors and Cortex-M microcontroller products.

The aims of CMSIS include:

- **Enhanced software reusability** - makes it easier to reuse software code in different Cortex-M projects, reducing time to market and verification efforts.
- **Enhanced software compatibility** - by having a consistent software infrastructure (e.g., API for processor core access functions, system initialization method, common style for defining peripherals), software from various sources can work together, reducing the risk in integration.
- **Easy to learn** - the CMSIS allows easy access to processor core features from the C language. In addition, once you learn to use one Cortex-M microcontroller product, starting to use another Cortex-M product is much easier because of the consistency in software setup.
- **Toolchain independent** - CMSIS-compliant device drivers can be used with various compilation tools, providing much greater freedom.
- **Openness** - the source code for CMSIS core files can be downloaded and accessed by everyone, and everyone can develop software products with CMSIS.

CMSIS Projects

- CMSIS-Core (Cortex-M processor support)
- CMSIS-DSP library
- CMSIS-SVD - the CMSIS System View Description
- CMSIS-RTOS - the CMSIS-RTOS is an API specification for embedded OS
- CMSIS-DAP - the CMSIS-DAP (Debug Access Port)

Standardization in CMSIS-Core

- Standardized access functions to access processor's features - These include various functions for interrupt control using NVIC, and functions for accessing special registers in the processors.
- Standardized functions for system initialization - Most modern feature-rich microcontroller products require some configuration of clock circuitry and power management registers before the application starts. In CMSIS-compliant device-driver libraries, these configuration steps are placed in a function called "SystemInit()." However, having a standardized function name and a standardized location where this function can be found makes it much easier for a designer to pick up and start using a new Cortex-M microcontroller device.
- Standardized software variables for clock speed information - This might not be obvious, but often our application code does need to know what clock frequency the system is running at. For example, such information might be needed for setting up the baud rate divider in a UART, or to initialize the SysTick timer for an embedded OS. A software variable called "SystemCoreClock" is defined in the CMSIS-Core.

Organization of CMSIS-Core

In a general sense, we can define the CMSIS into multiple layers:

- **Core Peripheral Access Layer -** Name definitions, address definitions, and helper functions to access core registers and core peripherals. This is processor specific and is provided by ARM.
- **Device Peripheral Access Layer -** Name definitions, address definitions of peripheral registers, as well as system implementations including interrupt assignments, exception vector definitions, etc. This is device specific (note: multiple devices from the same vendor might use the same file set).
- **Access Functions for Peripherals -** The driver code for peripheral accesses. This is vendor specific and is optional. You can choose to develop your application using the peripheral driver code provided by the microcontroller vendor, or you can program the peripherals directly if you prefer.

There is also a proposed additional layer for peripheral accesses:

Middleware Access Layer - This layer does not exist in current version of CMSIS. The idea is to develop a set of APIs for interfacing common peripherals such as UART, SPI, and Ethernet. If this layer exists, developers of middleware can develop their applications based on this layer to allow software to be ported between devices easily.

CMSIS-Core structure