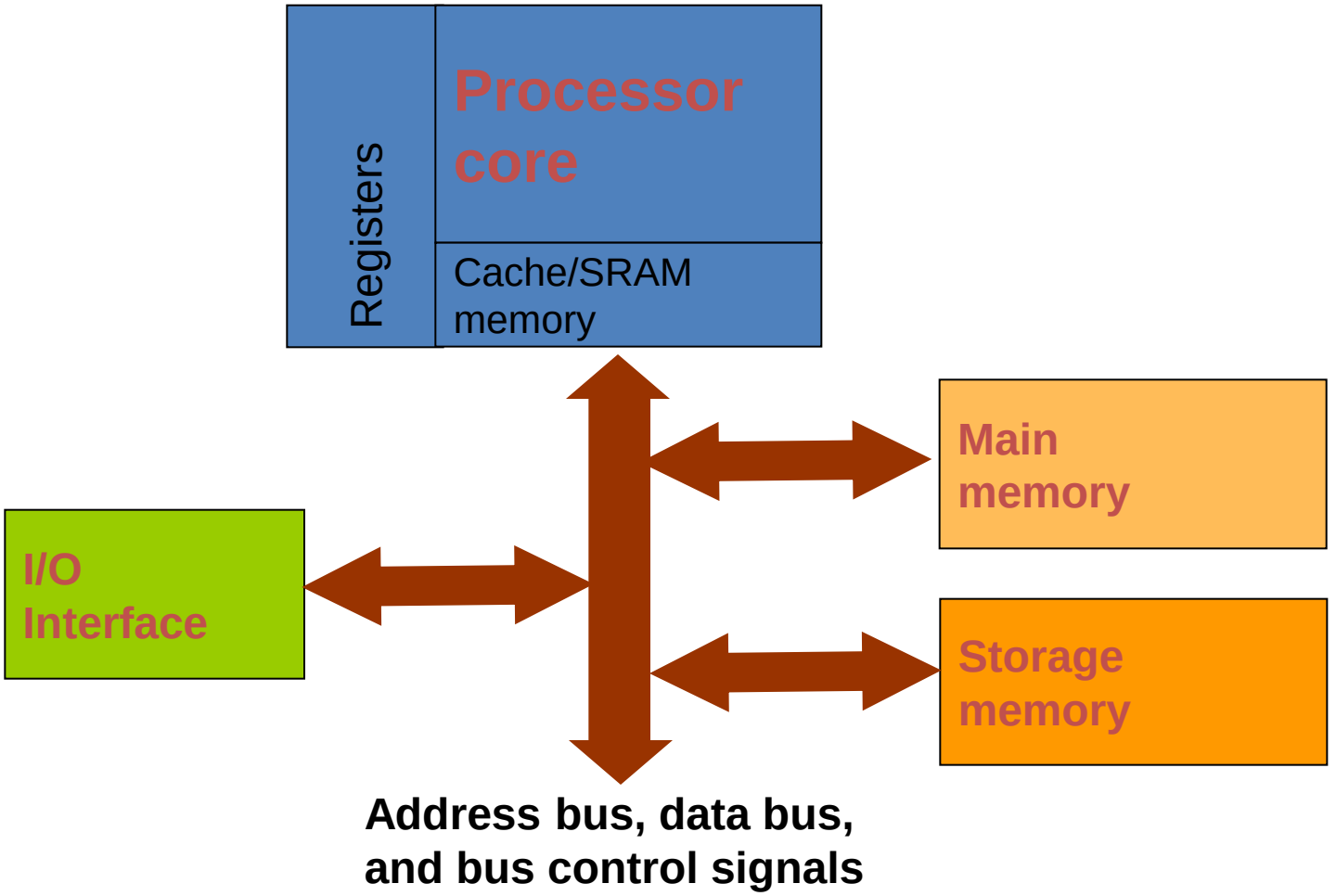
# ARM Architecture

# The RISC Philosophy

- Original RISC design (e.g. MIPS)
  - aims for high performance through
    - reduced number of instruction classes
    - large general-purpose register set
    - load-store architecture
    - fixed length instructions
    - pipelines
  - enables simpler hardware, therefore enabling it to scale to higher operating frequencies

# A Processor Architecture Refresher

# Basic Processor-Based System



# System Components

- The basic components:
  - Processor with its associated temporary memory (registers and cache if available) for code execution
  - Main memory and secondary memory where code and data are temporarily and permanently stored
  - Input and output modules that provide interfaces between the processor and the user
- Connected through an interface bus that consists of Address, Data, and Control signals
  - e.g., AMBA bus for the ARM-based processor

# Memory Hierarchy

- A typical processor is supported by:
  - on-board main memory (e.g. SDRAM up to GB)
  - on-chip or on-die cache memory (e.g. SRAM KB to MB)
  - on-die registers
- Some processors also provide general purpose on-chip
  - SRAM (e.g. embedded processor) which may be configured as SRAM/Cache combination (e.g. TI's DSP)
- Typically, a processor also utilizes secondary non-volatile memory
  - for permanent code and data storage like Flash-based memory and hard disks

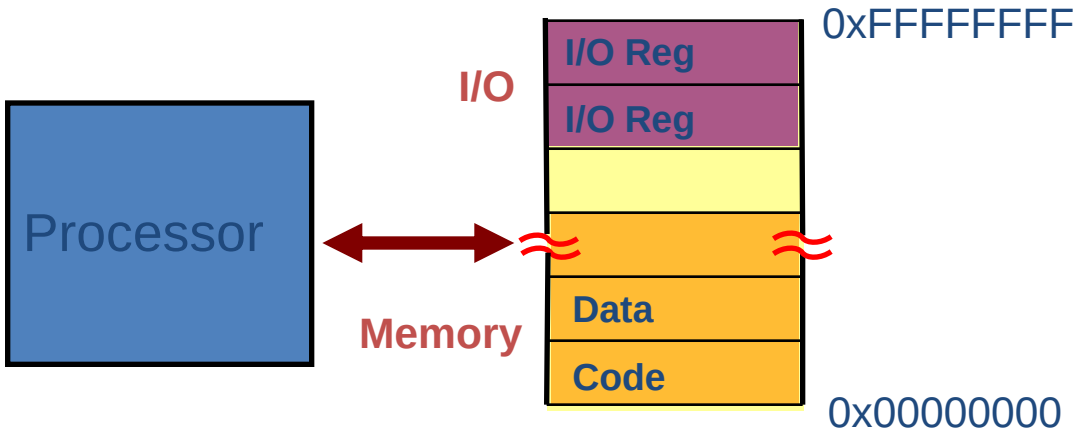
# Address Space

- The address space of a processor depends on its address decoding mechanism
  - Its size will depend on the number of address bits used
- Depending on the processor design, there may be two types of address space
  - one is used by normal memory access
  - another one is reserved for I/O peripheral registers (control, status, and data)
  - need extra control signal or special means of accessing the alternate address space

# Address Space (con't)

- Refer to the range of addresses that can be accessed by the processor, determined by the number of address bit utilized in the processor architecture

- Some processor families (e.g. ARM) utilize only one address space for both memory and I/O devices
  - i.e. everything is mapped in the same address space

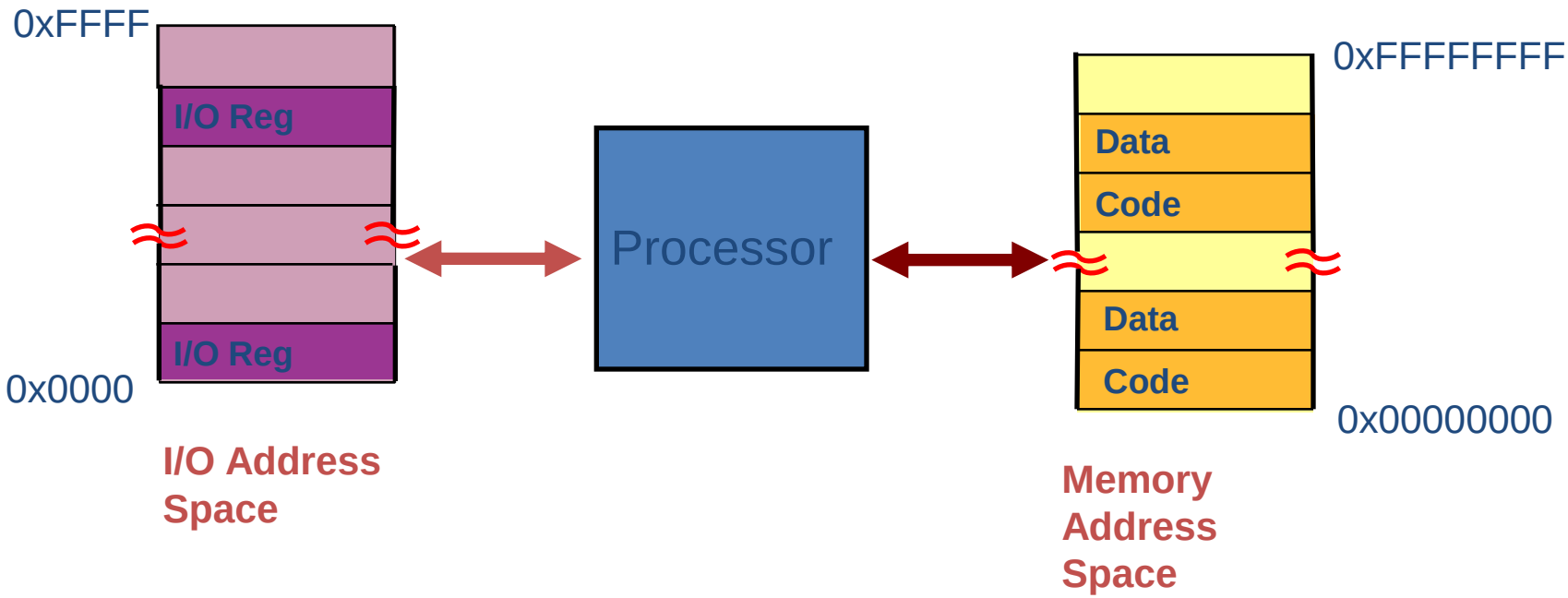


# Memory Mapped vs I/O Mapped

Some processor families have two address spaces.

E.g., for the x86 processor, memory and I/O devices can be mapped in two different address spaces:

- memory address space and I/O address space



# Memory System Architectures

- Two types of information are found in a typical program code:

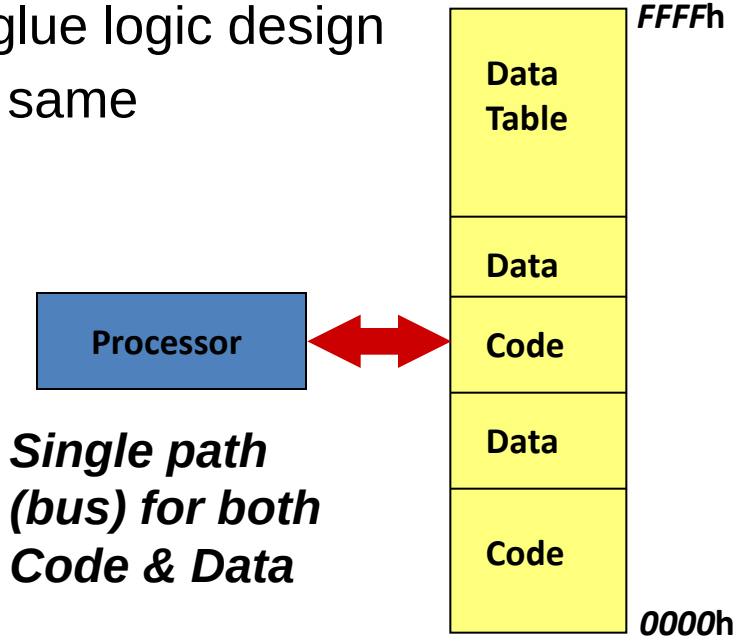
i. Instruction codes for execution
ii. Data that is used by the instruction codes

- Two classes of memory systems designed to store the information:

i. von Neumann architecture
ii. Harvard architecture

# von Neumann Architecture

- The von Neumann architecture utilizes only one memory bus for both instruction fetching and data access
  - simplifies the hardware and glue logic design
  - code and data located in the same address space

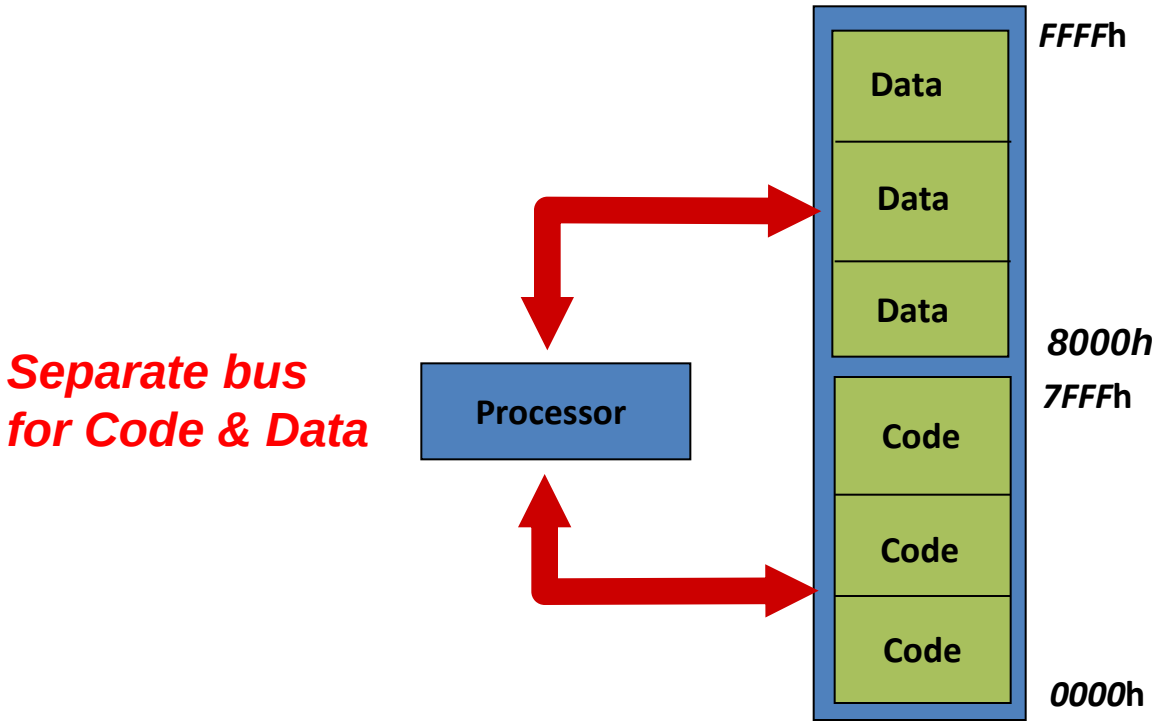


# von Neumann Features

- Single memory interface bus
  - simplifies the hardware and glue logic design
- More efficient use of memory
  - code and data can reside in the same physical memory chip
- More flexible programming style
  - e.g., can include self-modified code
- But data may overwrite code (e.g. due to program bug)
  - need memory protection (e.g. hardware-based MPU)
- Bottleneck in code and data transfer
  - only one memory bus for both data and code fetching

# Harvard Architecture

- The Harvard architecture utilizes separate instruction bus and data bus
  - code and data may still share the same memory space

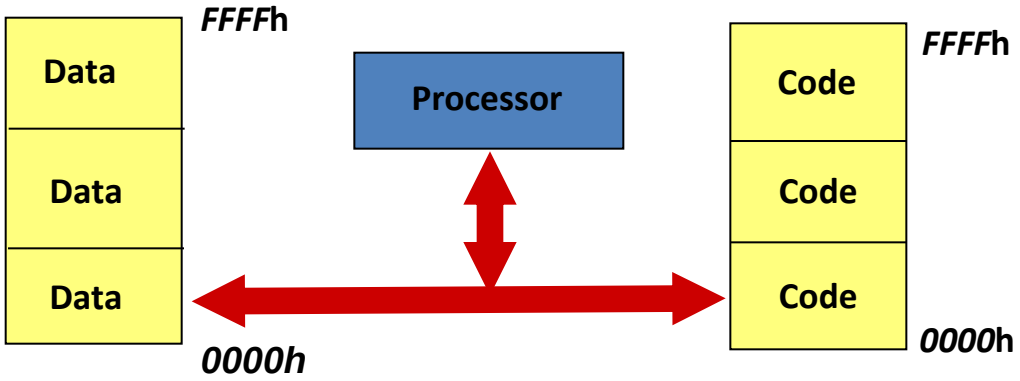


# Harvard Features

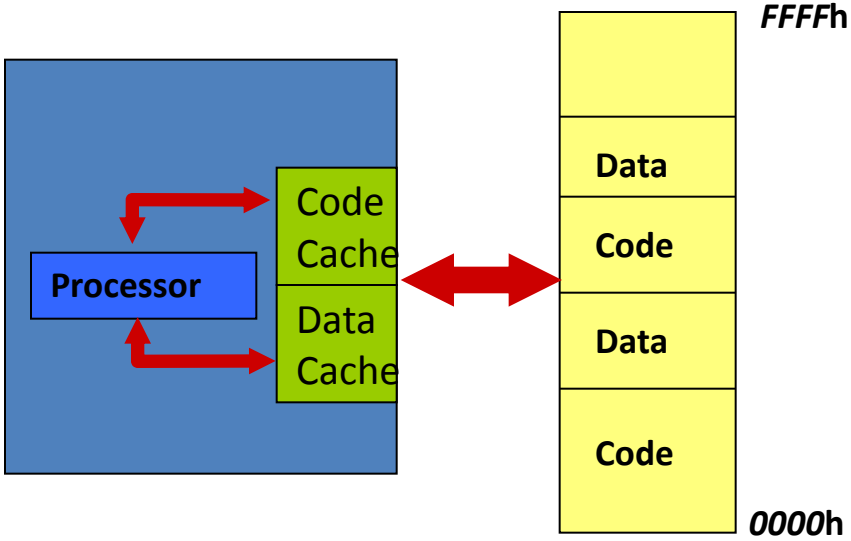
- Separate instruction and data buses
  - allow code and data access at the same time which gives improved performance
  - provide better support for instruction pipeline operations and shorter instruction execution time
  - allow different sizes of data and instructions to be used which results in more flexibility
  - do not incur any code corruption by data which makes the operations more robust
- But more sophisticated hardware glue logic is required to support multiple interface buses
- Cortex-M3 core is based on the Harvard architecture with separate buses for instructions and data

# Architecture Variations

Independent data and code memory but with one shared bus (e.g. 8051)




Two separate internal bus for code & data (e.g. ARM9)

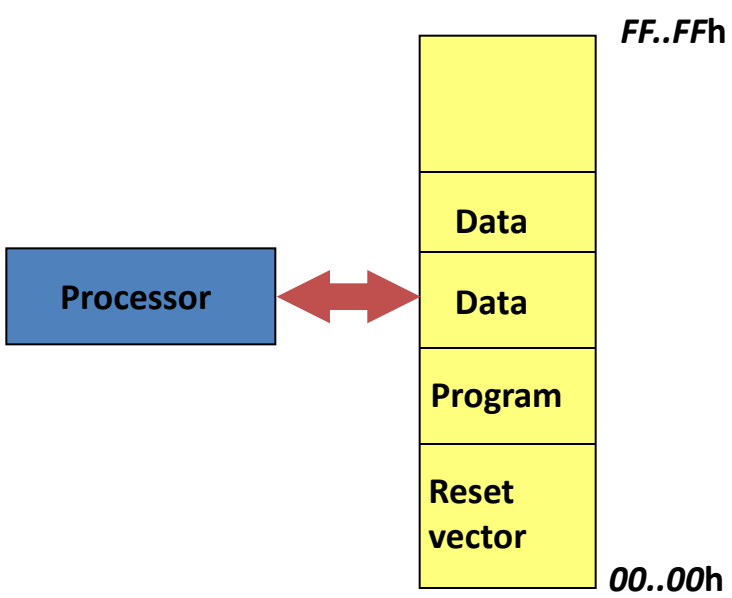


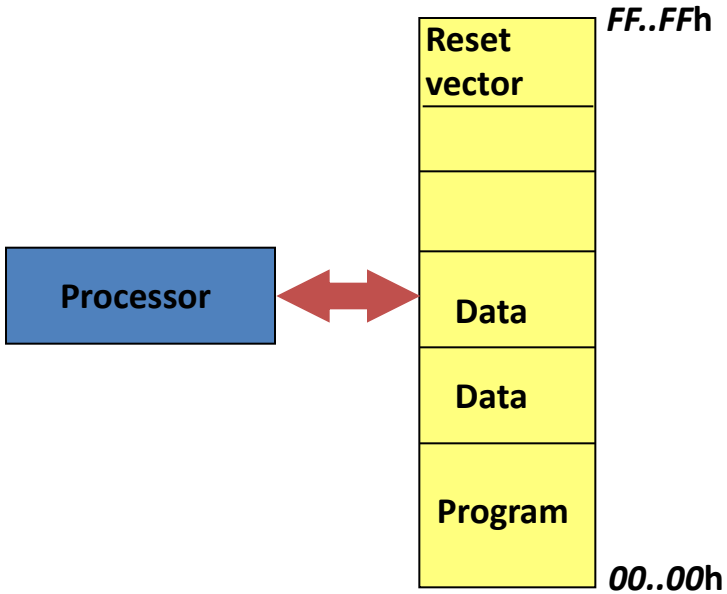
# Top Boot and Bottom Boot

Different processor families use different locations for their reset vectors for boot-up.

Examples:

- x86 processors boot up from the top of the memory space
- ARM processors boot up from the bottom of its memory space



# Processor ‘Size’

- Processor size is described in terms of ‘bits’ (e.g. an 8-bit or 32-bit processor)
  - corresponds to the data size that can be manipulated at a time by the processor
  - typically reflected in the size of the processor (internal) data path and register bank
- An 8-bit processor can only manipulate one byte of data at a time, while a 32-bit processor can handle one 32-bit double word sized data at a time
  - even though the data content may only be of single byte size

# Registers

- The most fundamental storage area in the processor
  - is closely located to the processor
  - provides very fast access, operating at the same frequency as the processor clock
  - but is of limited quantity (typically less than 100)
- Most are of the general purpose type and can store any type of information:
  - data – e.g., timer value, constants
  - address – e.g., ASCII table, stack
- Some are reserved for specific purposes
  - program counter (r15 in ARM)
  - program status register (CPSR in ARM)

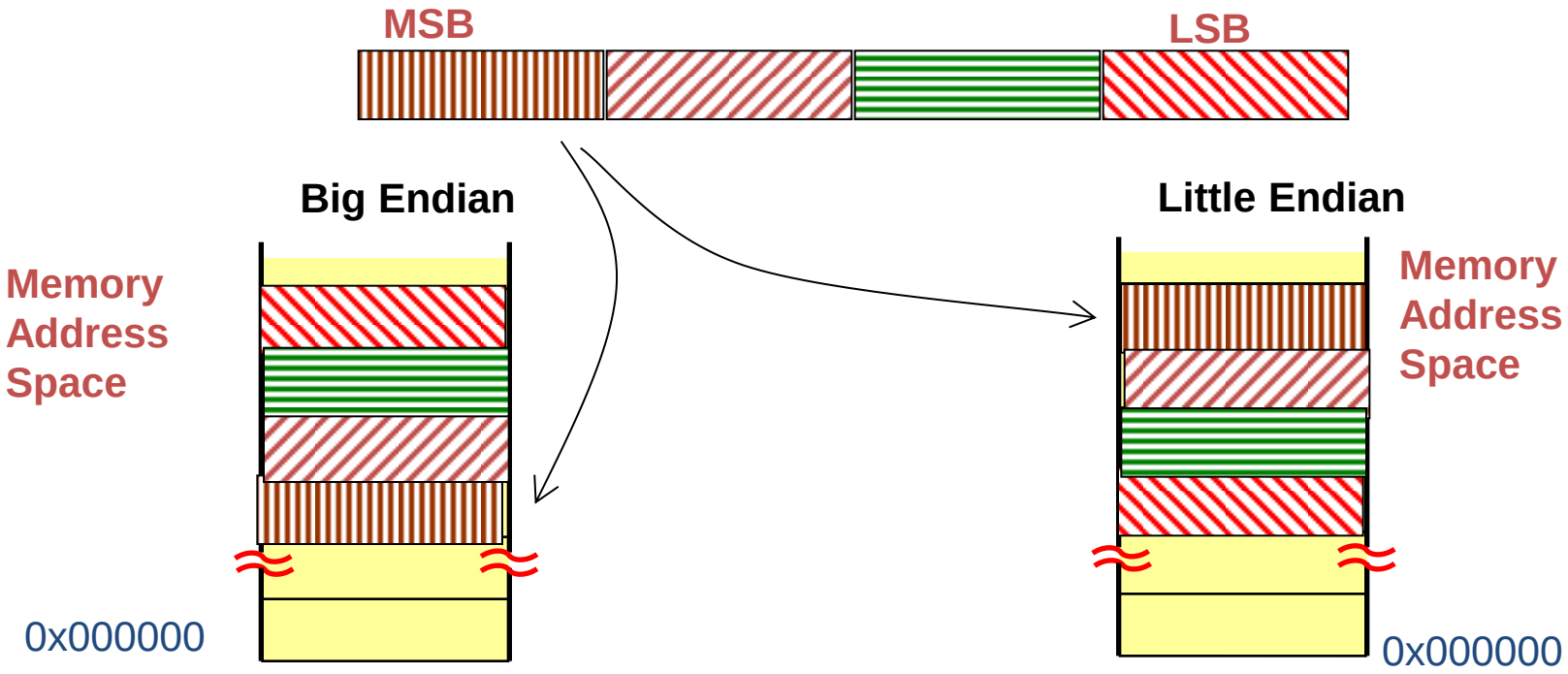
# Data Organization in Memory

- Typically, memory contains a storage locations that can store data of a certain fixed size
  - most commonly of the 8-bit (byte) size
- Each location is provided with a unique address.
- Depending on the data path size of the processor
  - the memory content is accessible in the size of an 8-bit byte, a 16-bit half word, a 32-bit word, and even a 64-bit double word

# Data Alignment

- A 32-bit data consists of four bytes of data, which are stored in four successive memory locations
- Data and code must be aligned to the respective address size boundary.
  - e.g., for a 32-bit system which aligns to the word boundary, the lowest two address bits equal to zero
- But what is the order of the four bytes of data?
  - depends on the Endianness adopted
- In the Little Endian format,
  - the least significant byte (LSB) is stored in the lowest address of the memory, with the most significant byte (MSB) stored in the highest address location of the memory.
- In the Big Endian format,
  - the least significant byte (LSB) is stored in the highest address of the memory, with the most significant byte (MSB) stored in the lowest address location of the memory.

# Data Endianness



# Comparison

- Little Endian
  - The order matched with processor instructions typically process numbers from LSB to MSB
  - The byte number corresponds with the address offset, suitable for multi-precision data manipulation
- Big Endian
  - Can compare numerical data by just accessing the zero offset byte
  - Corresponds to the written order of number (starting with the most significant digit)
- Some processors (e.g. ARM) have bi-endian hardware that feature ‘switchable’ endianness

# CISC

- Features of the Complex Instruction Set Computing (CISC):
  - many instructions
  - complex instructions
    - each instruction can execute several low level operations
  - complex addressing modes
    - smaller number of registers needed
- A semantically rich instruction set is accommodated by allowing instructions of variable length

# Advantages of CISC

- As each instruction can execute several low level operations,
  - the code size is reduced to save on memory requirements
  - less main memory access is required and hence processing time is reduced (faster)
- Backward code compatibility is maintained
  - can add new (and more powerful) instructions while retaining the ‘old’ instruction set for code compatibility (i.e. legacy programs can still run)
- Easy to program
  - direct support of high-level language constructs
  - complex instructions that fit well with high-level language expressions

# Limitations of CISC

- A highly encoded instruction set needs to be decoded by hardwired microcode electronic circuitry
  - more complex hardware design
  - slower instruction decoding/execution
- Variable length instructions
  - different execution time among instructions
  - affects pipelined operations

# RISC

RISC – Reduced Instruction Set Computing

- Small instruction sets
- Simpler instructions
- Fixed length instructions
- Large number of registers
- Simpler addressing mode with the Load/Store instructions for accessing memory

# Advantages of RISC

- Simpler instructions
  - one clock per instruction gives faster execution than on a CISC processor with the same clock speed
- Simpler addressing mode
  - faster decoding
- Fixed length instructions
  - faster decoding and better pipeline performance
- Simpler hardware
  - less silicon area
  - less power consumption

# RISC Memory Footprint

- The RISC processor typically needs more memory than a CISC processor does to store the same program
  - complex functions performed in a single but slower instruction in a CISC processor may require two, three, or more simpler instructions in a RISC processor
- To reduce memory requirements and cost
  - ARM provides the 16-bit Thumb instruction set as an option for its RISC processor cores

# Limitations of RISC

- Fewer instructions than CISC
  - as compared to CISC, RISC needs more instructions to execute one task
  - code density is less
  - needs more memory
- No complex instructions
  - no hardware support for division or floating-point arithmetic operations
  - needs a more complex compiler and longer compiling time
- But ARM also adds DSP-like instructions to support commonly used signal processing functions

# Instruction Code Format

- Opcode encoding depends on the number of bits used
  - Example: For ARM, all instructions are of 32-bit length, but only 8 bits (bit 20 to 28) are used to encode the instruction. Hence a total of 28 = 256 different instructions are possible
- A typical instruction is encoded with a specific bit pattern that consists of the following:
  - an opcode field specifying the operation to be performed
  - an operand(s) identification (address) field that depends on the modes of addressing;
    - this provides the address of the register/memory location (s) that store the operand(s), or the operand itself.

# Instruction Opcode Types

General categories of instruction operations:

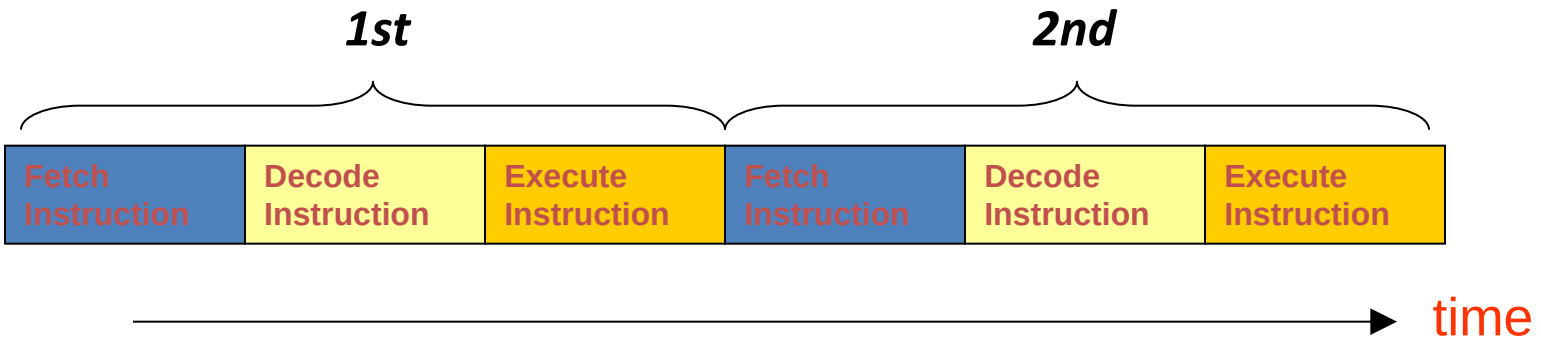
- Data transfer
  e.g., move, load, and store
- Data manipulation
  e.g., add, subtract, logical operation
- Program control
  e.g., branch, subroutine call

# Operand Addressing Types

- Immediate addressing
  - operand is given in the instruction
- Register addressing
  - operand is stored in a register
- Direct addressing
  - operand is stored in memory, with the address given in the instruction
- Indirect (Index) addressing
  - operand is stored in memory, with the address given in a register (address and an offset given in the instruction)
- Implied addressing
  - implicit location like stack and program counter

# Instruction Execution

- Multiple stages are involved in executing an instruction.
  - Example:
  1) Fetching the instruction code
  2) Decoding the instruction code
  3) Executing the instruction code
- Hence multiple processor clock cycles are needed to execute one single instruction.

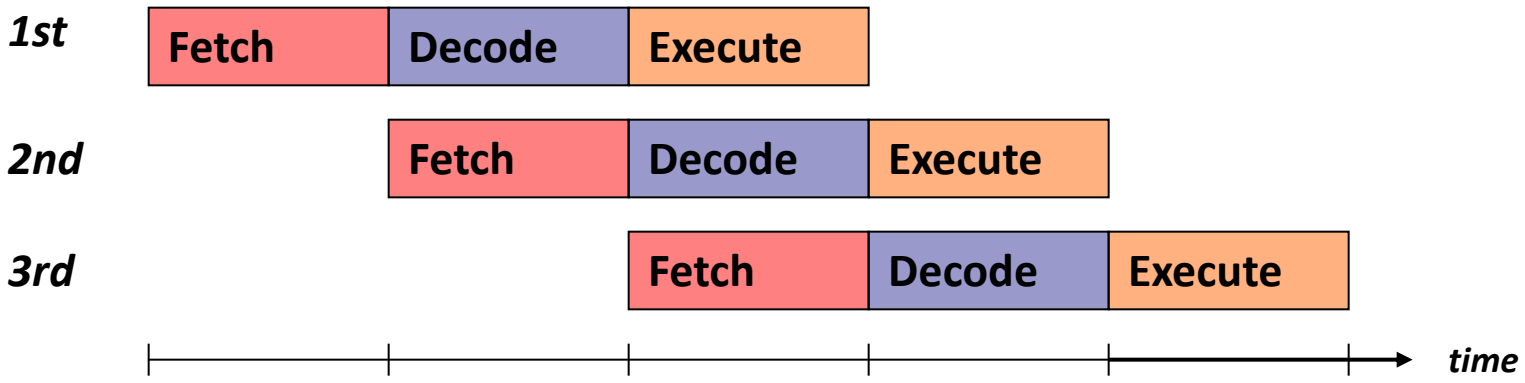


# Instruction Pipeline

- The pipeline allows concurrent execution of multiple different instructions
  - execution of different stages of multiple instructions at the same time
- During a normal operation
  - while one instruction is being executed
  - the next instruction is being decoded
  - and a third instruction is being fetched from memory
  - allows effective throughput to increase to one instruction per clock cycle

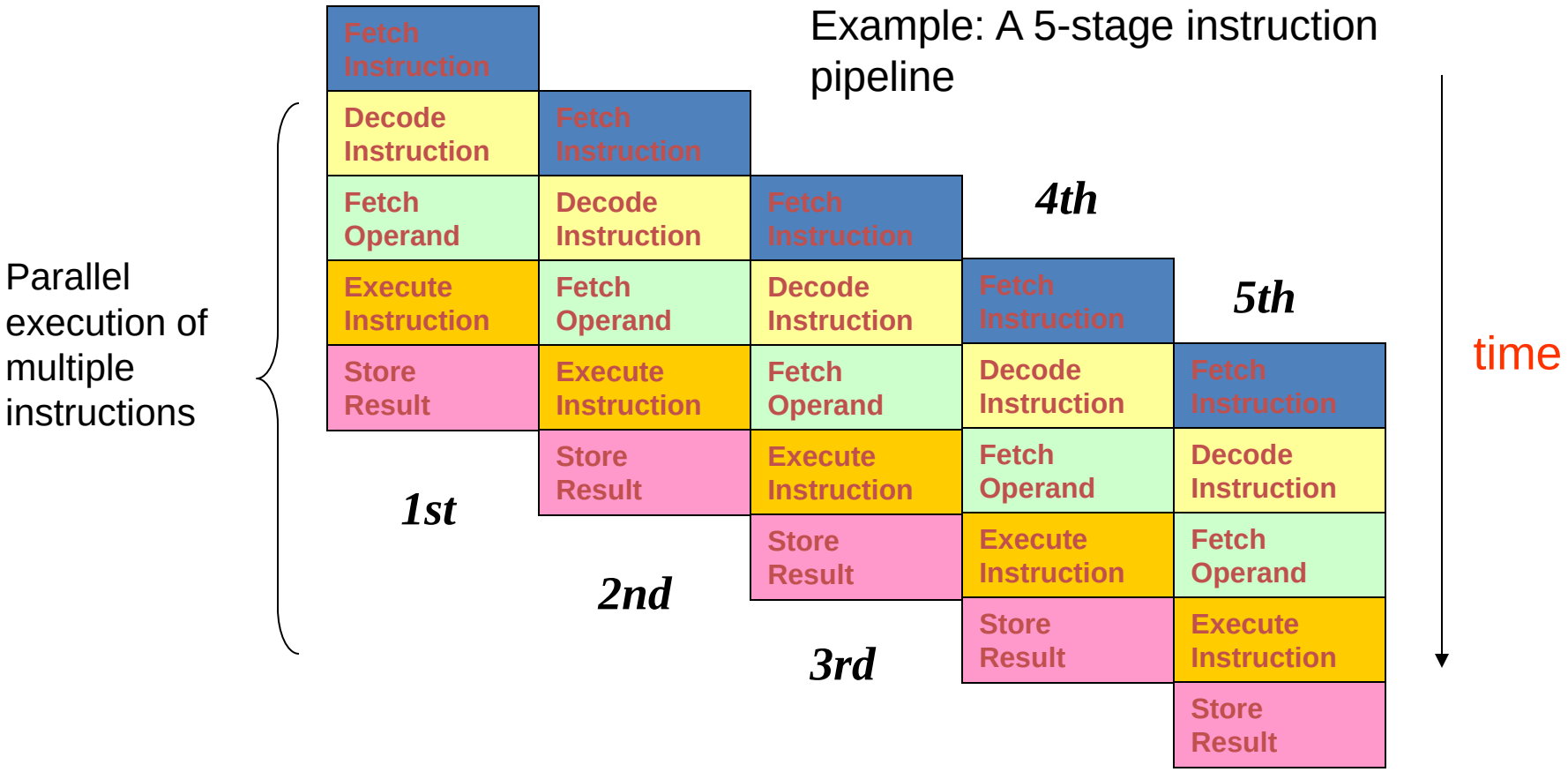
# Cortex-M3 Pipeline

- The Cortex-M3 Uses the 3-stage pipeline for instruction executions
  - Fetch ⇒ Decode ⇒ Execute
  - Pipeline design allows effective throughput to increase to one instruction per clock cycle
  - Allows the next instruction to be fetched while still decoding or executing the previous instructions

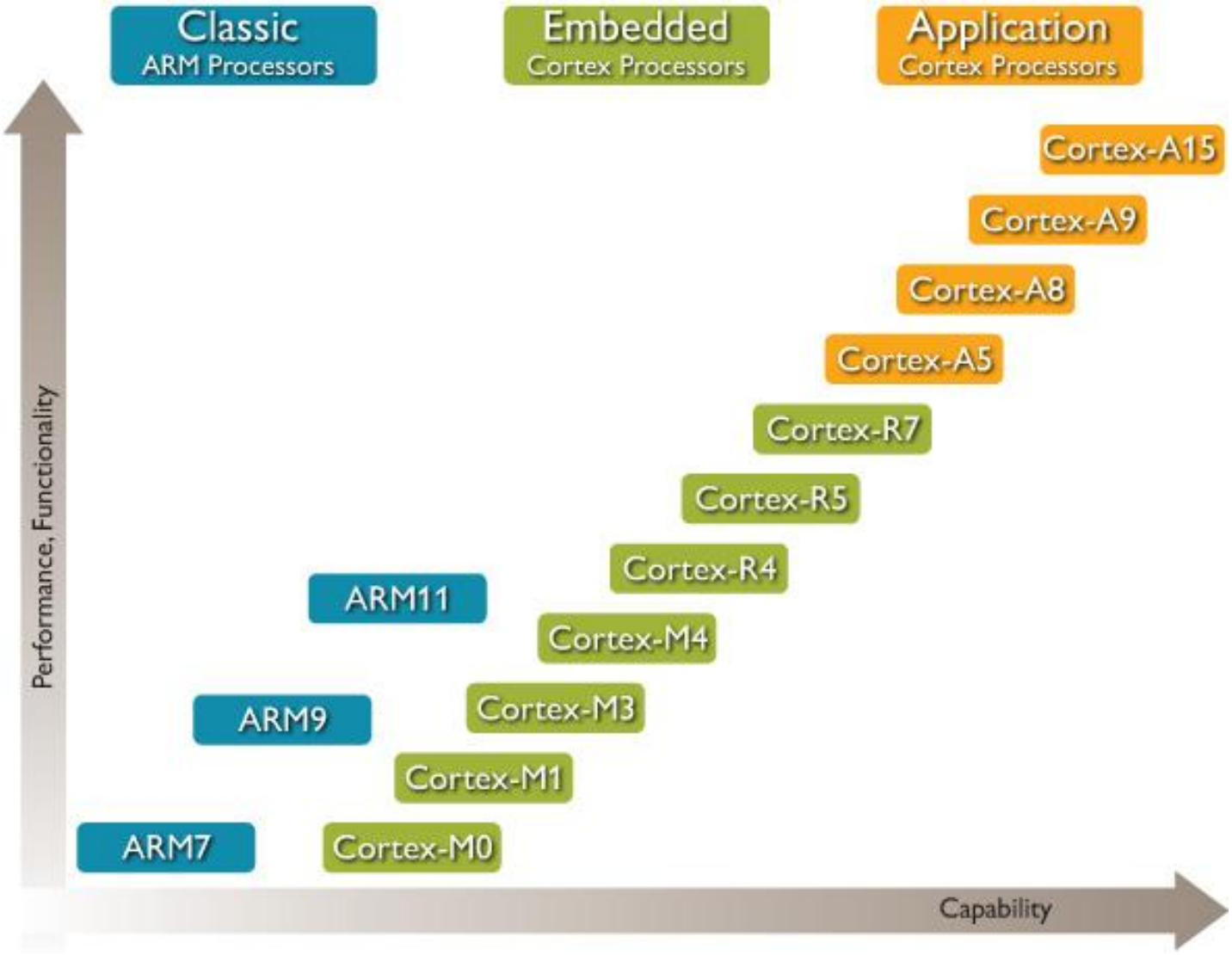


# Pipelined Architecture

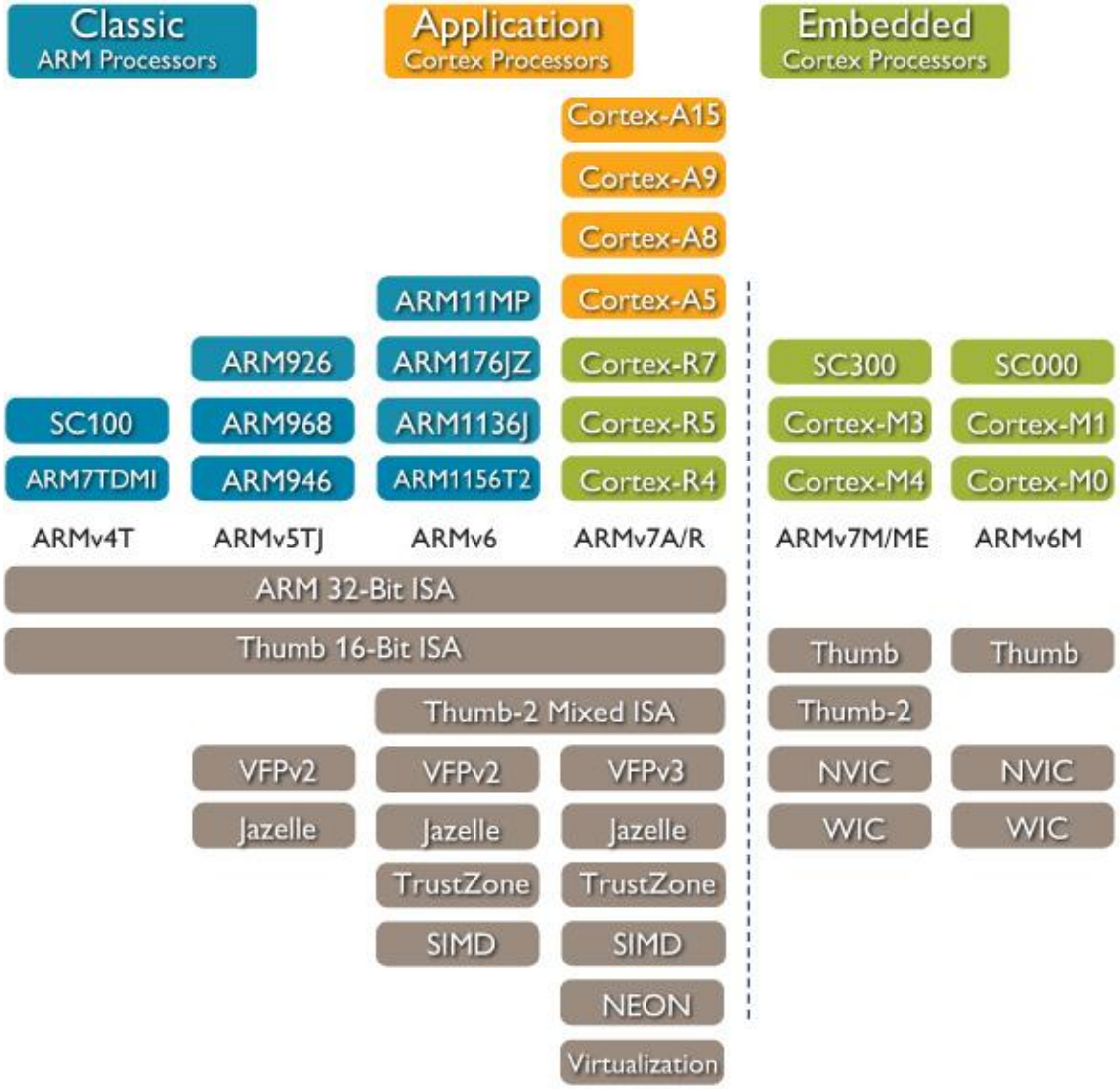
- A longer pipeline can also be used to further break down the operation carried out in the individual stage
  - simpler logic for each stage to increase system clock



# ARM Processors Families



# ARM Processors Architectures



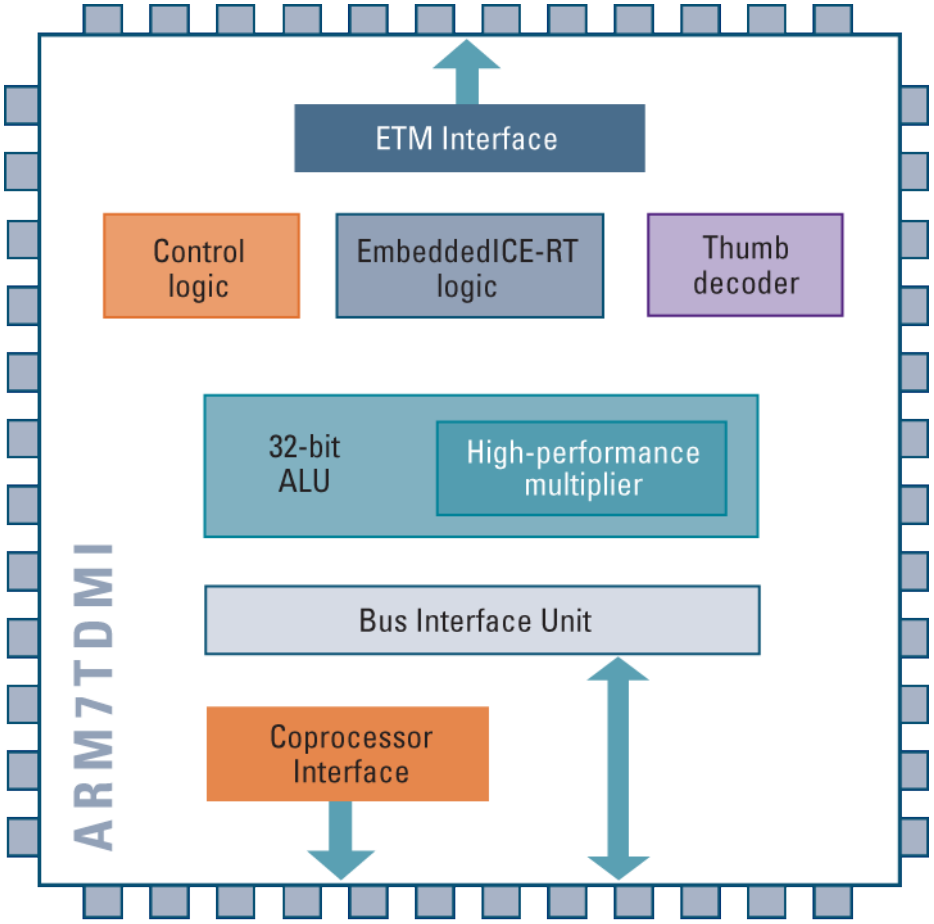
# ARM Processors Architectures (2)

- Key attributes: Implementation size, performance, and very low power.
- Architectures types:
  - **ARMv4T** architecture introduced the 16-bit Thumb® instruction set alongside the 32-bit ARM instruction set.
  - **ARMv5TEJ** architecture introduced arithmetic support for digital signal processing (DSP) algorithms.
  - **ARMv6** architecture introduced an array of new features including the Single Instruction Multiple Data (SIMD) operations.
  - **ARMv7** architecture implementsThumb-2 technology.
    - **Cortex-A** implements a virtual memory system architecture based on an MMU, an optional NEON processing unit for multimedia applications and advanced hardware Floating Point.
    - **Cortex-R** – implements a protected memory system architecture based on an MPU (memory protection unit).
    - **Cortex-M** – Microcontroller profile designed for fast interrupt processing.

# Classic ARM Processors



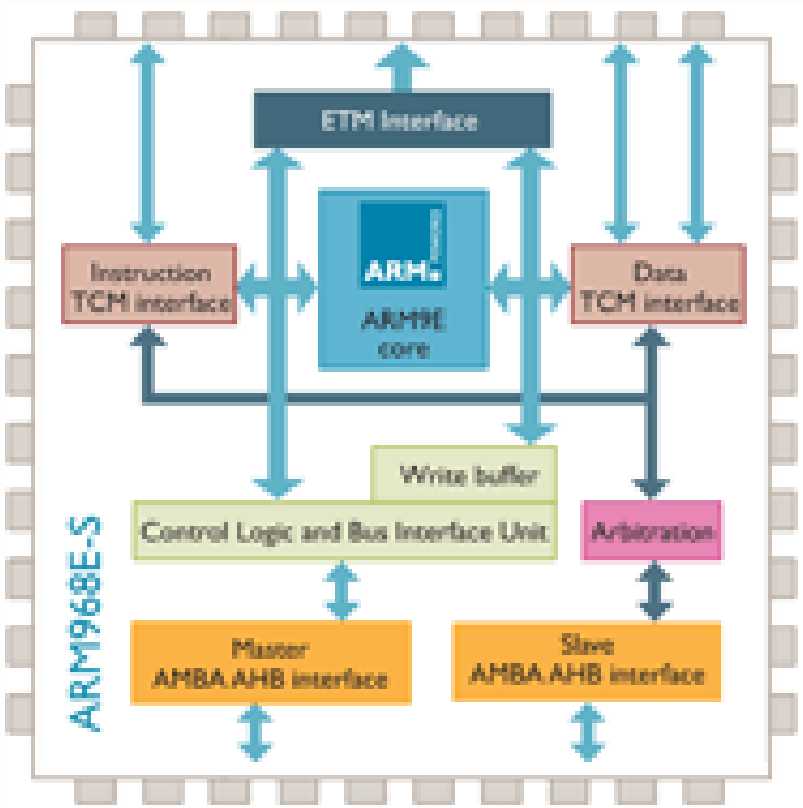
- ARM 7
  - Introduced in 1994.
  - More than 10 billion ARM7 processor family-based devices have powered a wide variety of applications.
  - Today is used for simple 32-bit devices.



# Classic ARM Processors (2)

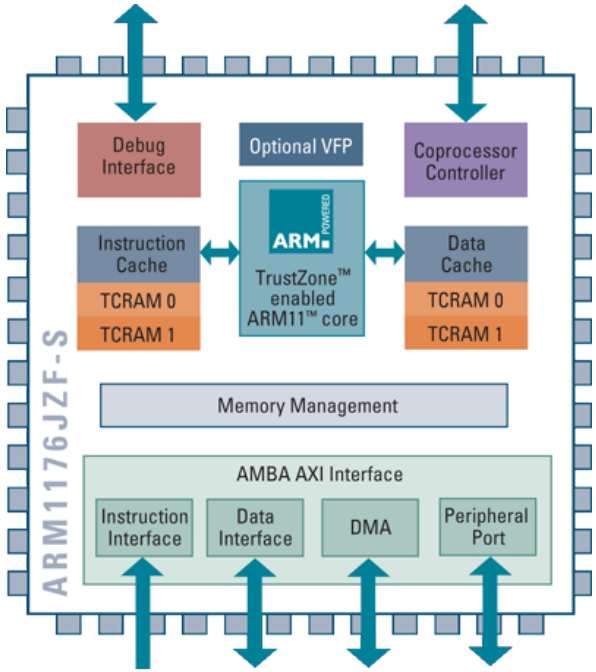


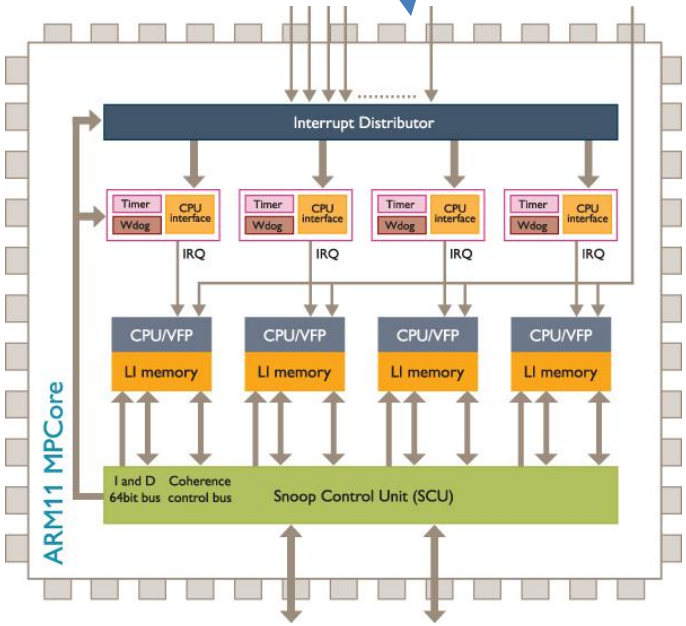
- ARM 9
  - Is the most popular ARM processor family ever.
    - Over 5 Billion ARM9 processors have been shipped so far.
  - Successfully deployed across a wide range of applications.



# Classic ARM Processors (3)

- ARM 11
  - Powers many today's smartphones.
  - Extreme low power.
  - Exists multicore version (ARM11MPCore).

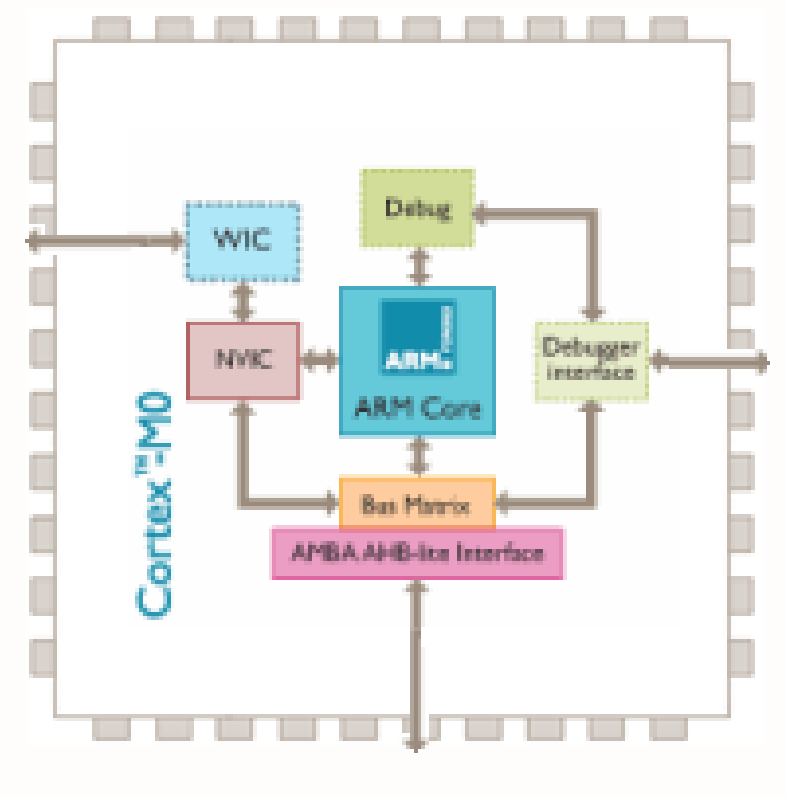


# Embedded ARM Cortex Processors



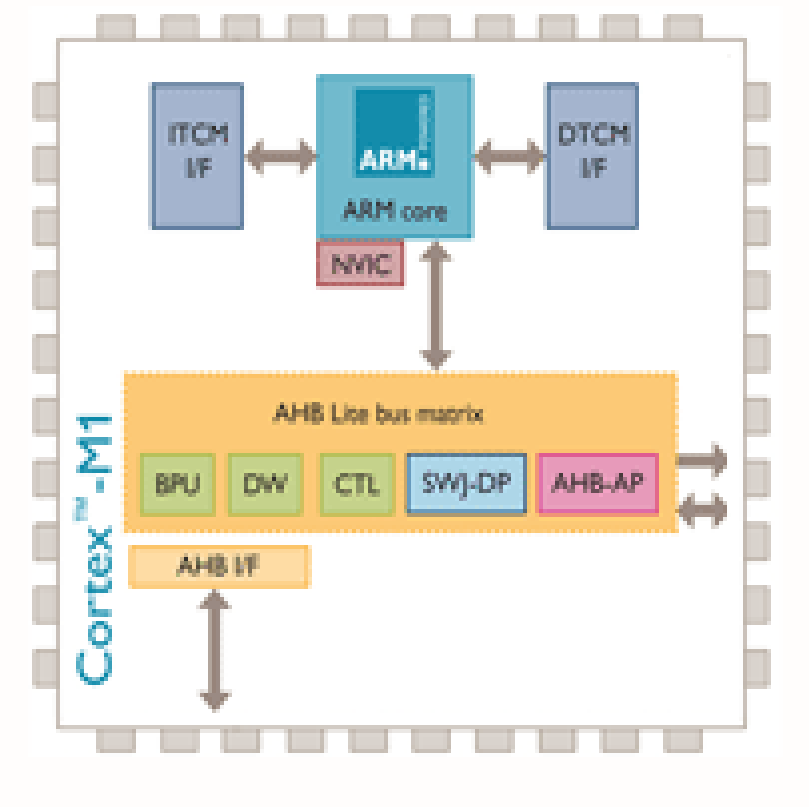
- Cortex M0:
  - Ultra low gate count (less that 12 K gates).
  - Ultra low-power (3 µW/MHz ).
  - 32-bit processor.



# Embedded ARM Cortex Processors (2)



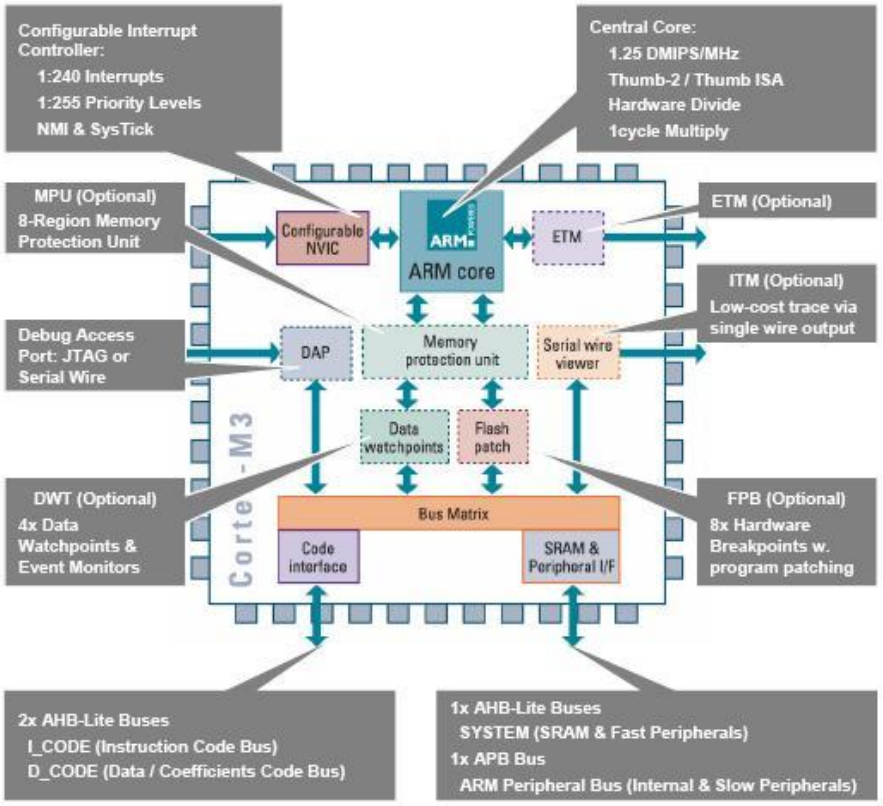
- Cortex M1:
  - The first ARM processor designed specifically for implementation in FPGAs.
  - Supports all major FPGA vendors.
  - Easy migration path from FPGA to ASIC.



# Embedded ARM Cortex Processors (3)



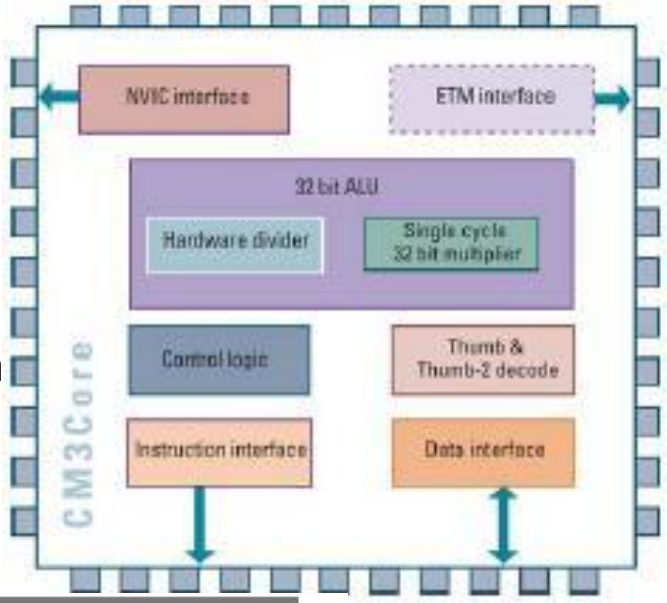
- Cortex M3:
  - The mainstream ARM processor for microcontroller applications.
  - High performance and energy efficiency.

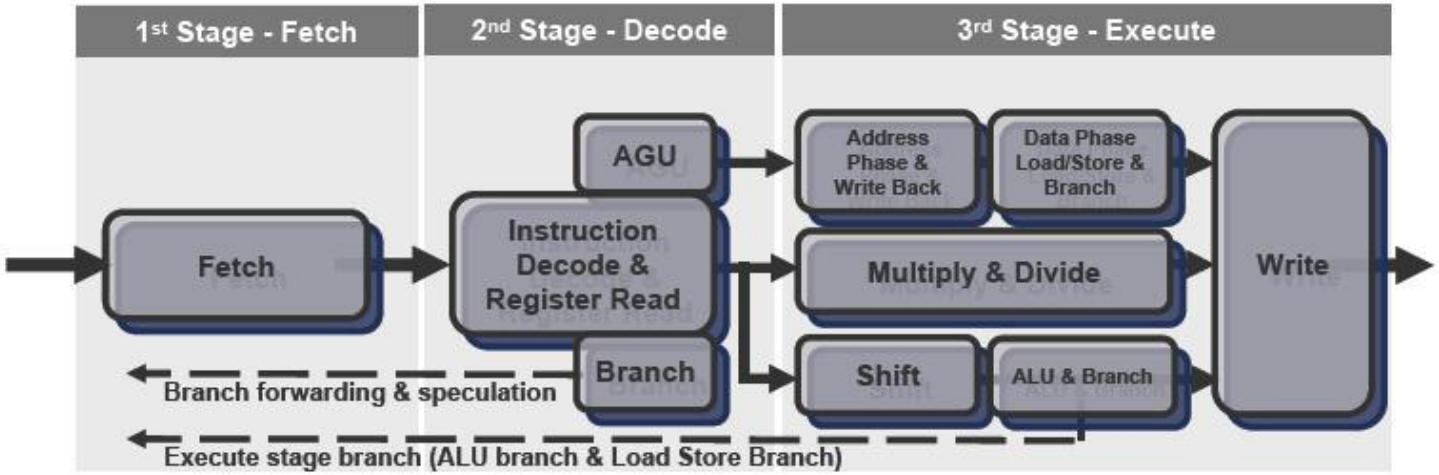


# Cortex M3 Central Core



- Harvard architecture:
  - Separate Instruction & Data buses enable parallel fetch & store.
- Advanced 3-Stage Pipeline:
  - Includes Branch Forwarding & Speculation
- Additional Write-Back via Bus Matrix.

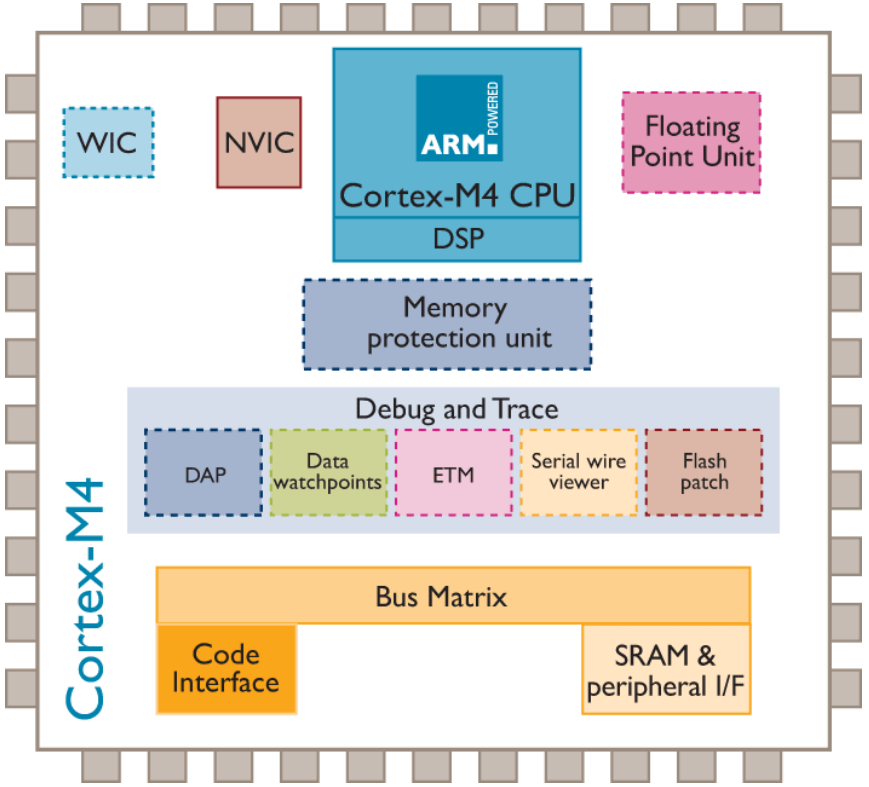


# Embedded ARM Cortex Processors (4)



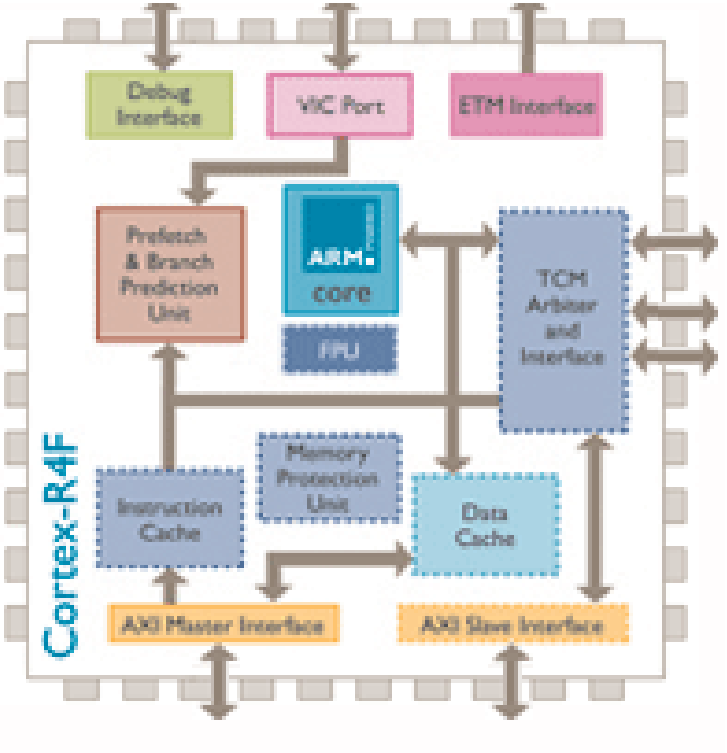
- Cortex M4:
  - The latest embedded processor for DSP.



# Embedded ARM Cortex Processors (5)



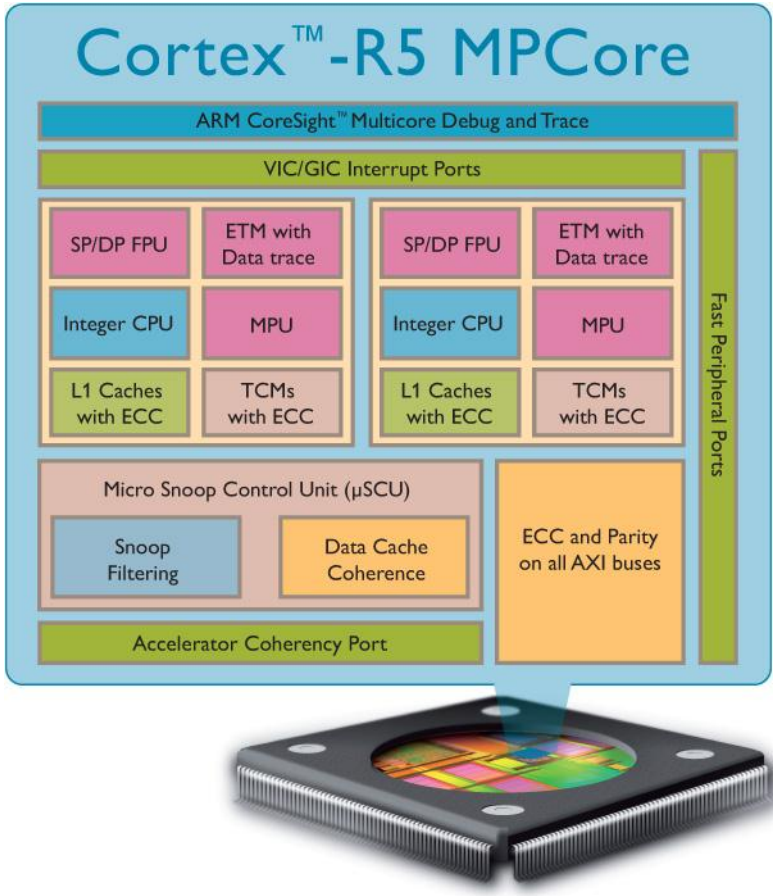
- Cortex R4:
  - First embedded real-time processor based on the ARMv7-R architecture.
  - For high-volume deeply-embedded System-on-Chip applications:
    - Hard disk.
    - Drive controllers.
    - Wireless baseband processors.
    - Electronic control units for automotive systems.



# Embedded ARM Cortex Processors (6)



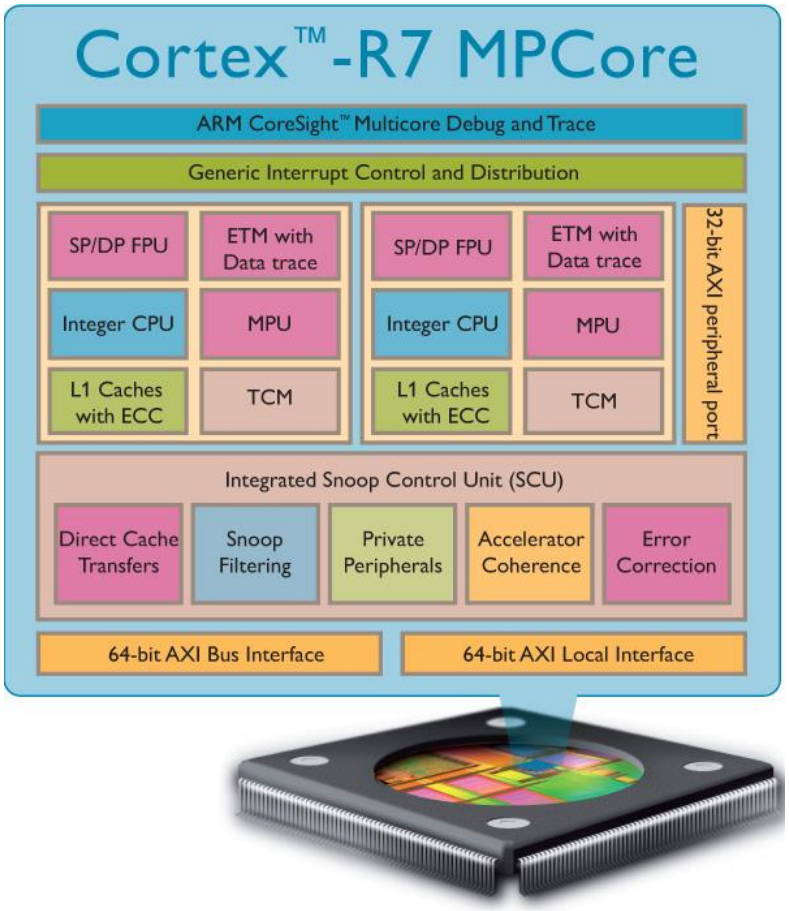
- Cortex R5:
  - Extends the feature set of the Cortex-R4.
  - Enables:
    - Higher levels of system performance.
    - Increased efficiency and reliability.
    - Enhanced error management in dependable real-time systems.



# Embedded ARM Cortex Processors (7)



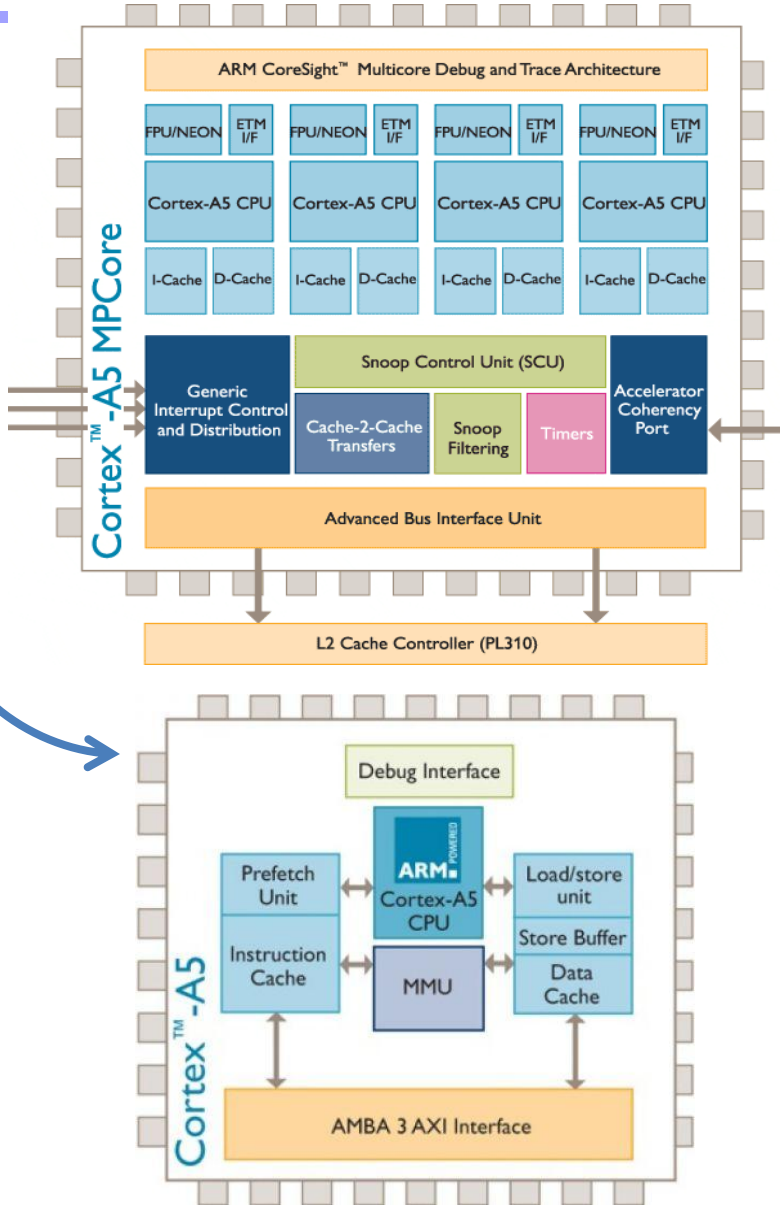
- Cortex R7:
  - High-performance dual-core.
  - It is the highest performing Cortex-R series processor.



# Application ARM Cortex Processors



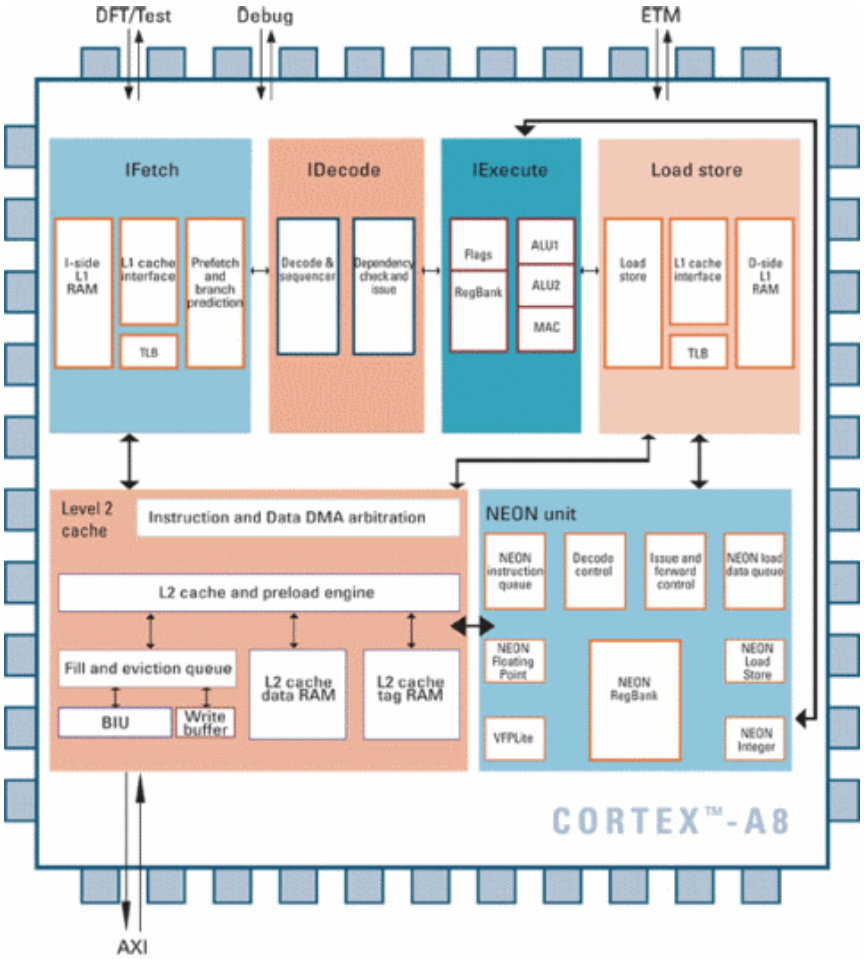
- Cortex A5:
  - Delivers high end features to power and cost sensitive applications.
  - Single core version also available.
  - Suitable for:
    - From entry level smartphones, low cost handsets and smart mobile devices.
    - To pervasive embedded, consumer and industrial devices.



# Application ARM Cortex Processors (2)



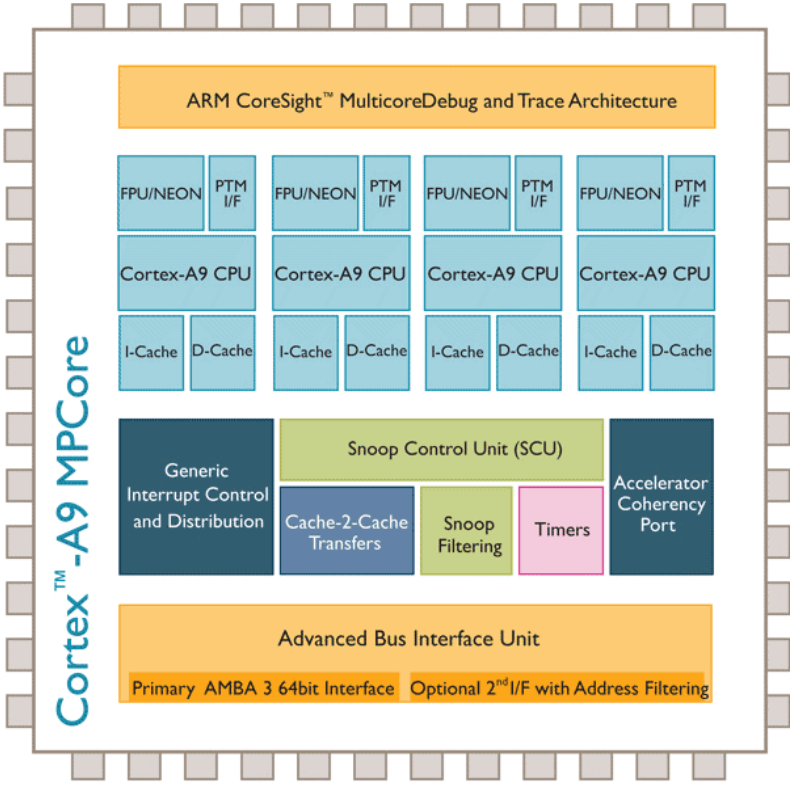
- Cortex A8:
  - Single core only.
  - Can scale in speed from 600MHz to greater than 1GHz.
  - Suitable for high-end feature phones, netbooks, DTVs, printers and automotive-infotainment.



# Application ARM Cortex Processors (3)



- Cortex A9:
  - From 1 to 4 cores.
  - Extremely high levels of performance and power efficiency.
  - Ideal solution for designs requiring high performance in low power or thermally constrained cost-sensitive devices.



# Application ARM Cortex Processors (4)



- Cortex A15:
  - Ultra low-power.
  - Suitable for:
    - Advanced Smartphones.
    - Mobile Computing.
    - High-end Digital Home Entertainment.
    - Wireless Infrastructure.
    - Low-power Servers.