Table of Contents

ARM History

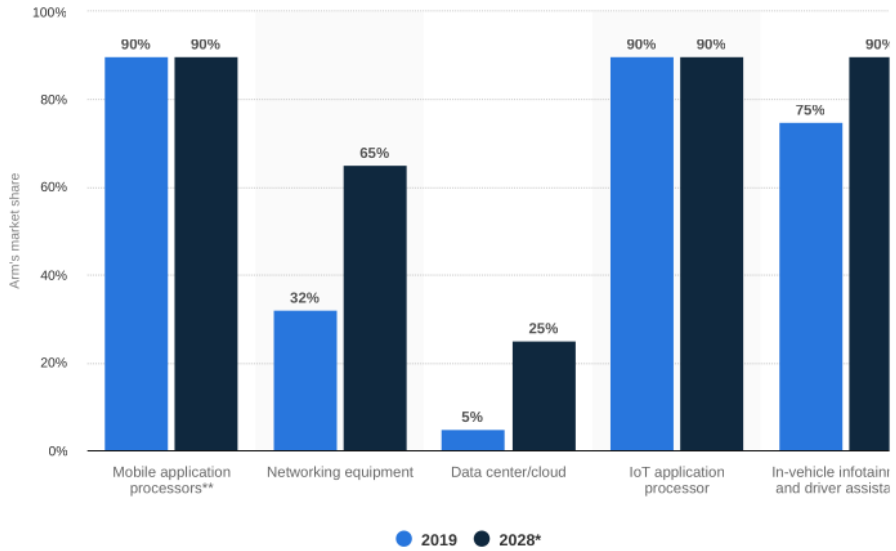
- Advanced RISC Machine
- In 1983, the British company Acorn collaborating with Cambridge University introduced ARM
- In April 1985, the first 32-bit ARM processor came to life. It was a RISC machine built with only 25,000 transistors.
- In 1990, a joint venture Advanced RISC Machine (ARM) between Acorn, Apple, and VLSI Technology was formed
- Texas Instrument (TI), NEC, Sharp and ST Microelectronics, licensed the ARM processor designs, extending the applications of ARM processor into mobile phones, personal digital assistants (PDAs), home entertainment systems, and many other consumer products.
- From 1998 onward rather than manufacturing and selling its own chips, ARM licenses its processor architectures to business partners, including a majority of the world's leading semiconductor companies. Processor and ARM Architecture

ARM History

- Arm generates the majority of its revenue from licensing sales of its technologies, and royalties arising from the subsequent sales of licensee's chips that contain Arm's technologies. Licensees use the design to create microcontrollers (MCUs), central processing units (CPUs), and system-on-chips (SoCs) or application processors (APs).
- An SoC is an integrated circuit that combines all the functions of a computer on one IC microchip.
- These chips typically incorporate the CPU, graphics processing unit (GPU), and memory (RAM), and are commonly found in smartphones, tablets, wearables, and other IoT devices.
- These companies create their processors, microcontrollers, and system-on-chip solutions. using ARM low-cost and power-efficient processor designs.
- This business model is known as IP licensing

ARM Business Target

- 93% of the 1.5 trillion photos captured on Arm-based smartphones and tablets, 7% generated by digital cameras equipped with Arm processors.
- Perhaps less than a decade, perhaps we're already there—we anticipate that 100 percent of the world's digital data will be generated, stored, transmitted and/or analyzed by Arm-based processors at some point during their lifetime.
  `https://newsroom.arm.com/100-percent-digital-data`



ARM Processor and ARM Architecture

- Arm Architecture
  - Describes the details of instruction set, programmer's model, exception model, and memory map
  - Documented in the Architecture Reference Manual
- Arm processor
  - Developed using one of the Arm architectures
  - More implementation details, such as timing information
  - Documented in processor's Technical Reference Manual

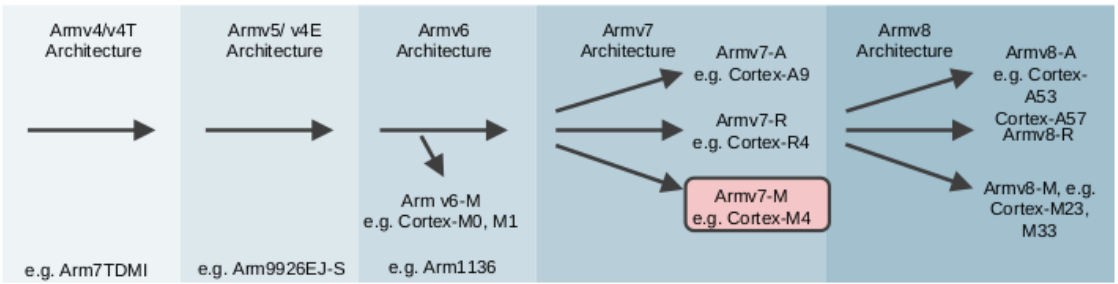


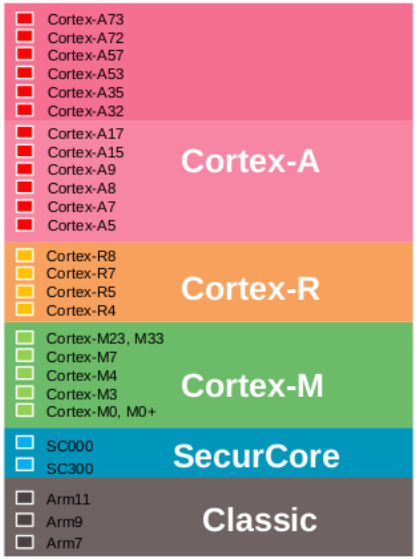
Figure 2

ARM Cortex Processor Family

- Cortex-A series (Application: Cortex-A8)
  - High performance processors capable of full Operating System (OS) support
  - Influenced by multi-tasking OS system requirements
  - Applications include smartphones, digital TV, smart books
  - Memory management support (MMU)
- Cortex-R series (Real-time: Cortex-R4)
  - High performance and reliability for real-time applications;
  - Applications include automotive braking system, powertrains

ARM Cortex Processor Family

- Cortex-M series (Microcontroller: Cortex-M3/M4)
  - Cost-sensitive solutions for deterministic microcontroller applications
  - Applications include microcontrollers, smart sensors
- SecurCoreseries
  - High security applications



ARM Cortex -M Series

- Cortex-M series: Cortex-M0, M0+, M3, M4, M7, M22, M23
  - Low cost, low power, bit and byte operations, fast interrupt response
- Energy efficient
  - Lower energy cost, longer battery life
- Smaller code (Thumb mode instructions)
  - Lower silicon costs
- Embedded applications
  - Smart metering, human interface devices, automotive and industrial control systems, white goods, consumer products and medical instrumentation

ARM Cortex-M Series Family

Processor	ARM Architecture	Core Architecture	Thumb®	Thumb®-2	Hardware Multiply	Hardware Divide	Saturated Math	DSP Extensions	Floating Point
Cortex-M0	ARMv6-M	Von Neumann	Most	Subset	1 or 32 cycle	No	No	No	No
Cortex-M0+	ARMv6-M	Von Neumann	Most	Subset	1 or 32 cycle	No	No	No	No
Cortex-M1	ARMv6-M	Von Neumann	Most	Subset	3 or 33 cycle	No	No	No	No
Cortex-M3	ARMv7-M	Harvard	Entire	Entire	1 cycle	Yes	Yes	No	No
Cortex-M4	ARMv7E-M	Harvard	Entire	Entire	1 cycle	Yes	Yes	Yes	Optional

Figure 3

Cortex M4 vs. M3

- The Cortex-M4 ISA is enhanced efficient DSP features
- Extended single-cycle cycle 16/32-bit multiply-accumulate (MAC)
- Dual 16-bit MAC instructions
- Optimized 8/16-bit SIMD arithmetic

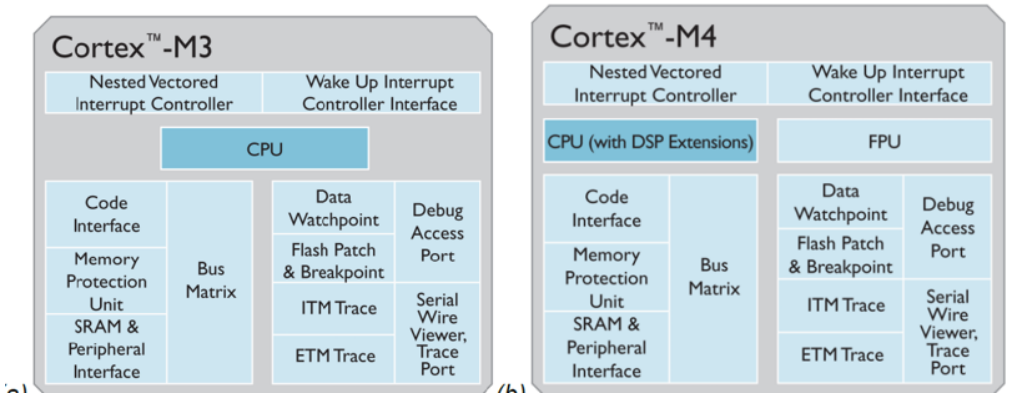


Figure 4

ARMv7 M Thumb Feature

- RISC processor typically needs more memory than a CISC does to store the same program.
- To reduce memory requirements and execute in a single-cycle ARM created the Thumb instruction set as an option for their RISC processor cores.
- Thumb instruction set is the ARM7TDMI. The "T" in the core's full name specifies Thumb
- The Thumb instruction set consists of 16-bit instructions that act as a compact shorthand for a subset of the 32-bit instructions of the standard ARM.
- Every Thumb instruction could instead be executed via the equivalent 32-bit ARM instruction. However, not all ARM instructions are available in the Thumb subset;

ARMv7 M Thumb Feature

- Why have two instruction sets in the same CPU? But really the ARM contains only one instruction set: the 32-bit set. When it's operating in the Thumb state, the processor simply expands the smaller shorthand instructions fetched from memory into their 32-bit equivalents.
- The difference between two equivalent instructions lies in how the instructions are fetched and interpreted prior to execution, not in how they function.
- Since the expansion from 16-bit to 32-bit instruction accomplished via dedicated hardware within the chip, it doesn't slow execution even a bit. But the narrower 16-bit instructions do offer memory advantages.
- The Thumb instruction set provides most of the functionality: Arithmetic and logical operations, load/store data movements, and conditional and unconditional branches are supported

Thumb-2 Feature

- Thumb-2 is an enhancement to the 16-bit Thumb Instruction Set Architecture (ISA).
- It adds 32-bit instructions that can be freely intermixed with 16-bit instructions in a program.
- The additional 32-bit instructions enable Thumb-2 to cover the functionality of the ARM instruction set.
- The 32-bit instructions enable Thumb-2 to deliver the code density of earlier versions of Thumb, together with performance of the existing ARM instruction set, all within a single instruction set.
- The main enhancements are:
  - 32-bit instructions added to the Thumb instruction set to:
  - provide support for exception handling in Thumb state
  - provide access to coprocessors
  - include Digital Signal Processing (DSP) and media instructions

Cortex M4 Diagram

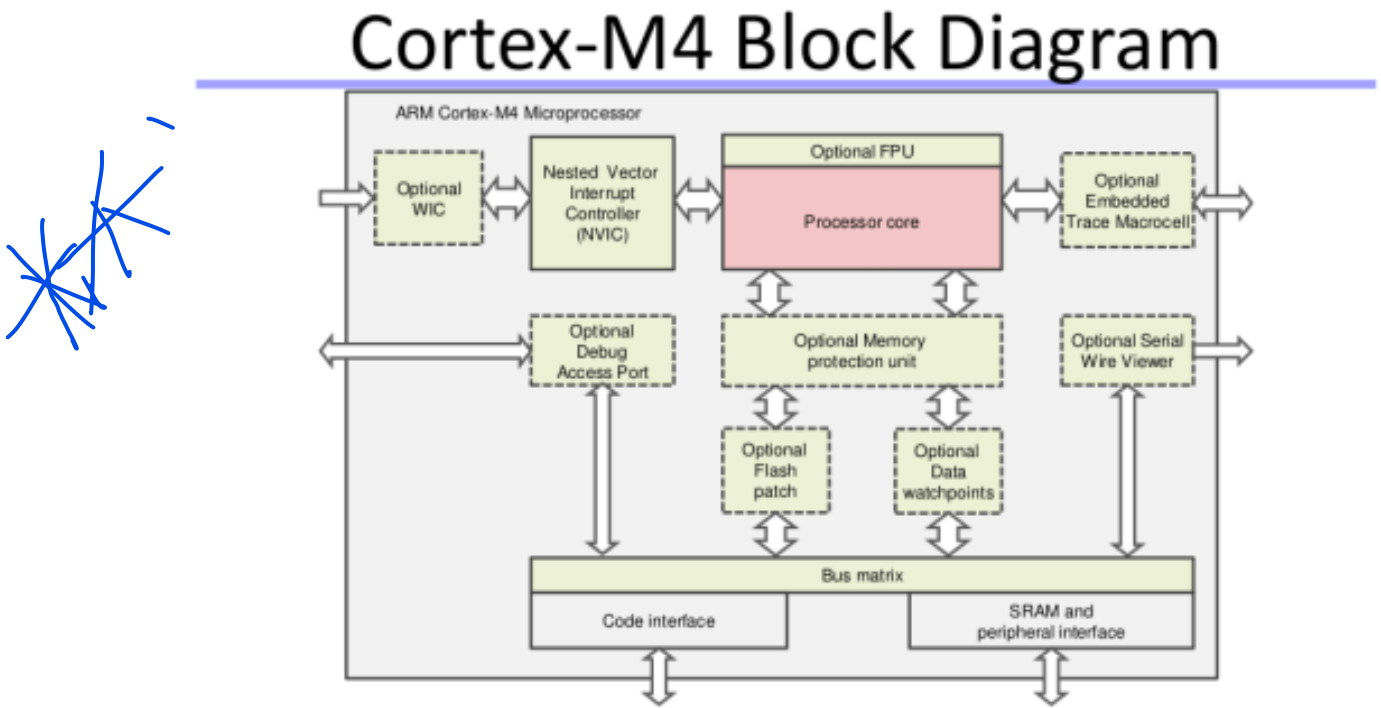


Figure 6

Cortex M4: Overview

- Introduced in 2010: Designed with a large variety of highly efficient signal processing features
- Harvard architecture: Separated data bus and instruction bus
- High Performance Efficiency: 1.25 DMIPS/MHz (DhrystoneMillion Instructions Per Second / MHz) at the order of μWatts / MHz
- Low Power Consumption : Longer battery life –especially critical in mobile products
- Instruction set: Include the entire Thumb®-1 (16-bit) and Thumb®-2 (16/ 32-bit) instruction sets
- 3-stage + branch speculation pipeline
- Supported Interrupts: (i)Non-maskableInterrupt (NMI) (ii) 1 to 240 physical interrupts

Cortex M4: Overview

- 32 bit RISC processor: 32-bit register, 32-bit internal data path, 32-bit bus interface
- handle 16, 32, 64 bit data
- Thumb-2 Technology
- Configurable Nested Vectored Interrupt Controller (NVIC)
- Better debugging support
  - breakpoints support
  - watchpoints
  - support printf() style debugging.
- Multiple high-performance bus interfaces
- Hardware Divide (2-12 Cycles)
- Barrel shifter
- DSP and SIMD extensions
- Single-cycle multiply-accumulate (MAC) unit (Up to 32 x 32 + 64 - 64)
- Optional Memory Protection Unit (MPU)
- Optional Floating Point Unit (FPU) unit

Single-cycle multiply-accumulate unit

- The multiplier unit allows any MUL or MAC instructions to be executed in a single cycle
  - Signed/Unsigned Multiply
  - Signed/Unsigned Multiply-Accumulate
  - Signed/Unsigned Multiply-Accumulate Long (64-bit)

Table 1: Cortex-M4 single cycle MAC

Operation	Instruction	M3	M4
16 x 16 =32	SMULBB, SMULBT, SMULTB, SMULTT	n/a	1
16 x 16+32 =32	SMULABB, SMULABT, SMULATB, SMULATT	n/a	1
16 x 16+64 =64	SMULALBB, SMULALBT, SMULALTB, SMULALTT	n/a	1
32 x 32 =32	MUL	1	1
32 x 32 =64	SMULL,UMULL	5-7	1

Figure 7

Operating States

- Two operating states:
  - ARM :32-bit, word-aligned ARM instructions are executed in this state.
  - Thumb: 16-bit, halfword-aligned Thumb instructions are executed in this state.
- The operating state can be switched between ARM state and Thumb state using the BX (branch and exchange) instructions

Operation Modes

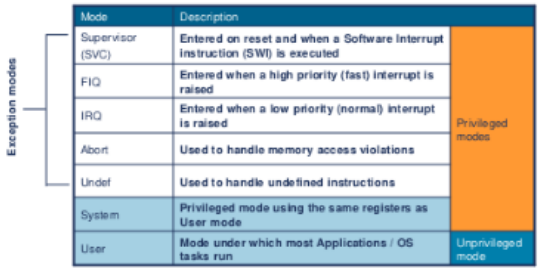
- Operation Modes
  - Thread mode
    - Used to execute application software.
    - The processor enters Thread mode when it comes out of reset.
  - Handler mode
    - Used to handle exceptions.
    - The processor returns to Thread mode when it has finished exception processing.
    - Software execution is always privileged.
- Operation States: The processor can operate in one of two operating states Thumb state. This is normal execution running 16-bit and 32-bit halfword aligned Thumb and Thumb-2 instructions. Debug State. This is the state when in halting debug.

Privileged Level

- Unprivileged
  - Has limited access to the MSR and MRS instructions, and cannot use the CPS instruction
  - Cannot access the system timer, NVIC, or system control block.
  - Might have restricted access to memory or peripherals.
  - Must use the SVC instruction to make a supervisor call to transfer control to privileged software.
- Privileged
  - can use all the instructions and has access to all resources.
  - can write to the CONTROL register to change the privilege level for software execution.

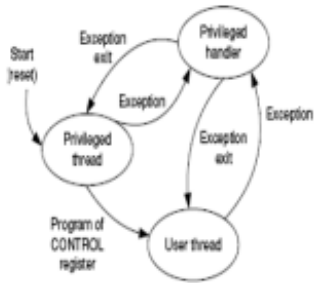
Processor Mode

- The ARM has seven basic operating modes:
  - Each mode has access to own stack and a different subset of registers
  - Some operations can only be carried out in a privileged mode



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

Figure 8



Why Learn Assembler

Given the advance of high-level languages, why do you need to learn assembly language programming? The reasons are:

1. Most industrial microcomputer users program in assembly language.
2. Many microcomputer users will continue to program in assembly language since they need the detailed control that it provides.
3. Many application require the efficiency of assembly language.
4. An understanding of assembly language can help in evaluating high-level languages.
5. Almost all microcomputer programmers ultimately find that they need some knowledge of assembly language, most often to debug programs, write I/O routines, speed up or shorten critical sections of programs written in high-level languages, utilize or modify operating system functions, and undertand other people's programs.

The rest of these notes will deal exclusively with assembler and assembly language programming.