# What is the difference between ARM7 and ARMv7

The term "ARM7" refers to a family of processor cores with a three-stage pipeline and a von Neumann memory interface. These cores are relatively old now but nevertheless still widely used. See http://www.arm.com/products/CPUs/families/ARM7Family.html for more information.

The term "ARMv7" refers to the Instruction Set Architecture (ISA) version 7. Variants of this ISA are to be found in the more recent Cortex family of cores. The ARM7 family or cores are ISA v4T; essentially, cores running later ISAs are backwardly compatible with earlier versions, so code written for an ARM7TDMI (ARM7 family - v4T ISA) will run on a Cortex core, but not necessarily the other way round.

The two are quite different, though they can share code, if the code is written for this.

The name Cortex comes from Core and Texas

Arm7 (1994-2001) uses the Armv4T architecture, which supports two instruction sets: The old Arm instruction set and Thumb

Arm Cortex-M0 uses the Armv6-M (only supports 16-bit thumb instructions).

Arm Cortex-M3 and later uses the Armv7-M which supports the Thumb2 instruction set (16-bit + 32-bit instructions).

Apart from the instructions, there are other differences in the architecture.

For instance, the interrupt handling is different. On Cortex-M, you can write an interrupt routine directly in C like any other subroutine, without adding any special attribute keywords.

On Arm7, your compiler need to add a special prologue/epilogue.

The prologue/epilogue mainly saves and restores registers on the stack.

There are many other differences, which I cannot answer in detail; for this, you'll need an answer from an expert.

Personally, I find both architectures great.

There is no FIQ interrupt on Cortex-M, on the other hand, Interrupts are very easy on this one.

Thumb instructions are slightly limited from the older Arm instructions.

You cannot use LSR in a load or store instruction. This means you can't have an index in the top N bits of a register and use it directly. In such cases, you'll need an extra instruction to extract those bits.

You cannot subtract an index register from the base register in a load or store instruction.

There are some other limits as well.

-But most of the functionality exist in the Thumb2 instruction set, so normally you won't miss any features.

# What does "TDMI-S" stand for?

T : supports both ARM (32-bit) and Thumb (16-bit) instruction sets

An instruction set is a list of binary patterns, or "opcodes", which represent the different logical operations a processor can perform. Software programs can be written at different levels of abstraction, from low level "assembly code" where each written instruction typically maps onto one corresponding opcode, up to high level languages where the written program (source code) needs to be processed by a compiler which typically converts each written instruction into a whole sequence of opcodes.

ARM processors support one or more instruction sets.

The original ARM instruction set consists of 32-bit opcodes, so the binary pattern for each possible operation is four bytes long.

To improve code density, a new, smaller instruction set called "Thumb" was developed, implementing the more commonly-used parts of the ARM instruction set but encoding these in a 16-bit or 2-byte pattern (or occasionally, a pair of such opcodes).

ARM7TDMI-S supports both ARM and Thumb instruction sets, with a defined mechanism for switching between instruction sets at natural program boundaries. This instruction set architecture is called ARMv4T.

Newer ARM processors support enhanced and extended versions of one or both of these instruction sets in ARMv5, ARMv6 and ARMv7 architectures, or may support a new 32-bit instruction set for a 64-bit datapath architecture in ARMv8, either instead of or along-side ARM- and Thumb-compatible instruction sets.

D : Contains Debug extensions

The debug extensions provide the mechanism by which normal operation of the processor can be suspended for debug, including the input signal ports to trigger this behavior; for example a signal to allow a breakpoint to be indicated and a signal to allow an external debug request to be indicated.

M : Enhanced (relative to earlier ARM cores) 32x8 Multiplier block

Earlier ARM processors (prior to ARM7TDMI) used a smaller, simpler multiplier block which required more clock cycles to complete a multiplication. Introduction of this more complex 32x8 multiplier reduced the number of cycles required for a multiplication of two registers (32-bit \* 32-bit) to a few cycles (data dependent). Modern ARM processors are generally capable of calculating at least a 32-bit product in a single cycle, although some of the smallest Cortex-M processors provide an implementation choice of a faster (single-cycle) or a smaller (32 cycle) 32-bit multiplier block.

I : EmbeddedICE macrocell

The EmbeddedICE macrocell consists of on-chip logic to support debug operations. In ARM7TDMI-S, this includes two instruction breakpoint and data watchpoint comparators, an Abort status register, and a debug communications channel to pass data between the target and the host. The EmbeddedICE interacts with the debug extensions, for example to signal a halt to the processor when a breakpoint is met.

-S : synthesizable (ie. distributed as RTL rather than a hardened layout)

ARM7TDMI (without the "-S" extension) was initially designed as a hard macro, meaning that the physical design at the transistor layout level was done by ARM, and licensees took this fixed physical block and placed it into their chip designs. This was the prevalent design methodology at the time. Subsequently, demand increased for a more flexible and configurable solution, so ARM moved towards delivering processor designs as a behavioral description at the "register transfer level" (RTL) written in a hardware description language (HDL), typically Verilog HDL. The process of converting this behavioral description into a physical network of logic gates is called "synthesis", and several major EDA companies sell automated synthesis tools for this purpose. A processor design distributed to licensees as an RTL description (such as ARM7TDMI-S) is therefore described as "synthesizable".

# ARM Cortex-A53

The **ARM Cortex-A53** is one of the first two [microarchitectures](https://en.wikipedia.org/wiki/Microarchitecture) implementing the [ARMv8-A](https://en.wikipedia.org/wiki/ARMv8-A) 64-bit [instruction set](https://en.wikipedia.org/wiki/Instruction_set) designed by [ARM Holdings](https://en.wikipedia.org/wiki/ARM_Holdings)' [Cambridge](https://en.wikipedia.org/wiki/Cambridge) design centre. The Cortex-A53 is a 2-wide decode [superscalar processor](https://en.wikipedia.org/wiki/Superscalar_processor), capable of dual-issuing some instructions.

## Overview

* 8-stage pipelined processor with 2-way [superscalar](https://en.wikipedia.org/wiki/Superscalar), in-order execution pipeline
* DSP and [NEON](https://en.wikipedia.org/wiki/ARM_architecture" \l "Advanced_SIMD_(NEON)) [SIMD](https://en.wikipedia.org/wiki/SIMD) extensions are mandatory per core
* [VFPv4](https://en.wikipedia.org/wiki/VFP_(instruction_set)) [Floating Point Unit](https://en.wikipedia.org/wiki/Floating-point_unit) onboard (per core)
* [Hardware virtualization](https://en.wikipedia.org/wiki/Hardware_virtualization) support
* [TrustZone](https://en.wikipedia.org/wiki/ARM_architecture" \l "Security_extensions) security extensions
* 64-byte [cache lines](https://en.wikipedia.org/wiki/Cache_line)
* 10-entry L1 [TLB](https://en.wikipedia.org/wiki/Translation_Lookaside_Buffer), and 512-entry L2 TLB
* 4 KiB conditional [branch predictor](https://en.wikipedia.org/wiki/Branch_predictor), 256-entry indirect branch predictor

The Cortex-A53 is also used in a number of [Qualcomm](https://en.wikipedia.org/wiki/Qualcomm) [Snapdragon](https://en.wikipedia.org/wiki/Qualcomm_Snapdragon) [SoCs](https://en.wikipedia.org/wiki/System_on_a_chip).[[5]](https://en.wikipedia.org/wiki/ARM_Cortex-A53" \l "cite_note-5)[[6]](https://en.wikipedia.org/wiki/ARM_Cortex-A53" \l "cite_note-6)[[7]](https://en.wikipedia.org/wiki/ARM_Cortex-A53" \l "cite_note-Snapdragon_625-7) Semi-custom derivatives of the Cortex-A53 have been used in the [Kryo 250](https://en.wikipedia.org/wiki/Kryo" \l "Kryo_250) and [Kryo 260](https://en.wikipedia.org/wiki/Kryo" \l "Kryo_260) CPUs

The Cortex®-A53 processor is an extremely power efficient ARMv8 processor capable of supporting 32-bit and 64-bit code seamlessly. It makes use of a highly efficient 8-stage in-order pipeline balanced with advanced fetch and data access techniques for performance. It fits in a power and area footprint suitable for entry level smartphones, at the same time, capable of delivering high aggregate performance in scalable enterprise systems via high core density.

It delivers significantly higher performance than the highly successful Cortex-A7, and is capable of deployment as a standalone applications processor or paired with the [Cortex-A57](http://www.arm.com/products/processors/cortex-a50/cortex-a57-processor.php) processor in a [big.LITTLE](http://www.arm.com/products/processors/technologies/biglittleprocessing.php) configuration for optimum performance, scalability and energy efficiency.

**The Cortex-A53 processor implements the Armv8-A architecture.** This includes:

* Support for both AArch32 and AArch64 Execution states.
* Support for all Exception levels, EL0, EL1, EL2, and EL3, in each execution state.
* The A32 instruction set, previously called the Arm instruction set.
* The T32 instruction set, previously called the Thumb instruction set.
* The A64 instruction set.

The Cortex-A53 processor supports the following architecture extensions:

* Optional Advanced SIMD and floating-point Extension for integer and floating-point vector operations.

### Interconnect architecture

The Cortex-A53 bus interface natively supports one of:

* AMBA 4 ACE bus architecture. See the Arm® AMBA® AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite.
* AMBA 5 CHI bus architecture. See the Arm® AMBA® 5 CHI Protocol Specification

### Generic Interrupt Controller architecture

The Cortex-A53 processor implements the Generic Interrupt Controller (GIC) v4 architecture. The Cortex-A53 processor includes only the GIC CPU Interface. See the Arm® Generic Interrupt Controller Architecture Specification.

## Features

The Cortex-A53 processor includes the following features:

* Full implementation of the Armv8-A architecture instruction set with the architecture options listed in [*Arm architecture*](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500j/BABHJJBC.html).
* In-order pipeline with symmetric dual-issue of most instructions.
* Harvard Level 1 (L1) memory system with a Memory Management Unit (MMU).
* Level 2 (L2) memory system providing cluster memory coherency, optionally including an L2 cache

## .4. Interfaces

The Cortex-A53 processor has the following external interfaces:

* Memory interface that implements either an ACE or CHI interface.
* Optional Accelerator Coherency Port (ACP) that implements an AXI slave interface.
* Debug interface that implements an APB slave interface.
* Trace interface that implements an ATB interface.
* CTI.
* Design for Test (DFT).
* Memory Built-In Self-Test (MBIST).
* Q-channel, for power management.

## . Implementation options

[Table 1.1](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500j/BABJGAHE.html" \l "CACHFIED) lists the implementation options at build time for the Cortex-A53 processor.

**Table 1.1. Cortex-A53 processor implementation options**

| Feature | | Range of options |
| --- | --- | --- |
| Number of cores | | Up to four cores. |
| L1 Instruction cache size | | * 8K. * 16K. * 32K. * 64K. |
| L1 Data cache size | | * 8K. * 16K. * 32K. * 64K. |
| L2 cache | | Included or not. |
|  | L2 cache size | * 128K. * 256K. * 512K. * 1024K. * 2048K. |
|  | L2 data RAM input latency | * 1 cycle. * 2 cycles. |
|  | L2 data RAM output latency | * 2 cycles. * 3 cycles. |
|  | SCU-L2 cache protection | Included or not. |
| Advanced SIMD and floating-point Extension | | Included or not. |
| Cryptography Extension | | Included or not. |
| CPU cache protection[[a](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500j/BABJGAHE.html" \l "ftn.id9000237)] | | Included or not. |
| AMBA 5 CHI or AMBA 4 ACE interface | | * AMBA 5 CHI. * AMBA 4 ACE. |
| Accelerator Coherency Port (ACP)[[b](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500j/BABJGAHE.html" \l "ftn.id9000294)] | | Included or not. |
| v7 or v8 Debug memory map | | * v8 Debug memory map. * v7 Debug memory map. |
| [[a](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500j/BABJGAHE.html" \l "id9000237)] Not implemented if the L2 cache is implemented and SCU-L2 cache protection is not implemented.  [[b](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500j/BABJGAHE.html" \l "id9000294)] Not implemented if the Cortex-A53 processor does not include an L2 cache. | | |

### Note

* The L1 duplicate tags in the SCU are protected by the CPU cache protection.
* There is no option to implement floating-point without Advanced SIMD.
* There is no option to implement the Cryptography Extension without the Advanced SIMD and floating-point Extension.
* All cores share a common L2 cache.

### Processor configuration

All cores in a cluster have identical configurations, that were determined during the build configuration. These configurations cannot be changed by software:

* Either all of the cores have L1 cache protection, or none have.
* Either all of the cores have Advanced SIMD and floating-point Extensions, or none have.
* Either all of the cores have Cryptography Extensions, or none have.
* All cores must have the same size L1 caches as each other.