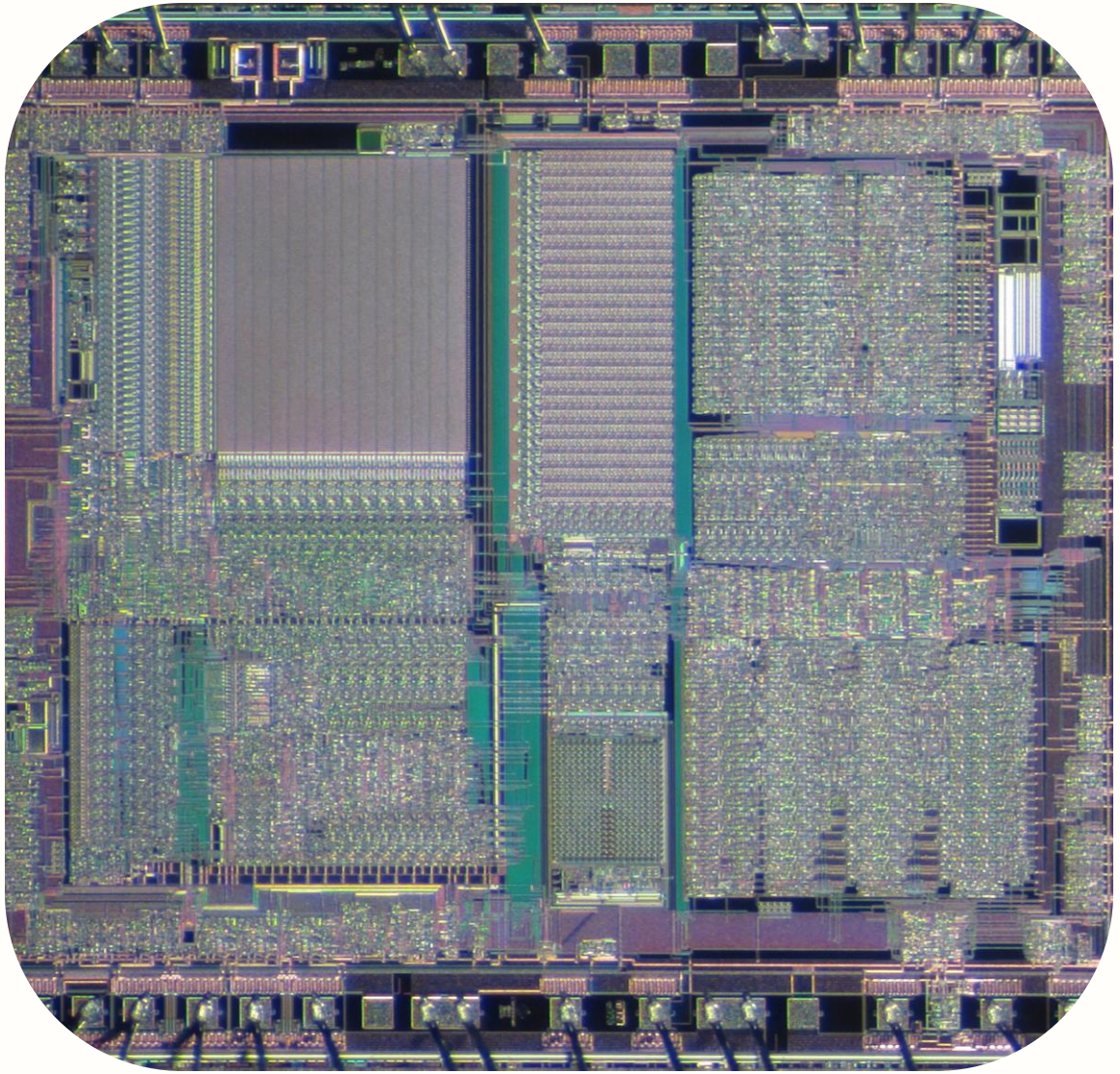


Evolution of Instruction Set Architecture and ARM ISA

Session #3

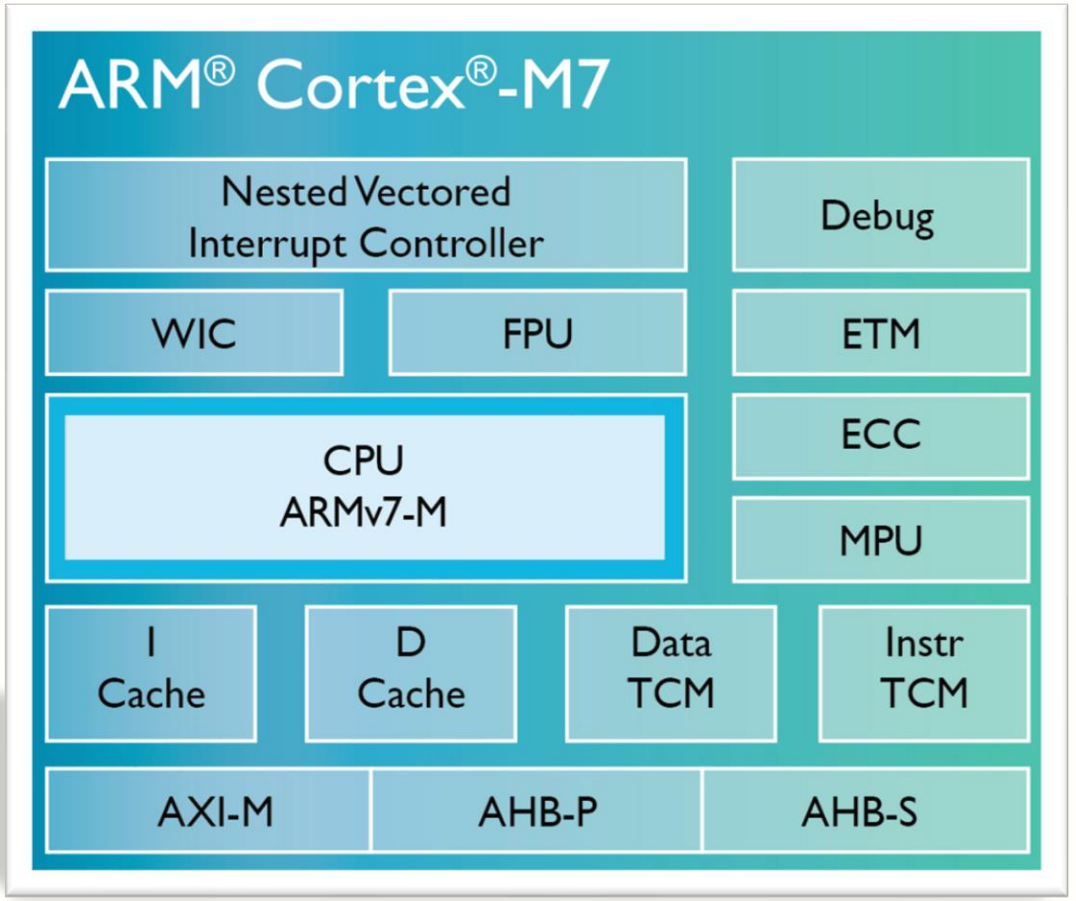
A journey into the evolution of ISA and how it shapes the world of MCUs

What had we seen in Session #1?

- Journeyed through how technology evolved in last 80 years (From Mechanical computing to today's computing using ICs)
- Learnt how it progressed from Semiconductor to Diode, Transistors, LSI Logic ICs, ALUs, Memory, First processor, First MCU

What had we seen in Session #2?

- Journeyed through the evolution of CPU Memory Architecture
- We learnt what are all the major blocks available within an MCU and why they are there and how does that shapes the performance of an MCU



Brief Summary of Session #2

- Memory Architecture: Von Neumann / Harvard / Modified Harvard
- Prefetch / Cache / TCM
- DMA
- DSP and FPU
- NVIC
- Pipelining
- Bus Protocols and Bus Matrix

Session #3: Content List

Instruction Set Architecture (ISA)

1. **What is ISA, Micro Architecture**
2. A glance at the **ISA Evolution:**
    1. CISC→RISC→RISC-V
3. A detailed look into **ISA from ARM**

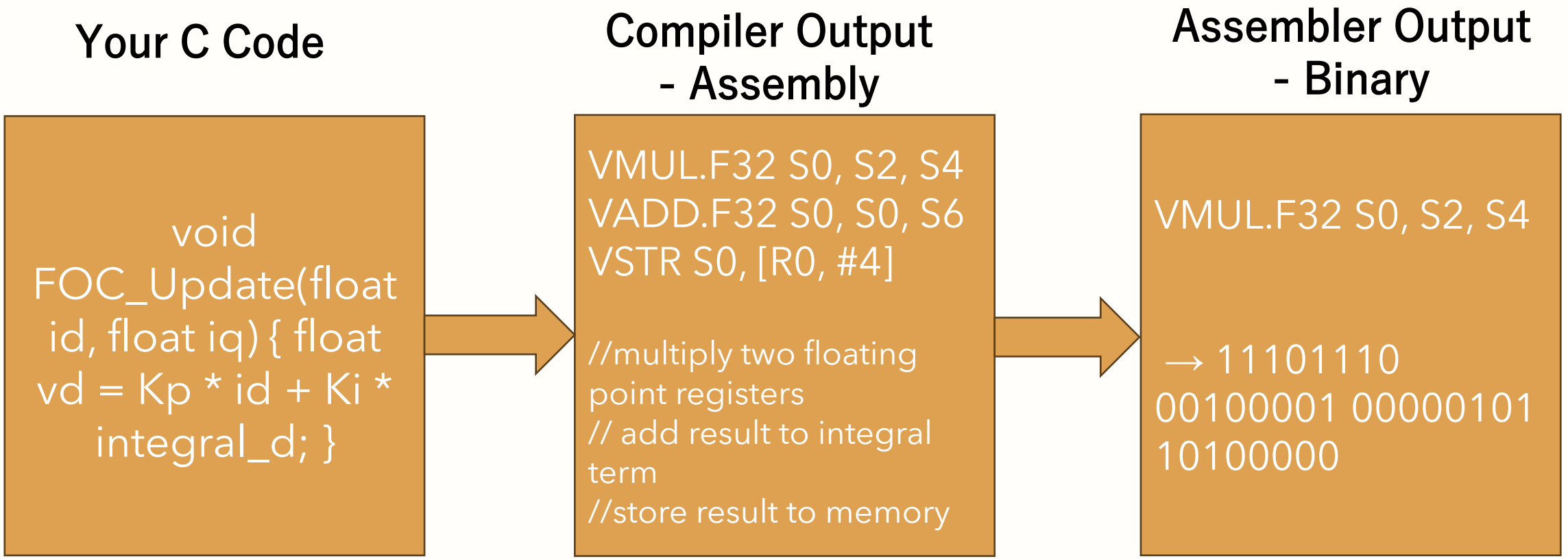
The layers above ISA from ARM

1. What is **ARM Extension?**
2. What is **ARM Accelerator**, Accelerators from ARM, MCU Vendors
3. **CMSIS Libraries** from ARM, Vendor Specific Libraries

Instruction Set Architecture (ISA)

CISC · RISC · RISC-V · ARM · Thumb · Thumb-2 · DSP Extensions

The Journey of your C Code to Silicon



Programmer flashes Bin To Flash Memory → **On power up CPU fetches Binary, decodes, executes**

So, CPU understands only Binary..

Who defines that?

C to Binary – The role of ISA

Level 1 — Your C Code:

Human readable. No CPU understands this directly.

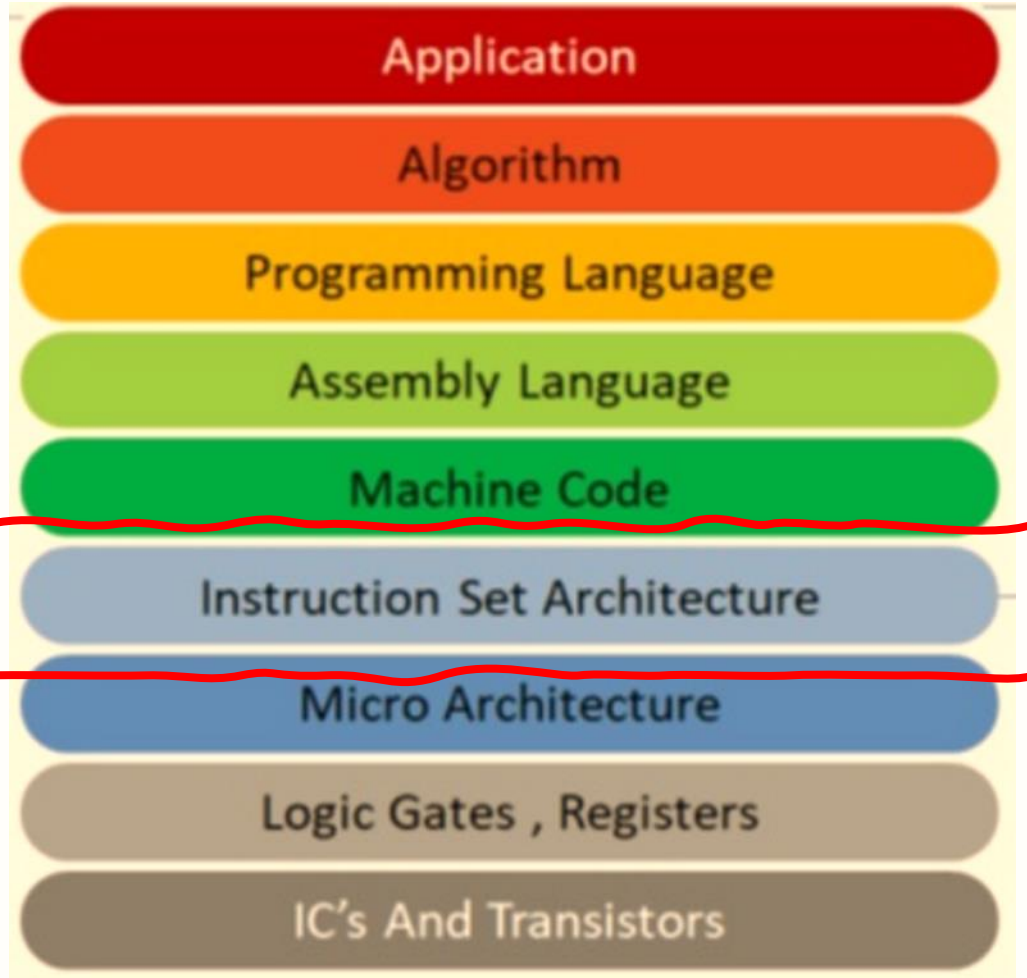
Level 2 — C Compiler Output — Assembly:

The compiler translates C into human-readable assembly instructions **specific to the target ISA.**

Level 3 — Assembler Output — Machine Code (Binary):

The ISA defines every bit of this encoding.

The ISA defines what every possible binary pattern means and what the CPU must do with it.



https://www.geeksforgeeks.org/computer-organization-architecture/microarchitecture-and-instruction-set-architecture/

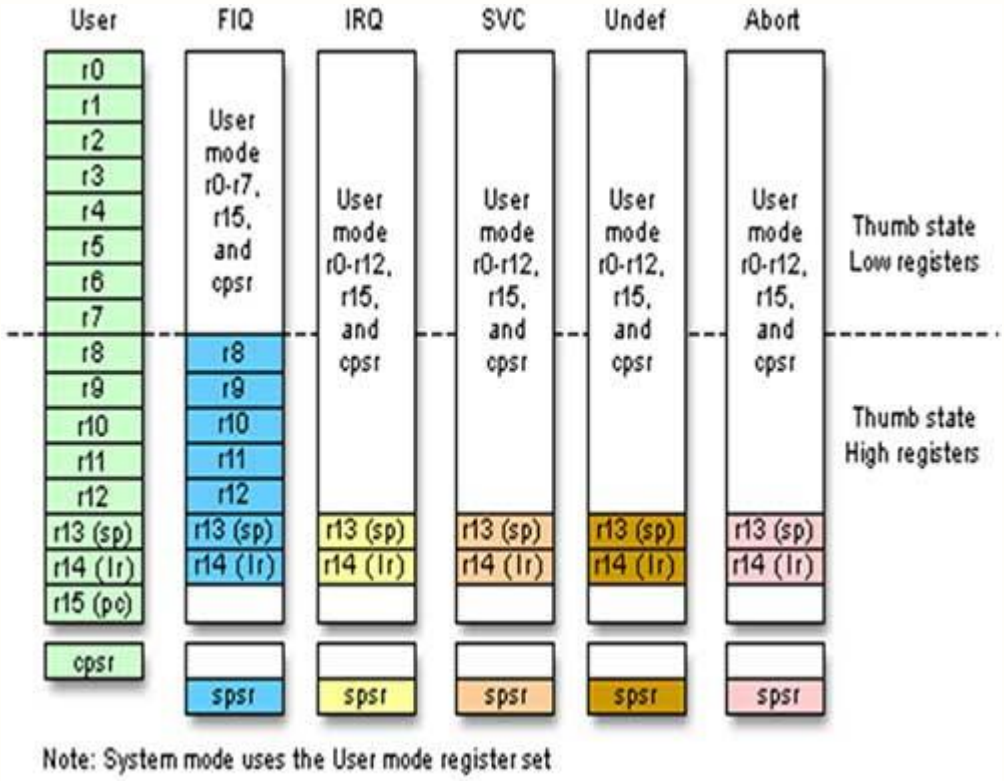
ISA Defines what CPU can do

Instruction set —ADD, SUB, MUL, LDR and so on.

Register file — How many registers, how wide they are, what each is named.

ARM Cortex-M has 16 general-purpose registers R0 through R15 plus special registers.

Data types — Byte, halfword, word, doubleword, single-precision float, double-precision float.

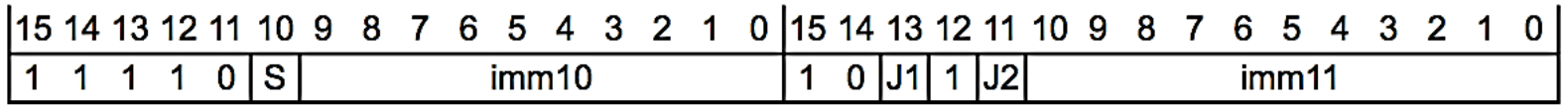


ISA Defines How instructions look

Instruction encoding — Which bits = opcode; registers; immediate

Encoding T4 ARMv6T2, ARMv7

B<c>.W <label> Outside or last in IT block



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	S	imm10										1	0	J1	1	J2	imm11										

```
I1 = NOT(J1 EOR S);  I2 = NOT(J2 EOR S);  imm32 = SignExtend(S:I1:I2:imm10:imm11:'0', 32);
if InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

Memory addressing modes — how the CPU calculates memory addresses. Register indirect, register plus offset, PC-relative, and so on.

ISA Defines How Programs Run

Exception model — how interrupts and faults are handled. Which registers are saved automatically. How the CPU enters and exits interrupt handlers.

Calling convention — how function arguments are passed, how return values are communicated, which registers the caller must preserve.

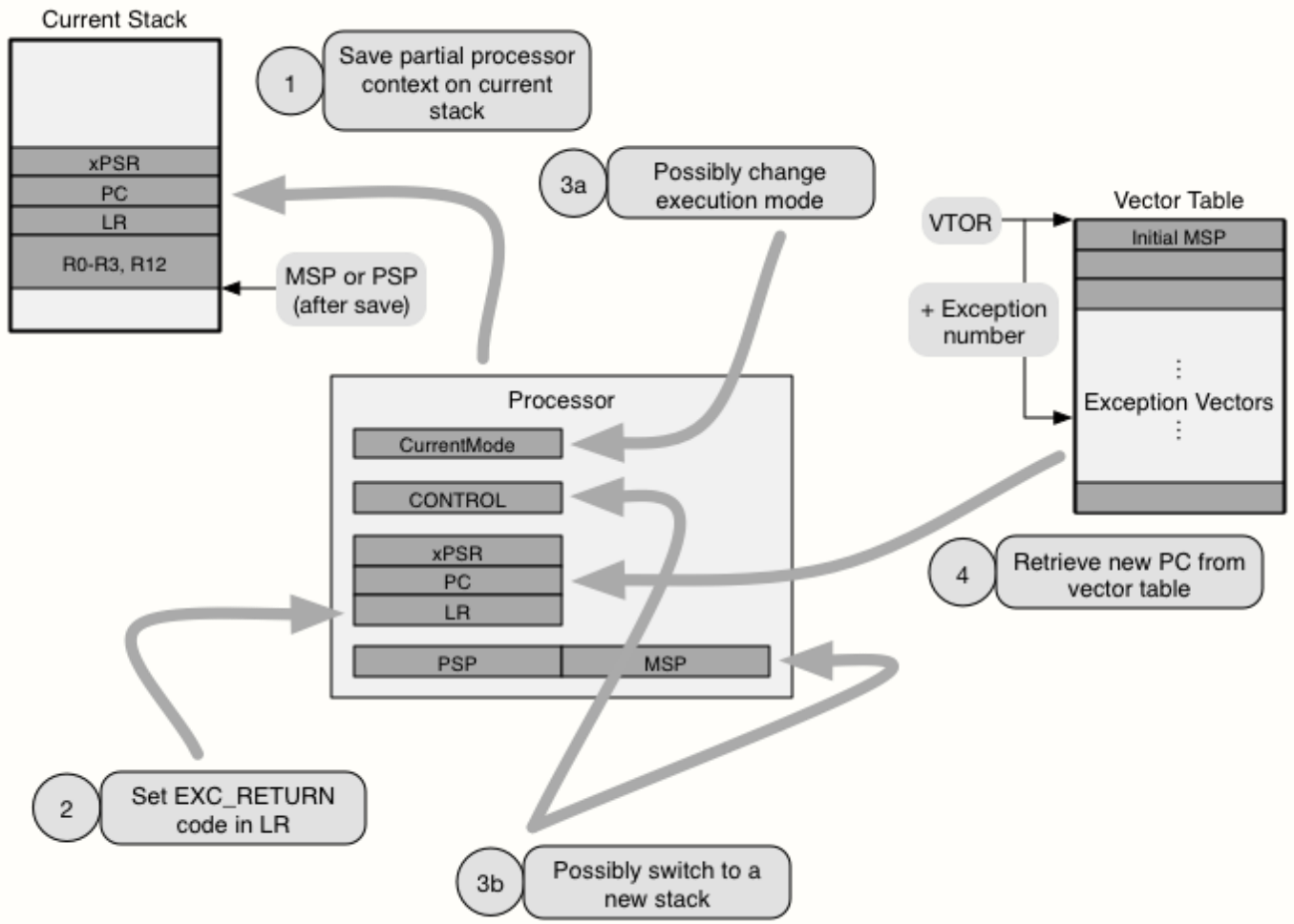


Image Courtesy: Cortex-M Exception Handling (Part 2) - Ivan Cibrario Bertolotti

What drove the evolution of ISA?

- As we look back at ISA evolved, we know the first ISA was called CISC, and then came RISC, then ARM and RISC-V
- When Flash Memory was expensive → CISC
- When simplicity and determinism was needed → RISC
- Dedicated ISA design & enhancements → ARM
- Open ecosystem → RISC-V

Instruction Set Architecture comparison

Feature	8051 (8bit)	PIC16 (8bit)	ARM (32 bit)	Thumb	Thumb-2
ISA style	**CISC**	**RISC**	**RISC**	**Compressed RISC**	**Hybrid RISC**
Instruction length	Variable	14-bit	32-bit	16-bit	16 + 32-bit
Memory access	Many instructions	Load/store	Load/store	Load/store	Load/store
Registers	Accumulator based	Small register file	16 registers	Same ARM registers	Same ARM registers
Code density	Moderate	Moderate	Lower	High	High
Performance	Moderate	Moderate	High	Slightly lower	High

ISA vs Microarchitecture
The Distinction That Changes How You Think About CPUs

Before we go deeper into ISA, lets discuss ISA and microarchitecture as they are the two most confused concepts in processor design.

ISA — Instruction Set Architecture:

- ISA once defined can be used across multiple core designs although the different cores (For example ARM's ISA Architecture ARMV6-M is used across ARM Cortex cores M0, M0+, M1)

Microarchitecture:

- The microarchitecture is the internal hardware implementation of the ISA within a given CPU core version.
- It defines how the CPU physically executes the instructions the ISA specifies — **using pipelines, caches, branch predictors, execution units, and other hardware structures.**
- Microarchitecture can change with every new CPU/Core generation. ***Two CPUs with identical ISAs can have completely different microarchitectures.*** The ISA is the "**what**" — the microarchitecture is the "**how**".

ISA vs Microarchitecture – ARM Cortex M Example #1

Example 1 — Same ISA, Different Pipeline Depth:

- Cortex-M3 — ARMv7-M ISA — 3-stage pipeline — Fetch, Decode, Execute.
- Cortex-M7 — ARMv7E-M ISA — 6-stage pipeline — Fetch, Decode, Dispatch, Issue, Execute, Writeback.
- Same ISA family. Binary compiled for Cortex-M3 runs on Cortex-M7 without recompilation. But Cortex-M7 executes the same binary 2 to 4 times faster because of its deeper pipeline, TCM, and separate I-Cache and D-Cache.
- **The ISA did not change. The microarchitecture did. The performance difference is entirely microarchitectural.**

ISA vs Microarchitecture – ARM Cortex M Example #2

Example 2 — Same ISA, Same Core, Different Vendor Implementation:

- STM32F407 — Cortex-M4 — ARMv7E-M — 168 MHz — CoreMark 566 — ART Accelerator.
- LPC4088 — Cortex-M4 — ARMv7E-M — 120 MHz — CoreMark 407 — No ART Accelerator.
- Both run identical ARMv7E-M binary.
- Both implement identical ISA. Zero ISA difference.
- Performance difference — entirely from clock frequency, Flash wait state handling, and bus matrix topology. All microarchitectural factors. All vendor-implementation factors.

Complex Instruction Set Computing (CISC) ISA

Complex Instruction Set Computing

CISC processors are designed to **reduce the semantic gap** between high-level programming languages and machine code **through sophisticated instruction sets**. (**Year: 1970s and early 1980s).**

CISC was the rational engineering response to the constraints of its time such as high cost of Flash memory, slower CPU etc.,

- **Complex multi-operation instructions** that can perform multiple tasks in a single instruction cycle, reducing program size.
- **Variable instruction lengths** ranging from 1 to 15 bytes, allowing flexible encoding of different operation complexities.
- **Rich addressing modes** supporting direct, indirect, indexed, and base-displacement memory access patterns.
- **Microcode implementation** translating complex instructions into sequences of simpler micro-operations for execution.

Why CISC Lost in Embedded Systems

Three key reasons:

Power consumption: CISC's complex decoding uses much more power than RISC; for example, Intel Atom uses over ten times the power of Cortex-M0 per MHz, reducing battery life.

Unpredictable timing: Variable-length CISC instructions cause inconsistent execution times and interrupt delays, unsuitable for real-time systems.

Memory cost got cheaper: Memory costs dropped significantly, making CISC’s goal of saving memory less important, while RISC’s simpler, faster hardware became preferable.

CISC is still used – in x86 Microprocessors

- **Legacy Support:** The x86 architecture is nearly 50 years old; Must maintain backward compatibility
- **Reduced Instruction Fetching due to dense Code:** CISC allows for smaller code size. This means more instructions can be packed into cache, reducing the need to access slower main memory and lowering memory bandwidth bottlenecks.
- **Compiler/Developer Ease:** CISC instructions can map more directly to higher-level language statements, making the compilation process more efficient.
- **Hybrid Approach:** Modern x86 chips use CISC for the instruction set (ISA) but break these complex instructions into RISC-like "micro-ops" internally. This combines the compatibility of CISC with the speed of RISC.

Reduced Instruction Set Computing (RISC) ISA

Reduced Instruction Set Computing

- Reduced Instruction Set Computer (RISC) design philosophy emphasizing simplicity and efficiency for improved processor performance. In 1980, [David Patterson (UC Berkeley) and John Hennessy (Stanford)]concluded that 80% CPU time is spent on a small set of simple instructions.
- Most RISC ISAs (like ARM) are proprietary, requiring licenses and royalties.
- RISC-V is an open-source, free ISA managed by RISC-V International.

Simple fixed-length instructions that execute in single clock cycles for predictable performance.

Load-store architecture with large register files to minimize memory access overhead.

Simplified addressing modes for efficiency and reduced instruction complexity.

Pipeline optimization for compiler performance and instruction-level parallelism.

CISC Vs RISC (Complex Vs Simple instructions)

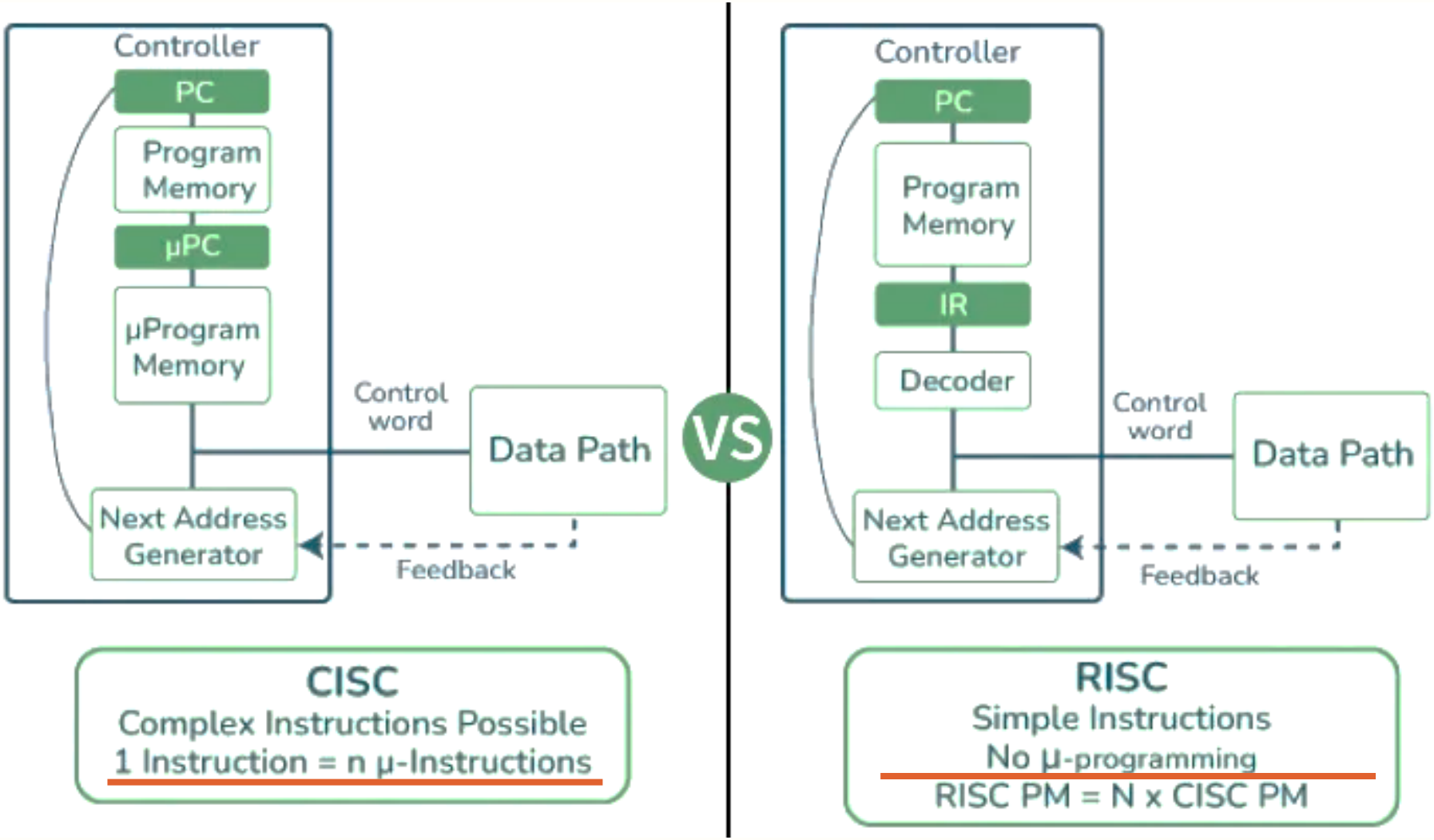


Image Courtesy: https://www.geeksforgeeks.org/computer-organization-architecture/computer-organization-risc-and-cisc/

CISC Vs RISC (Variable length Vs Fixed Length instructions)

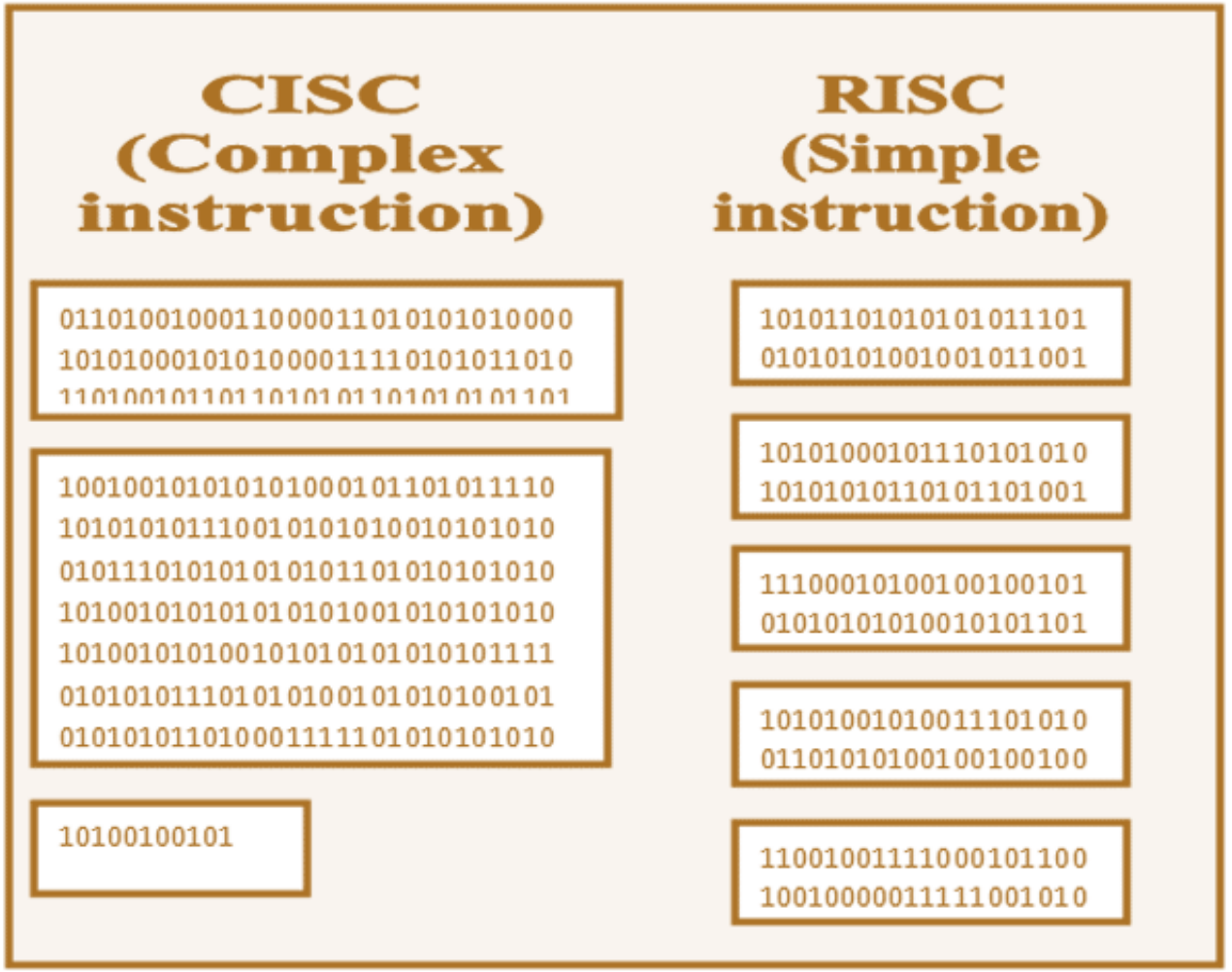


Image Courtesy: Difference Between RISC And CISC Embedded Architecture – PCB HERO

Core Principles of RISC #1

Principle 1 — Load/Store Architecture:

- Only LOAD (LDR) and STORE(STR) instructions can interact with memory
- All arithmetic and logical operations are designed to work solely on CPU registers (R0 to R12).

Example:

```
LDR   R0, [R1]      ; Step 1: Load value from memory (address given in R1) into R0
ADD   R2, R2, #10   ; Step 2: Add 10 (operation only on registers not affecting  the registers of LDR, STR,
; this instruction stuffing is done by compiler to make use of the wait state causes by memory access)
STR   R0, [R1]      ; Step 3: Stores R0 to the memory location given by R1
```

Note: If the memory access is to the SRAM off the CPU, the wait state can be 0 to 1 cycle. If the memory access is to the Flash area, then the wait state can be more. Hence the compiler does try to schedule more independent instructions after the LDR to hide the extra latency — if those instructions are available and independent. The compiler's scheduler is latency-aware. It knows the approximate load latency for different memory regions

Such a flexibility is not possible in CISC ISA instructions that directly work on memory.

Core Principles of RISC #2

Principle 2 — Fixed Instruction Length: Every instruction has a uniform length, typically 32 bits.
Significance —enabling highly efficient and uniform pipeline stages **conducive to deep pipelining.**

Principle 3 — Large Register File: A substantial number of general-purpose registers, usually between 16 and 32, are available

Significance — **Registers provide single-cycle, low-power access compared to memory.** For example, the ARM Cortex-M includes 16 registers (R0–R15)

Core Principles of RISC #3

Principle 4 — Hardwired instruction decode logic: Instruction decode logic is implemented through combinational hardware logic rather than microcode lookup tables.

Significance — **Hardwired decoding is faster, simpler, and more energy-efficient. It enables decoding within a single clock cycle.**

Principle 5 — Compiler-Centric Complexity: RISC architectures transfer complexity from hardware to software, specifically to the compiler.

The compiler handles tasks such as instruction scheduling, register allocation, loop unrolling, and producing efficient sequences of simple instructions.

RISC vs CISC — The Definitive Comparison

Feature	CISC	RISC
Instruction complexity	High — multi-operation	Low — single operation
Instruction length	Variable — 1 to 15 bytes	Fixed — typically 32 bits
Instructions to do same work	Fewer	More
Code size	Smaller	Larger
Decode complexity	High — microcode	Low — hardwired
Pipeline efficiency	Lower — variable length stalls	Higher — uniform stages
Execution time per instruction	Variable — 1 to many cycles	Predictable — typically 1 cycle
Power consumption	Higher — complex decode	Lower — simple hardware
Real-time determinism	Lower — variable timing	Higher — predictable timing
Compiler complexity	Lower — rich instructions	Higher — must schedule efficiently
Dominant use today	Desktop / Server (x86)	Mobile / Embedded / MCU (ARM)

Reduced Instruction Set Computing - V

(RISC-V) ISA

Open + modular

RISC-V – An Opensource ISA

RISC-V was created at UC Berkeley. Royalty free.

Pros

- No Royalties; Extreme ISA Customization possible; ISA open for audit

Cons

- Fragmentation: Because anyone can customize it, "Standard RISC-V" can become fragmented. Software written for one RISC-V chip might not run optimally on another if they use different custom extensions.
- Maturity Gap: While catching up quickly, the software ecosystem (debuggers, optimized libraries) is not yet as "plug-and-play" as ARM's.
- Verification Burden: If you design your own RISC-V core, you are responsible for the hardware verification. This is a massive engineering effort compared to buying a pre-verified Cortex-M core.

RISC-V ISA – Base Technical Design

RISC-V was designed with clean-slate hindsight — learning from 40 years of ISA design mistakes. Its architecture reflects this:

Base Integer ISA — RV32I:

- **32 general-purpose registers** — x0 through x31. 32-bit fixed instruction width in the base ISA.
- **47 base instructions** — deliberately minimal. Every implementation must support all 47.
- **Load/Store architecture** — exactly as RISC principles dictate.
- **Simple clean encoding** — opcode always in bits 6-0, destination register always in bits 11-7, source registers always in bits 19-15 and 24-20. Completely regular.

RISC-V ISA – Modular Extension

Modular Extension System — The Key Design Innovation:

- Unlike ARM where the ISA is monolithic — you get the whole thing — RISC-V uses a modular extension system. You implement the base ISA plus only the extensions you need.
- A typical embedded RISC-V MCU implements **RV32IMAC** — base integer plus multiply plus atomic plus compressed instructions. The C extension is particularly important — it gives RISC-V code density comparable to ARM Thumb-2.

Extension	Letter	What It Adds
Integer multiply/divide	M	MUL, DIV, REM instructions
Atomic operations	A	Atomic read-modify-write for RTOS
Single precision float	F	32-bit IEEE 754 floating point
Double precision float	D	64-bit IEEE 754 floating point
Compressed instructions	C	16-bit compressed instruction encoding
Vector operations	V	SIMD vector instructions
Bit manipulation	B	Bit counting, rotation, extraction

RISC-V vs ARM — The Definitive Comparison

Factor	ARM Cortex-M	RISC-V
Licence cost	Paid — per-chip royalty	Free — zero royalty
ISA maturity	40 years — battle tested	15 years — maturing rapidly
Silicon implementations	Hundreds — ST NXP Nordic Infineon	Growing — Espressif SiFive WCH GigaDevice
Toolchain maturity	Excellent — GCC Clang IAR Keil	Good — GCC Clang — IAR adding support
RTOS support	Excellent — FreeRTOS Zephyr ThreadX	Good — FreeRTOS Zephyr support
Community size	Very large — millions of developers	Growing — hundreds of thousands
Documentation quality	Excellent — application notes	Variable — improving rapidly
Eval board availability	Extensive — every major vendor	Limited but growing
China availability	Restricted — US export considerations	Unrestricted — strategic advantage
Code density	Excellent — Thumb-2	Good — RV32C comparable to Thumb
Power efficiency	Excellent — M0+ best in class	Comparable — implementation dependent
Debug standard	ARM CoreSight — mature	RISC-V Debug spec — maturing
Interrupt latency	Defined — 12 cycles on Cortex-M	Implementation defined — varies

Summary of what we have covered so far

1) What is ISA
2) Evolution of ISA
3) ISA Vs Micro architecture
4) CISC, RISC, RISC-V

Next, we will spend good time to discuss on the ISA from “ARM” as it has become a widely licensed CPU as on date

ARM ISA evolution

Introduction - Advanced RISC Machines (ARM)

Arm (formerly ARM - Advanced RISC Machines) is a **British semiconductor and software design company** that **creates the foundational intellectual property (IP) for processors (CPUs)**

Founded in 1990, Arm does not manufacture its own chips; instead, it licenses IP.

Ownership & Structure: Arm is now(Mar2026) a publicly listed company (NASDAQ: ARM) predominantly owned (roughly 90%) by the Japanese conglomerate SoftBank Group.

Arm China: In 2018, Arm sold 51% of its Chinese subsidiary (Arm China) to Chinese investors. However, this is a joint venture, and the main global IP company is not owned by China.

Based in Cambridge, England, Arm is considered a "Switzerland" of the tech industry because it licenses to all, **though its recent move into making its own chips has begun to shift this neutral position** (a production-ready chip built for running inference in an AI data center, Meta as end customer).

The ARM ISA Family — Gen 1 — ARM (32-bit) — 1985

- The original ARM ISA. Every instruction is exactly 32 bits wide (4 Bytes) — fixed length — pure RISC. Clean, fast, pipeline-friendly. Executes in one clock cycle. **Excellent performance.**

- **The problem it created — Not so optimized code density.**
- Every instruction occupies 4 bytes in Flash. A program that needs 1000 instructions occupies 4000 bytes minimum which was okay for desktop workstations
- Foe embedded applications, 4 bytes per instruction was too expensive.

The ARM ISA Family — Gen 2 — Thumb (16-bit) — 1995:

- **Thumb: ARM's solution to the code density problem.** A compressed subset of the ARM ISA encoded in 16 bits instead of 32. Average **code size reduction of 30% compared to 32-bit ARM.**
- **The problem — performance and capability.**
- Not every ARM instruction has a Thumb equivalent.
- Thumb instructions can only access the low 8 registers — R0 through R7 — limiting compiler flexibility.
- Some operations require multiple Thumb instructions. Performance drops approximately 10–15% compared to pure ARM on the same hardware.
- **Additionally — Thumb and ARM were separate modes.** The CPU could operate in ARM state or Thumb state — **switching required an explicit BX instruction — a branch-and-exchange.** Code had to decide upfront which mode to use.

The ARM ISA Family — Gen 3 — Thumb-2 — 2003:

- **Solution to Thumb's limitations.** Thumb-2 is not a new ISA — it is an extension to Thumb that adds 32-bit instructions to the 16-bit Thumb encoding space. The result — no mode switching required.
- The CPU determines whether each instruction is 16-bit or 32-bit by examining the first few bits — if bits 31:29 are 111 — it is a 32-bit Thumb-2 instruction. Otherwise 16-bit. This determination happens in the fetch stage with no pipeline penalty.
- **Thumb-2 achieves code density close to Thumb — while achieving performance close to pure ARM** — within 5% on typical workloads.
- All ARM Cortex-M3 and above use Thumb-2 exclusively. There is no ARM state in Cortex-M — Thumb-2 is the only ISA the CPU understands.

Did you notice an interesting pattern in ARM ISA evolution?

1) Performance (ARM ISA)
2) Code Density (Thumb ISA)
3) Why not both? (Thumb-2 ISA)

Doesn't it sound like the evolution of Memory architecture from Von-Neumann → Harvard → Modified Harvard

The ARM Architecture Version Numbers — Clearing Up Confusion

Cortex-M Core	Year	Architecture Version	Key Features / Notes
Cortex-M3	2004	**ARMv7-M**	First Cortex-M; full Thumb-2; NVIC; bit-banding; widely used in STM32F1 etc.
Cortex-M1	2007	**ARMv6-M**	FPGA-optimized variant of early low-end design.
Cortex-M0	2009	**ARMv6-M**	Ultra-small, low-power entry-level; ~56 instructions.
Cortex-M4	2010	**ARMv7E-M**	Adds DSP/SIMD extensions + optional FPU; popular for signal processing.
Cortex-M0+	2012	**ARMv6-M**	Improved power efficiency over M0; optional MPU (8 regions).
Cortex-M7	2014	**ARMv7E-M**	Highest performance in v7 series; optional caches + TCM; dual-issue pipeline.
Cortex-M23	2016	**ARMv8-M Baseline**	Low-power successor to M0/M0+; optional TrustZone; hardware divide.
Cortex-M33	2016	**ARMv8-M Mainline**	Successor to M3/M4; optional TrustZone + DSP; better security.
Cortex-M35P	2018	**ARMv8-M Mainline**	M33 with physical security features (tamper resistance, side-channel protection).
Cortex-M55	2020	**ARMv8.1-M Mainline**	Introduces **Helium** (vector extension for ML/DSP); optional caches.
Cortex-M85	2022	**ARMv8.1-M Mainline**	High-performance with Helium; branch prediction; strong ML focus.
Cortex-M52	2023	**ARMv8.1-M Mainline**	Mid-range with Helium; balances performance, power, and security.

An interesting feature of ARM -ISA Compatibility Across the cores

- A motor control library compiled for Cortex-M4 (ARMv7E-M) links and runs correctly on Cortex-M33 (ARMv8-M Mainline) — because ARMv8-M Mainline ISA is backwards compatible with ARMv7E-M ISA. The ISA contract is honored.
- But the Cortex-M33 adds TrustZone — a microarchitectural addition — and the Cortex-M55 adds Helium vector registers — an ISA extension. These are additive — they do not break existing binaries.

ARM ISA variations – Practical implications for developer #1

Four practical takeaways for a working engineer:

Takeaway 1 — You never choose the ISA manually:

Your compiler targets the correct ISA automatically based on the ***-mcpu*** or ***-march flag***.

gcc -mcpu=cortex-m4 generates Thumb-2 with DSP.

gcc -mcpu=cortex-m0 generates ARMv6-M Thumb subset.

You configure the compiler once — it handles everything.

Takeaway 2 — Code density is automatic:

The compiler uses 16-bit Thumb-2 instructions wherever possible and 32-bit instructions only when necessary.

You do not specify which width — the compiler decides per instruction. Optimization flag -Os (optimize for size) pushes harder toward 16-bit encoding.

ARM ISA variations – Practical implications for developer #2

Takeaway 3 — ISA compatibility means firmware portability:

A library compiled for ARMv7E-M (Cortex-M4) links and runs on any ARMv7E-M or ARMv8-M Mainline target.

Vendor HAL libraries, RTOS kernels, DSP libraries — all pre-compiled and compatible across the ISA family. The ARM ecosystem benefits directly from this compatibility.

Takeaway 4 — Disassembly is readable:

When debugging at assembly level — knowing Thumb-2 encoding helps.

A 4-byte instruction starting with 0xF or 0xE8 is a 32-bit Thumb-2 instruction.

A 2-byte instruction is a 16-bit Thumb instruction.

Most debuggers — Ozone, STM32CubeIDE, Keil — show disassembly with instruction widths labeled automatically.

Is ISA alone good enough? How does the major CPU IP supplier ARM go beyond ISA?

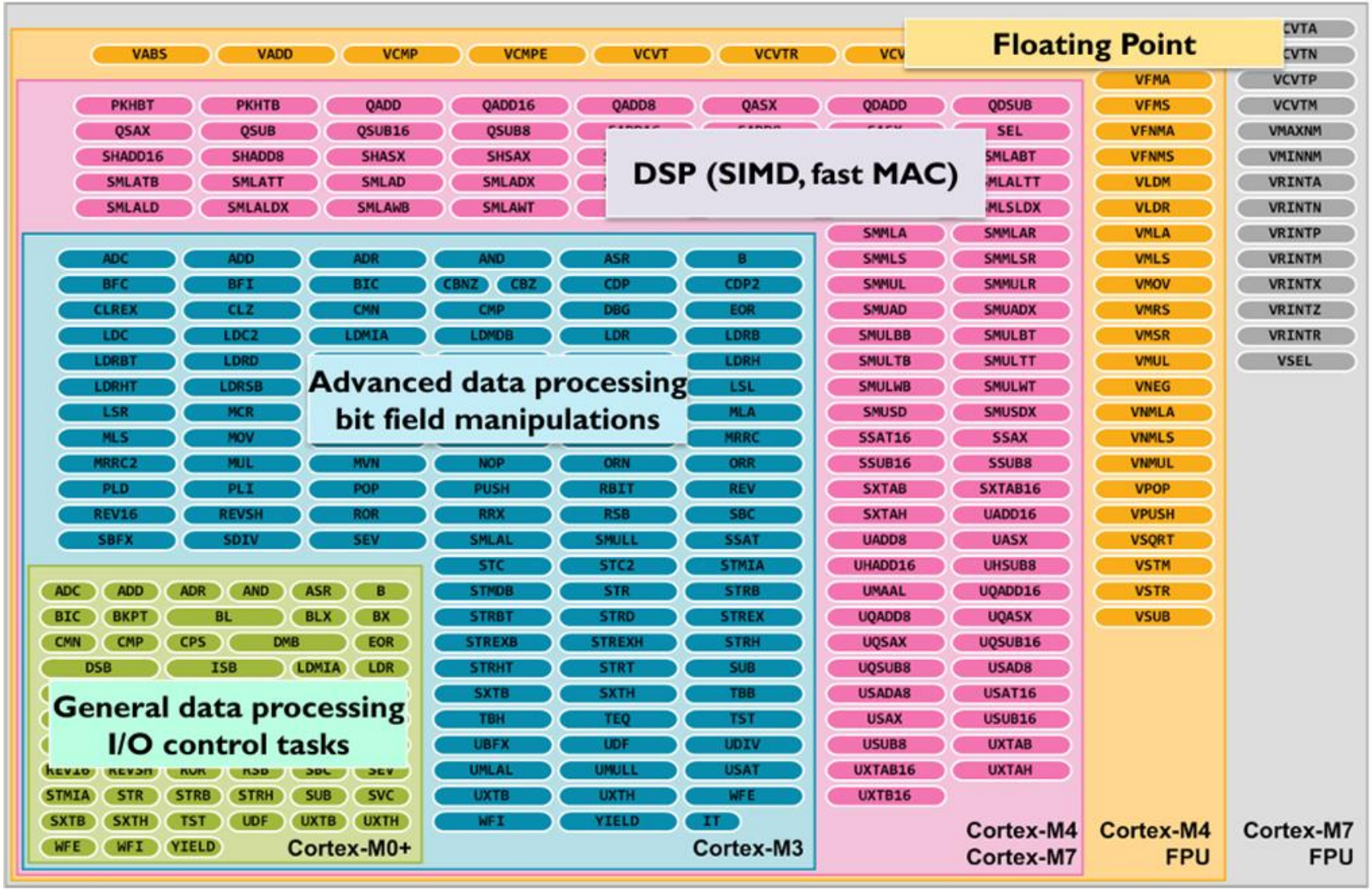
- ARM ISA Extensions speed up math heavy operations
- ARM Accelerators acts as a dedicated CPU itself to accelerate the processing for specific applications
- ARM Software libraries make software development modular and Easier

ARM Extensions

What is ARM ISA Extension?

- **An ARM Extension is a formally defined, optional set of new instructions, registers, execution modes, or behaviors that extend the capabilities of a specific ARM architecture version (e.g., ARMv7-M, ARMv8-M, ARMv8.1-M) while preserving backward compatibility with existing code.**
- ARM Extensions are **part of the Architecture (ISA)**
- These extensions define **new programmer-visible behavior.** Compilers and libraries (CMSIS-DSP, CMSIS-NN) generate code that uses these instructions when the target ISA supports them.

A glance at ISA extension in ARM MCUs



The ARM ISA Extension — Important Takeaways

- **Almost all standard "ARM Extensions" in Cortex-M are optional hardware blocks supplied by ARM.** The MCU vendor pays extra (in licensing) and includes them in the silicon when they fabricate the chip.
- **The only major exception is Arm Custom Instructions** (CDE) — here ARM gives you a clean "plug-in" slot, but the actual custom acceleration logic is designed by the MCU vendor. Because CDE was introduced with the ARMv8-M (Cortex-M33) and ARMv8.1-M (Cortex-M55/M85) architectures, it is still relatively new to the mass market.
- Pure ISA-only extensions (no extra hardware) are rare in Cortex-M. Most extensions you see advertised come with dedicated silicon from ARM.

This is why different MCUs with the "same" Cortex-M4 can have very different capabilities (some have FPU + DSP, some have only DSP, some have neither) — it depends on which optional ARM IP blocks the vendor chose to include.

The ARM DSP Extension — Single Cycle MAC

Calculating: Acc = Acc + (R1 * R2)

With standard ARM ISA, no DSP extension

```
MUL  R3, R1, R2    ; Step 1: Multiply R1 and R2, store in R3
ADD  R0, R0, R3    ; Step 2: Add R3 to the accumulator (R0)
```

With DSP extension

```
MLA  R0, R1, R2, R0  ; Single Step: Multiply R1 & R2 and ADD to R0
```

The ARM DSP Extension — Floating Point Unit

32 x 32 floating point Multiplication: M3 Vs M7 ARM cores

```
LDR    r0, [sp, #4]    ; Load first float into R0
LDR    r1, [sp, #8]    ; Load second float into R1
BL     __aeabi_fmul    ; Call software emulation routine
STR    r0, [sp, #12]   ; Store result (returned in R0)
```

```
VLDR    s0, [r7, #4]    ; Load float from memory into FPU register S0
VLDR    s1, [r7, #8]    ; Load float from memory into FPU register S1
VMUL.F32 s2, s0, s1     ; Hardware floating-point multiplication (32-bit)
VSTR    s2, [r7, #12]   ; Store the result from S2 back to memory
```

Feature	Cortex-M3	Cortex-M7
Mechanism	Software Library (__aeabi_fmul)	Hardware FPU Instruction (VMUL.F32)
Registers	General Purpose (R0-R12)	Floating Point (S0-S31)
Code Size	High (Includes library overhead)	Low (Single opcode)
Cycle Count	~50 to 150+ cycles	1 cycle (typically)

Why the M7 is significantly faster: Beyond just having the VMUL.F32 instruction, the **M7 is a superscalar processor.** It can often "hide" the floating-point multiplication by executing an integer instruction (like an address calculation) in a second pipeline at the exact same time. **On the M3, the processor is completely "stuck" executing the dozens of instructions inside the __aeabi_fmul function.**

Example: 128 Points FFT – Algorithm requirements

*"128-point FFT requires **7 stages** with **448 butterfly operations** totaling **896 complex multiplications** before DSP optimization."*

COMPUTATIONAL STAGES

Seven processing stages ($\log_2(128)$) with structured data flow and memory access patterns.

BUTTERFLY OPERATIONS

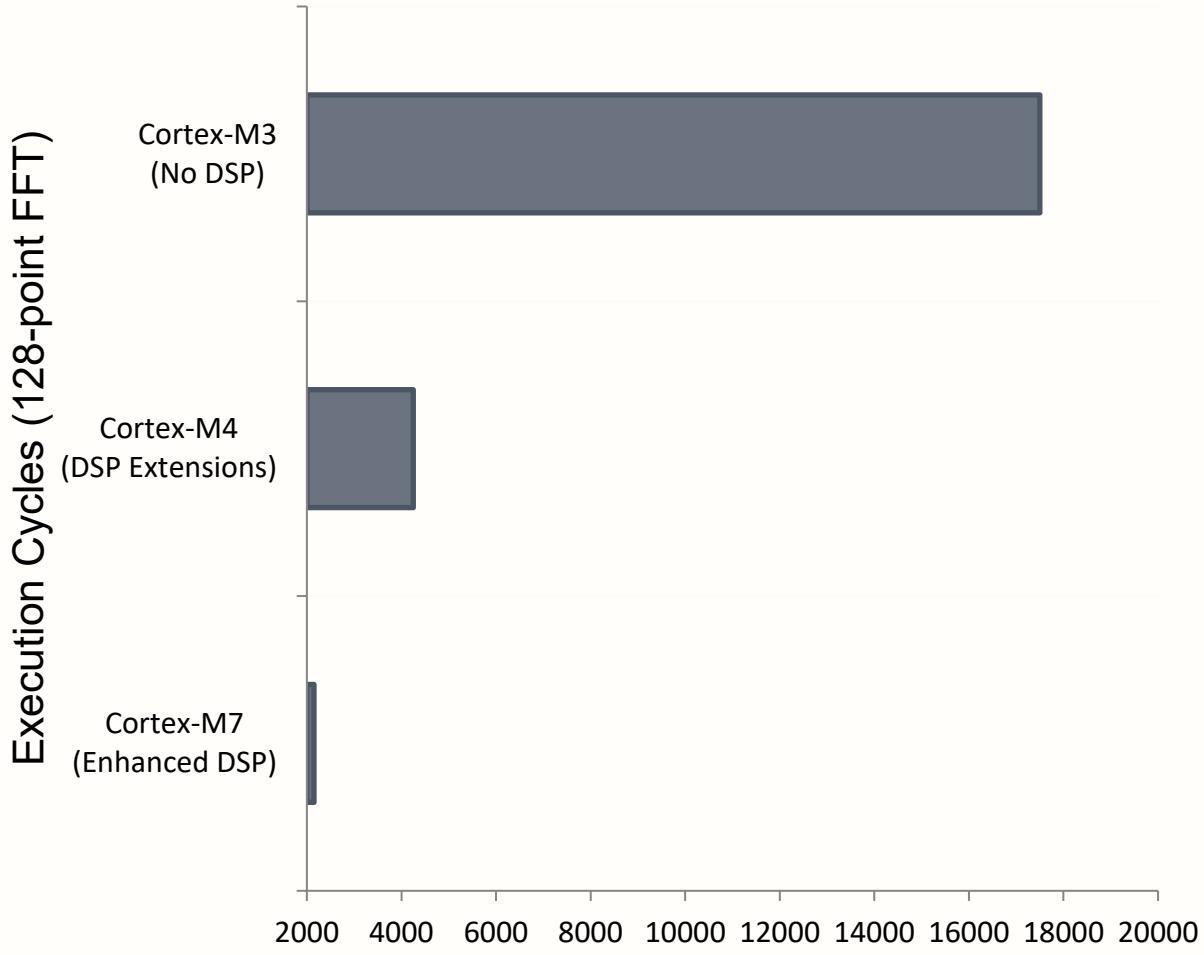
Each butterfly involves 1 complex multiplication and 2 complex additions/subtractions per stage.

DSP OPTIMIZATION

Parallel processing and single-cycle MAC operations dramatically reduce computational overhead.

```c
#include <stdio.h>
#include <math.h>
#include <complex.h>

#define PI 3.14159265358979323846
#define N 128

// Bit-reversal Permutation
void bit_reverse(double complex *X) {
    for (int i = 1, j = 0; i < N; i++) {
        int bit = N >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j) {
            double complex temp = X[i];
            X[i] = X[j];
            X[j] = temp;
        }
    }
}

// FFT Routine
void fft(double complex *X) {
    bit_reverse(X);

    // Iterative Butterfly computation
    for (int len = 2; len <= N; len <<= 1) {
        double angle = -2.0 * PI / len;
        double complex wlen = cos(angle) + I * sin(angle);

        for (int i = 0; i < N; i += len) {
            double complex w = 1.0 + 0.0 * I;
            for (int j = 0; j < len / 2; j++) {
                double complex u = X[i + j];
                double complex v = X[i + j + len / 2] * w;
                X[i + j] = u + v;
                X[i + j + len / 2] = u - v;
                w *= wlen;
            }
        }
    }
}
```

Example: 128 Points FFT with and without Extension



Performance Analysis

- **Cortex-M4 delivers 3-4x improvement over M3** through DSP extensions enabling parallel SIMD operations and single-cycle MAC instructions.
- **Cortex-M7 achieves 6-8x speedup** with enhanced DSP capabilities, advanced pipeline architecture, and optimized memory access patterns.
- **DSP extensions provide compound benefits:** 2x from parallel SIMD, 2x from single-cycle MAC, and 1.5x from memory optimization.
- **Dramatic cycle reduction** from ~17,500 cycles (M3) to ~2,150 cycles (M7) transforms real-time processing capabilities for embedded applications.

Conclusion - 128 Points FFT with and without Extension

Performance Gains

DSP extensions deliver **3-8x FFT speedup** with **40% code size reduction**, significantly improving computational efficiency for signal processing applications.

Processor Selection

Cortex-M4 provides optimal price/performance for moderate DSP requirements, while **Cortex-M7** excels in intensive signal processing scenarios.

Cost-Benefit Analysis

Trade-offs between performance gains and resource requirements are **favorable for most applications** requiring FFT processing capabilities.

Implementation Guidelines

Choose DSP-enabled processors when FFT operations are frequent, prioritizing **energy efficiency** and **real-time performance** requirements.

ARM Cortex-M Architectural Extensions (as of early 2026) #1

Extension	What it adds	ARM provides full silicon IP?	Details / Notes
DSP Extension (SIMD, MAC)	Specialized instructions for signal processing (multiply-accumulate, saturation, SIMD)	**Yes** (hardware datapath)	ARM-designed MAC units + SIMD logic inside Cortex-M4/M7/M33/M55 etc.
FPU (Floating-Point Unit)	Hardware support for single/double-precision floating-point	**Yes** (dedicated FPU hardware)	FPv4-SP or FPv5 in Cortex-M4/M7/M33/M55. ARM supplies the floating-point pipeline and registers.
TrustZone (Security Extension)	Hardware isolation between Secure and Non-Secure worlds	**Yes** (security partitioning logic)	Armv8-M Security Extension. Full hardware state machines, memory tagging, and bus separation from ARM.
Helium / MVE (M-Profile Vector Extension)	Vector/SIMD processing for AI/DSP/ML (128-bit vectors)	**Yes** (vector hardware)	In Cortex-M55, M85, M52. ARM-designed vector unit that re-uses existing FPU registers.

ARM Cortex-M Architectural Extensions (as of early 2026) #2

Extension	What it adds	ARM provides full silicon IP?	Details / Notes
MPU (Memory Protection Unit)	Region-based memory protection and access control	**Yes** (hardware MPU block)	Optional in almost all Cortex-M cores. ARM provides the protection hardware.
Arm Custom Instructions (CDE)	Ability to add your own custom instructions	**Partial** (ARM provides framework)	ARM supplies the instruction decode + register interface; vendor implements their own custom datapath logic. First introduced for M33/M55/M85.

ARM Cortex-M Architectural Extension benefits #1

Feature	Description / Use	Which Cortex-M Cores Support It	End-Application Benefits
Hardware Divide	Single-cycle or multi-cycle integer divide.	All except M0/M0+/M1 (M23+ have improved divide)	General embedded math (PID, scaling); avoids slow software division routines.
DSP Extension (SIMD)	Single-Instruction Multiple-Data instructions + saturating arithmetic + fast MAC.	M4, M7 (v7E-M), M33, M35P, M55, M85 (v8-M Mainline)	Audio filtering, motor control, image processing, vibration analysis. 2–4× faster DSP algorithms vs non-DSP cores.
Floating Point Unit (FPU) – Single Precision (SP) (FPv4-SP or FPv5-SP)	Hardware acceleration for 32-bit IEEE 754 single-precision floating-point ops (add, mul, div, MAC, sqrt, trig via library).	M4, M7, M33, M35P, M52, M55, M85 (M4 uses FPv4-SP; others FPv5-SP)	Motor control (FOC/PID), audio processing, sensor fusion, graphics, ML inference. 5–10× speedup vs software float; lower power than double.

ARM Cortex-M Architectural Extensions benefits #2

Feature	Description / Use	Which Cortex-M Cores Support It	End-Application Benefits
Floating Point Unit (FPU) – Double Precision (DP) (FPv5-DP)	Hardware 64-bit IEEE 754 double-precision ops (full support in FPv5).	M7, M52, M55, M85 (optional alongside SP)	High-accuracy industrial control, scientific instrumentation, medical devices, high-end audio/DSP where single-precision rounding errors matter.
Floating Point Unit (FPU) – Half Precision (HP)	16-bit floating-point (via Helium vector instructions).	M55, M85 (as part of Helium/MVE)	Edge AI/ML (tiny neural nets), computer vision, low-power signal processing. Dramatically higher throughput for vector math.
Helium (M-Profile Vector Extension – MVE)	128-bit vector processing (int8/16/32, float16/32) + low-overhead loops.	M55, M85, M52 (v8.1-M Mainline)	AI/ML inference at the edge, advanced DSP, voice recognition, computer vision. Up to 4× performance uplift in vector-heavy code.

The ARM ISA Extension Stack — How It Builds Up

Level 1 — Base Thumb-2 — ARMv7-M — Cortex-M3: General purpose integer arithmetic. Load/Store. Branches. Basic data processing. Available on all Cortex-M3 and above.

Level 2 — DSP Extensions — ARMv7E-M — Cortex-M4: Adds single-cycle 32×32 MAC. Dual 16×16 SIMD MAC. Saturating arithmetic. Parallel 8-bit and 16-bit operations. Available on Cortex-M4, M7, M33, M35P, M55, M85.

Level 3 — FPU — Optional on ARMv7E-M and above: Adds 32 single-precision float registers. Hardware float multiply, add, divide, square root. Fused multiply-accumulate. Available on Cortex-M4F, M7, M33F, M35P, M55, M85.

Level 4 — Helium MVE — ARMv8.1-M — Cortex-M55: Adds 8 × 128-bit vector registers. Integer and float vector operations. Tail predication. Gather/scatter memory operations. Available on Cortex-M55 and Cortex-M85.

Practical Guidance — When Do You Need Each Level

Application	Base Thumb-2	DSP Extensions	FPU	Helium
Simple sensor threshold detection	✅ Sufficient	❌ Not needed	❌ Not needed	❌ Not needed
UART communication protocol	✅ Sufficient	❌ Not needed	❌ Not needed	❌ Not needed
PID control loop — integer math	✅ Sufficient	✅ Beneficial	❌ Optional	❌ Not needed
PID control loop — float math	✅ Insufficient alone	✅ Needed	✅ Essential	❌ Not needed
FOC motor control — full	✅ Base needed	✅ Essential	✅ Essential	❌ Not needed
Audio codec — FIR/IIR filters	✅ Base needed	✅ Essential	✅ Beneficial	❌ Optional
Keyword spotting — ML inference	✅ Base needed	✅ Needed	✅ Needed	✅ Transformative
Image classification on device	✅ Base needed	✅ Needed	✅ Needed	✅ Essential
Anomaly detection — vibration	✅ Base needed	✅ Essential	✅ Essential	✅ Beneficial

ARM Accelerators

ARM Accelerators

ARM Accelerators for the Cortex-M family primarily refer to **dedicated, licensable hardware IP blocks** from Arm that offload compute-intensive tasks (especially **machine learning inference**) from the Cortex-M CPU core.

What They Are

- Unlike **ARM Extensions** (which are ISA-level additions like Helium/MVE, DSP SIMD, or FPU that add new instructions inside the core), **ARM Accelerators** are **separate hardware units** (often called **microNPUs** — micro Neural Processing Units).
- They connect to the Cortex-M via high-speed buses (AXI/AMBA) and are controlled by the Cortex-M acting as the **host processor**. The CPU handles control logic, pre/post-processing, and unsupported operations, while the accelerator delivers massive throughput for specific workloads.
- These are not part of the Cortex-M core itself — they are optional IP that silicon vendors license and integrate into their SoCs/MCUs. Example : Ethos-U Family (microNPUs)

Key Characteristics of Arm Accelerators

- **Purpose**: Accelerate neural network operators (CNNs, RNNs, some transformers) with high efficiency — especially quantized int8/int16 workloads.
- **Performance uplift**: Up to 480× faster ML inference compared to Cortex-M CPU alone (depending on model and configuration). Heavy operators run in hardware; fallback to CPU (often boosted by **Helium** vector extension).
- **Programming**: Uses **Arm Vela compiler** (optimizes TensorFlow Lite Micro models for the NPU) + runtime driver on the Cortex-M. Integrates well with **CMSIS-NN**.
- **Typical pairing**: Best with modern cores like **Cortex-M55**, **M85**, or **M52** (which have **Helium** for vector DSP/ML support). Cortex-M + Helium handles general tasks; Ethos-U handles heavy inference.
- **Reference subsystems**: Arm provides **Corstone** platforms (e.g., Corstone-300/310/320) that bundle Cortex-M + Ethos-U + other IP for faster SoC design.

Dedicated ARM Accelerators / Companion IP (Separate Hardware Blocks)

Accelerator	Type	Performance (typical)	Compatible Cortex-M Hosts	Primary Use & Benefit	Notes / Examples
Ethos-U55 (microNPU)	Neural Processing Unit for ML inference (CNNs, RNNs)	32–256 MACs; up to ~0.5 TOPS	M4, M7, M33, M55, M85 (best with Helium cores)	Object detection, keyword spotting, speech recognition, anomaly detection. Up to 480× ML uplift vs. CPU-only.	Most common microNPU for Cortex-M. Used in Renesas RA8, Synaptics, Alif Ensemble, Arm Corstone-300/310.
Ethos-U65	Enhanced microNPU (higher MAC count, better DRAM support)	256–512 MACs; up to 1 TOPS	M55, M7 (also works in mixed M+A systems)	Similar to U55 but for more demanding or heterogeneous SoCs.	Good for industrial/IoT with some DRAM.
Ethos-U85	High-performance microNPU (supports transformers)	128–2048 MACs; up to 4 TOPS	M7, M55, M85	Advanced edge AI: pose estimation, image segmentation, NLP tasks. 20% better efficiency than predecessors.	Newer option; pairs with Corstone-320 (includes Mali-C55 ISP).

Accelerators for ARM from specific MCU Vendors

What is Vendor specific Accelerators in ARM MCUs

- In the world of ARM Microcontrollers (MCUs), Vendor-Specific Accelerators are hardware modules designed by silicon manufacturers (like STMicroelectronics, NXP, or Renesas) to perform specialized tasks more efficiently than the standard ARM CPU core could do on its own to solve specific engineering challenges like motor control, encryption, or complex math.

Path Component	ISA Extension (FPU/Helium)	Vendor Accelerator (TMU/PowerQuad)
Instruction Used	**Functional Opcodes** (VADD, VMUL)	**Data Movement Opcodes** (using LDR, STR)
Target	Internal CPU Registers ($S0, Q1$)	External Peripheral Registers (Memory Map)
Bus Access	None (Internal to Pipeline)	Needs the System Bus (AHB/APB)
Decoding	Decoded by ARM Instruction Decoder	Decoded by Peripheral Address Decoder

- "An ISA Extension adds new 'verbs' (instructions) to the CPU's language. A Vendor Accelerator adds new 'places' (memory addresses) that the CPU visits using standard 'verbs' like Load and Store."

Vendor-Specific Accelerators

Vendor	Accelerator	Description / Use Case	Supported Cores (Typical)	Key Benefits & Applications
ST	CORDIC	Trigonometric, hyperbolic, exp/log, sqrt (fixed-point)	Cortex-M4/M7/M33 (G4, U5, H5, H7)	10–20× faster trig/math in FOC motor control, metering, graphics
ST	FMAC (Filter Math Accelerator)	FIR/IIR filters, convolution, vector MAC with local buffer	Cortex-M4/M7 (G4, some H7)	Digital power supplies, audio filtering, compensators; parallel to CPU
ST	HSP (Hardware Signal Processor)	FFT, complex/real signal processing (fixed/floating)	Cortex-M33 (newer U3 series & upcoming)	9–13× faster spectral analysis; low-power sensing, vibration, edge DSP
ST	Neural-ART Accelerator (NPU)	Proprietary neural processing unit (~300 MACs)	Cortex-M55 (STM32N6 series)	Up to 600× ML inference uplift; edge AI in high-end STM32
TI	TinyEngine NPU	Dedicated Neural Processing Unit for tinyML	Cortex-M0+ (MSPM0G5xx), Cortex-M33 (AM13Ex)	90–120× lower latency/energy for AI inference; wearables, appliances, motor AI
TI	Trigonometric Math Accelerator	Fast hardware trig calculations	Cortex-M33 (AM13Ex series)	Multi-motor control + AI; reduces external components
NXP	eIQ Neutron NPU (in select MCX)	Neural Processing Unit for ML	Cortex-M33 (some MCX N series)	TinyML inference (anomaly detection, voice, classification)
Renesas	Ethos-U55 (Arm-licensed but vendor-integrated)	MicroNPU for CNN/RNN inference (up to 256 GOPs in RA8P1)	Cortex-M85 + M33 (RA8 series)	High-performance edge AI (vision, speech); often paired with Helium
Geehy / GD32	Motor Control Hardware Accelerators	Dedicated blocks for dual-motor + PFC control	Cortex-M33 (GD32M531, Geehy G32R series)	Precise multi-motor control in appliances & industrial drives
Infineon	Limited dedicated math/crypto	Relies more on core features + analog; some crypto in XMC/AURIX	Cortex-M0/M4 (XMC series)	Industrial power/motor; fewer pure math accelerators vs. ST/NXP

Vendor-Specific Memory & Real-Time Accelerators

Vendor	Extension / Feature	Description / Use Case	Supported Cores (Typical)	Key Benefits & Applications
STM	ART Accelerator	Adaptive Real-Time prefetch/cache for Flash execution	Cortex-M4/M7/M33 (many STM32 series)	Zero-wait-state Flash performance; general embedded speed-up
STM	CCM-SRAM (Routine Booster)	Static cache-like SRAM for critical code	Cortex-M4 (G4, some H7)	Deterministic real-time loops; motor control, safety

Summary of what have we covered so far

1) What is ISA
2) Evolution of ISA
3) ISA Vs Micro architecture
4) CISC, RISC, RISC-V
5) ISA evolution of ARM
6) ARM ISA Extensions
7) ARM Accelerators and Vendor specific Accelerators

Summary ISA, ISA Extension and Accelerators

#	Takeaway	One Line Proof
1	ISA is the contract between software and hardware	Same ARMv7E-M binary runs on Cortex-M4 from ST, NXP, Infineon, Microchip — without recompilation
2	RISC won in embedded for three decisive reasons	Power efficiency, real-time determinism, and memory became cheap — eliminating CISC's original advantage
3	RISC-V is the open challenger worth watching	Zero licence fees, modular extensions, ESP32-C3 already shipping millions — ecosystem growing rapidly
4	Thumb-2 is genuinely better than both ARM and Thumb	26% smaller than ARM code, 98% of ARM performance — no mode switching, no compromise
5	DSP extensions and FPU are non-negotiable for motor control	Clarke-Park transform — 200 cycles without FPU, 12 cycles with FPU — 16× difference
6	Vendor accelerators are where real application differentiation lives	ST CORDIC, NXP PowerQuad, Nordic PPI — none visible in CoreMark — all visible in your application benchmark
7	ISA determines compatibility — microarchitecture determines performance	M3 and M7 run identical binary — M7 executes it 3 to 4× faster — entirely microarchitectural

Libraries for ARM Cortex M

Cortex Microcontroller Software Interface Standard (CMSIS)

What is CMSIS?

- **CMSIS is a vendor-independent hardware abstraction layer (HAL) for Arm Cortex-M processor-based microcontrollers and some entry-level Cortex-A processors. It is designed to provide consistent, simple software interfaces to the processor and its peripherals, which enables:**
- **Software reuse:** Code can easily be ported across different Cortex-M microcontrollers as CMSIS is not MCU vendor specific.
- **Standardization:** Provides a common approach to interface with peripherals, real-time operating systems (RTOS), and middleware.
- If CMSIS was not supported by ARM, the ARM Cortex-M ecosystem would be severely fractured, characterized by low software reuse, high development costs, and slow time-to-market for new devices.

Core Arm Libraries (CMSIS Family) #1

Library	Purpose	ISA / Architecture Connection	Connection to Extensions & Accelerators	Typical Use Cases
CMSIS-Core (Cortex-M)	Basic runtime, startup code, register access (NVIC, SysTick, MPU, FPU, etc.), intrinsics	Supports **all** Cortex-M: v6-M (M0/M0+), v7-M / v7E-M (M3/M4/M7), v8-M (M23/M33), v8.1-M (M52/M55/M85)	Provides intrinsics for **DSP SIMD**, **FPU**, **TrustZone**, **Helium** (MVE) when present. Automatic detection in many cases.	All projects; device initialization, interrupt handling, core feature access.
CMSIS-RTOS2	Standard RTOS API (with reference RTX implementation)	All Cortex-M; works with TrustZone on v8-M	Indirect: benefits from MPU (task isolation) and TrustZone (secure/non-secure partitioning)	Real-time applications with multiple tasks.
CMSIS-Driver	Standardized peripheral driver interfaces (SPI, I2C, UART, Ethernet, etc.)	All Cortex-M	Vendor implements the actual drivers; library provides consistent API	Middleware and portable peripheral code.

Core Arm Libraries (CMSIS Family) #2

Library	Purpose	ISA / Architecture Connection	Connection to Extensions & Accelerators	Typical Use Cases
CMSIS-DSP	Over 60+ DSP functions (filters, FFT, matrix, basic math, complex math, etc.) for fixed-point (q7/q15/q31) and floating-point	Works on all cores; heavily optimized for cores with **DSP extension** (v7E-M and v8-M Mainline) and **Helium**	- Uses **DSP SIMD** on M4/M7/M33 - Automatic **Helium (MVE)** vectorization on M52/M55/M85 (biggest uplift for vector ops) - Falls back to scalar on v6-M / Baseline	Audio filtering, motor control, sensor fusion, signal processing.
CMSIS-NN	Efficient neural network kernels (convolutions, fully connected, pooling, activation, etc.)	All Cortex-M; best on cores with **DSP** or **Helium**	- Optimized for **DSP SIMD** - Major uplift with **Helium** (MVE) for int8/int16 matrix ops (up to 4–15× vs scalar) - Fallback path when using **Ethos-U NPU** for unsupported operators	TinyML / edge AI inference (keyword spotting, anomaly detection, image classification).

Machine Learning / AI Specific Stack (Often Used with CMSIS)

- **TensorFlow Lite for Microcontrollers (TFLu / TFLM)**: Official Google framework for tinyML. It integrates **CMSIS-NN** as a backend for optimized kernels on Cortex-M. When **Ethos-U** is present, TFLu uses the **Ethos-U driver** for supported operators and falls back to **CMSIS-NN** (often Helium-accelerated) for others.
- **Arm Vela Compiler**: Not a runtime library but a critical tool. It takes a TFLite model and optimizes/compiles it for **Ethos-U NPUs** (U55/U65/U85). The output runs on the NPU with Cortex-M as host.
- **Arm ML Embedded Evaluation Kit (MLEK)**: Example pack with CMSIS-based ML applications, supporting both CPU-only (CMSIS-NN + Helium) and Ethos-U configurations.
- **How it connects**: Cortex-M (with/without Helium) + **CMSIS-NN** handles control + fallback + lightweight inference. **Ethos-U NPU** (dedicated accelerator) offloads heavy CNN/RNN layers for much higher throughput (30–480× uplift on supported ops). The driver + Vela + CMSIS-NN form a hybrid stack.

Vendor specific Libraries

Vendor-Specific Libraries & HALs - Examples

Vendors provide their own **Hardware Abstraction Layer (HAL)** and drivers that sit on top of CMSIS-Core.

- **STMicroelectronics (STM32Cube)**: STM32 HAL + LL (Low Layer) drivers. Dedicated functions/examples for **CORDIC**, **FMAC** (Filter Math Accelerator), **HSP**, and **ART Accelerator.** Often includes CMSIS-DSP/NN integration examples.
- **NXP (MCUXpresso SDK)**: Comprehensive SDK with drivers for **PowerQuad** (DSP/math coprocessor) and **CASPER** (crypto). Provides CMSIS-compatible wrappers and examples using PowerQuad for matrix/FFT/CORDIC-like operations.
- **Texas Instruments**: MSPM0 / AM series SDK with support for **TinyEngine NPU** and trigonometric accelerator. Includes CMSIS-DSP integration.
- **Renesas (Flexible Software Package - FSP)**: Drivers and examples for RA series, including **Ethos-U55** integration and Helium-optimized code.
- **Other vendors** (Infineon, Microchip, etc.): Their HALs build on CMSIS-Core and add peripheral-specific drivers. They rarely add heavy math accelerators.

How to Choose & Build Libraries

How to Choose & Build

- Start with **CMSIS-Core** + vendor HAL for any project.
- Add **CMSIS-DSP** for signal processing and **CMSIS-NN** for ML.
- For heavy AI: Use **TFLu + CMSIS-NN** (CPU/Helium) or **TFLu + Vela + Ethos-U driver** (NPU offload).
- **Compiler flags matter:** Enable -O3 -ffast-math, target the correct core (-mcpu=cortex-m55), and define macros like ARM_MATH_HELIUM for best optimization.
- **Tools:** Keil MDK, Arm GNU Toolchain, IAR, or vendor IDEs (STM32CubeIDE, MCUXpresso, etc.).

Summary

1. Foundations: ISA vs. Microarchitecture, RISC vs. CISC: * RISC (ARM, RISC-V):
2. The Evolution of ARM Execution: ARM State, Thumb, Thumb-2
3. Extending the Core: Extensions (ARM Standard) vs. Accelerators (ARM, Vendor Proprietary)
4. The Software Bridge: CMSIS & HAL