Cortex-M4 Processor Overview

with ARM Processors and Architectures

Introduction

ARM

- **ARM was developed at Acorn Computer Limited of Cambridge, UK (between 1983 & 1985)**
  - RISC concept introduced in 1980 at Stanford and Berkeley

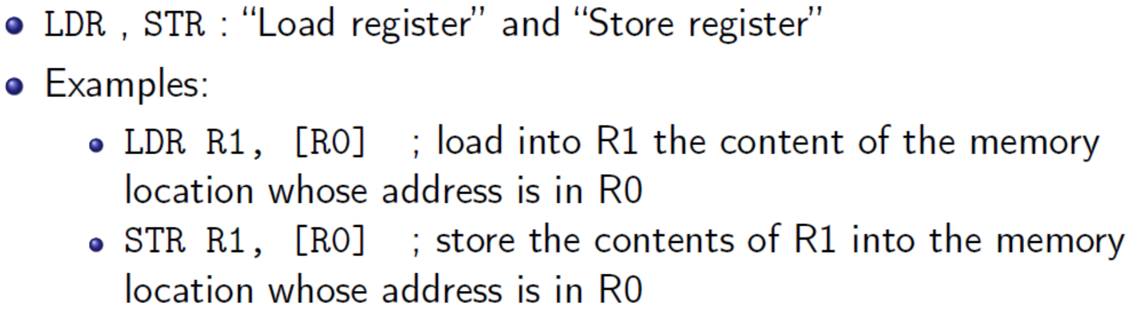
- **ARM founded in November 1990**
  - Advanced RISC Machines

- **Best known for its range of RISC processor cores designs**
  - Other products – fabric IP, software tools, models, cell libraries - to help partners develop and ship ARM-based SoCs

- **ARM does not manufacture silicon**
  - Licensed to partners to develop and fabricate new micro-controllers
    - Soft-core

ARM Architecture

- **Based upon RISC Architecture with enhancements to meet requirements of embedded applications**
  - A large uniform register file
  - Load-store architecture

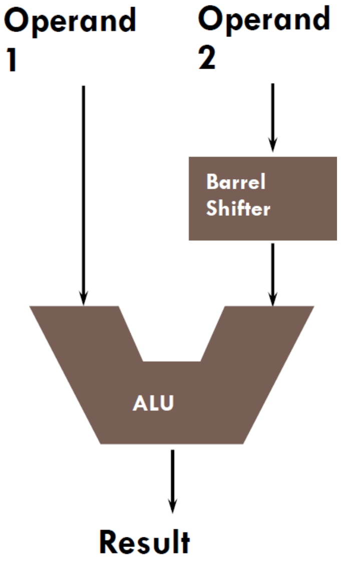


- `LDR` , `STR` : “Load register” and “Store register”
- Examples:
  - `LDR R1, [R0]` ; load into R1 the content of the memory location whose address is in R0
  - `STR R1, [R0]` ; store the contents of R1 into the memory location whose address is in R0

  - Fixed length instructions
  - 32-bit processor (v1-v7), 64-bit processor (v8)
  - Good speed/power
  - High code density

Enhancement to Basic RISC

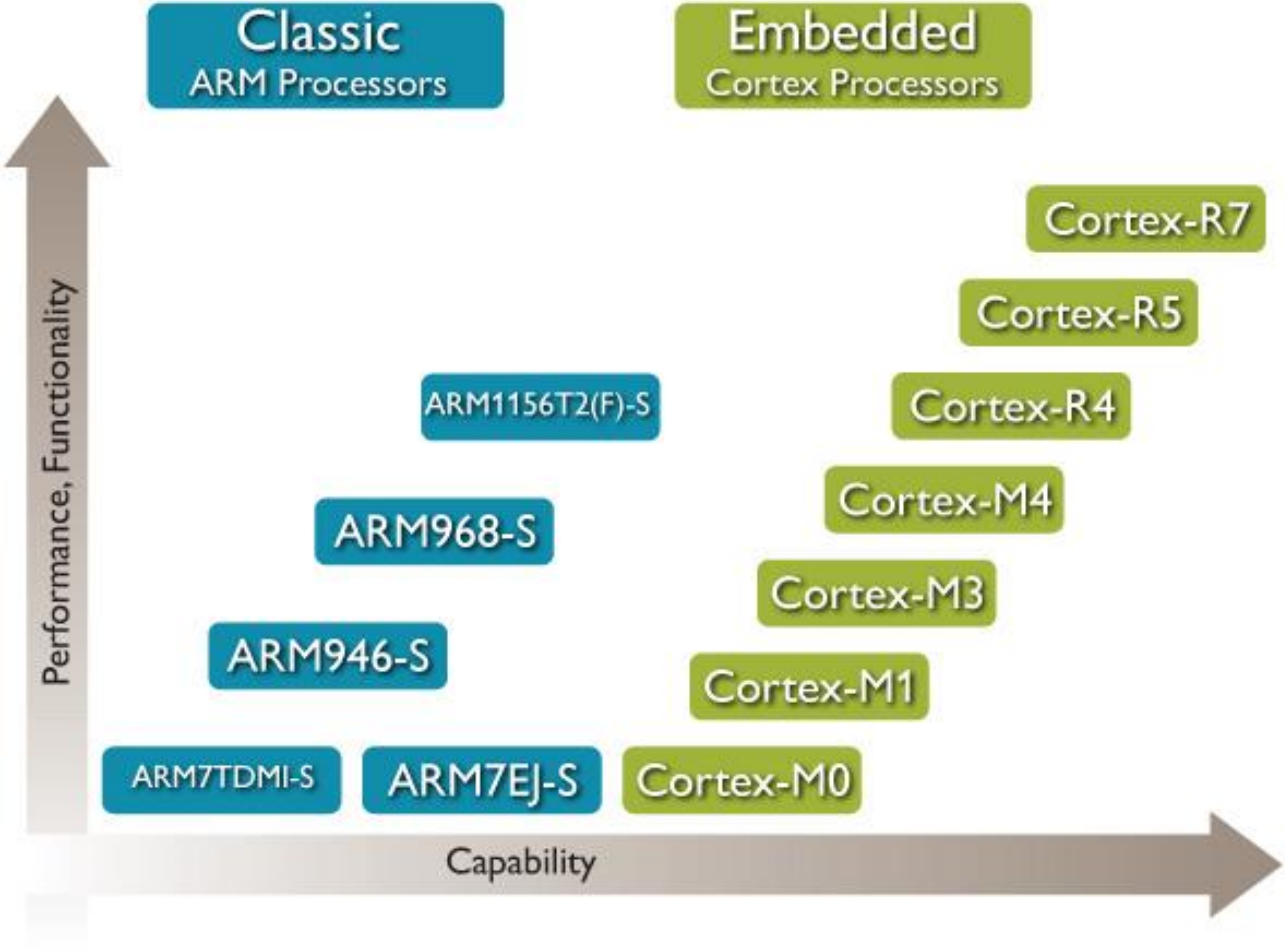
- **Control over both ALU and shifter for every data processing operations**
  - ADD r2, r3, r4, LSL #2 ; r2 = r3 + (r4 * 4)



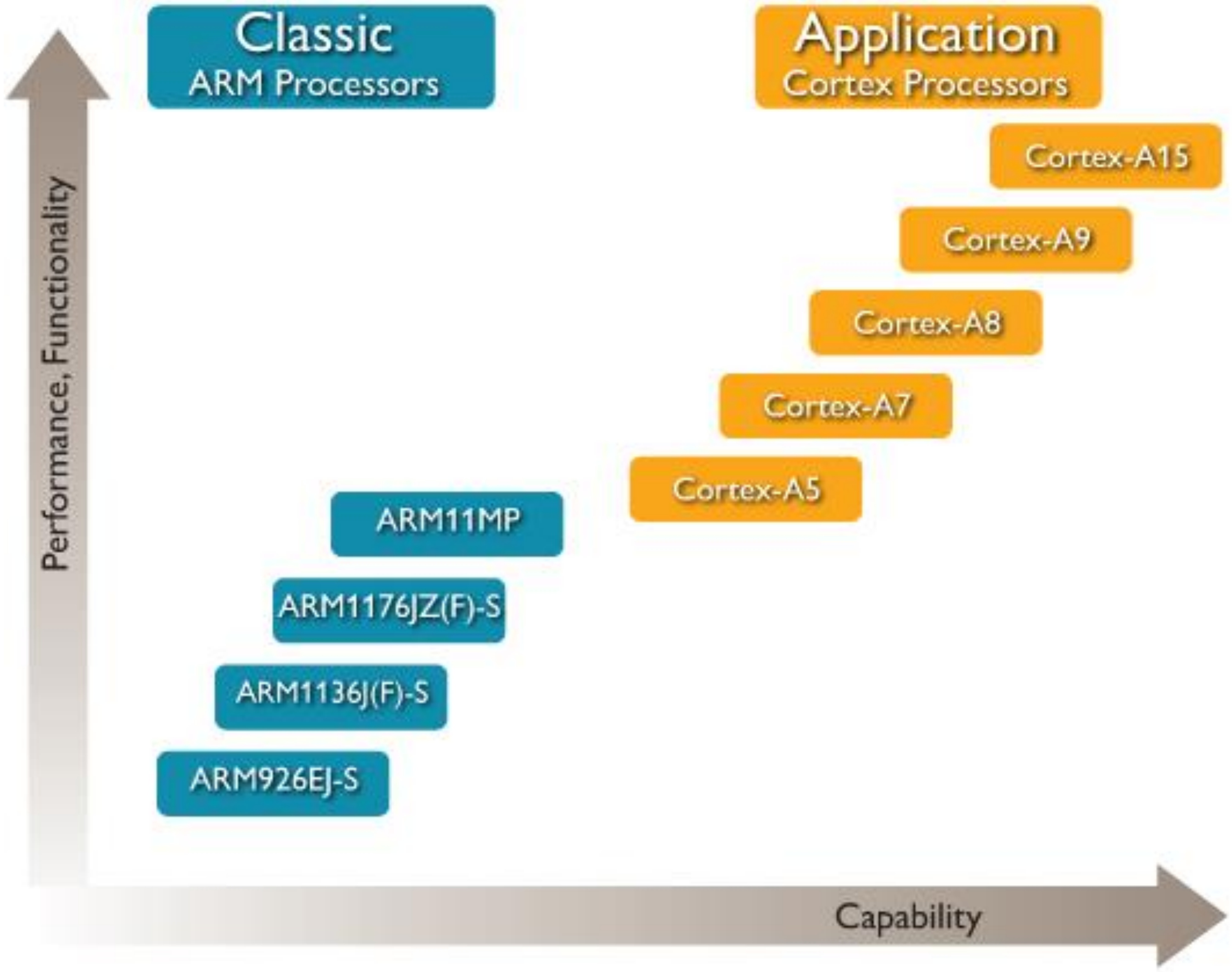
- **Auto-increment and auto-decrement addressing modes**
  - To optimize program loops
- **Load/Store multiple data instructions**
  - To maximize data throughput
  - LDM, STM
- **Conditional execution of instructions**
  - To maximize execution throughput
    - Example: ADDEQ r0, r1, r2

      → Instruction will only be executed when the zero flag is set to 1

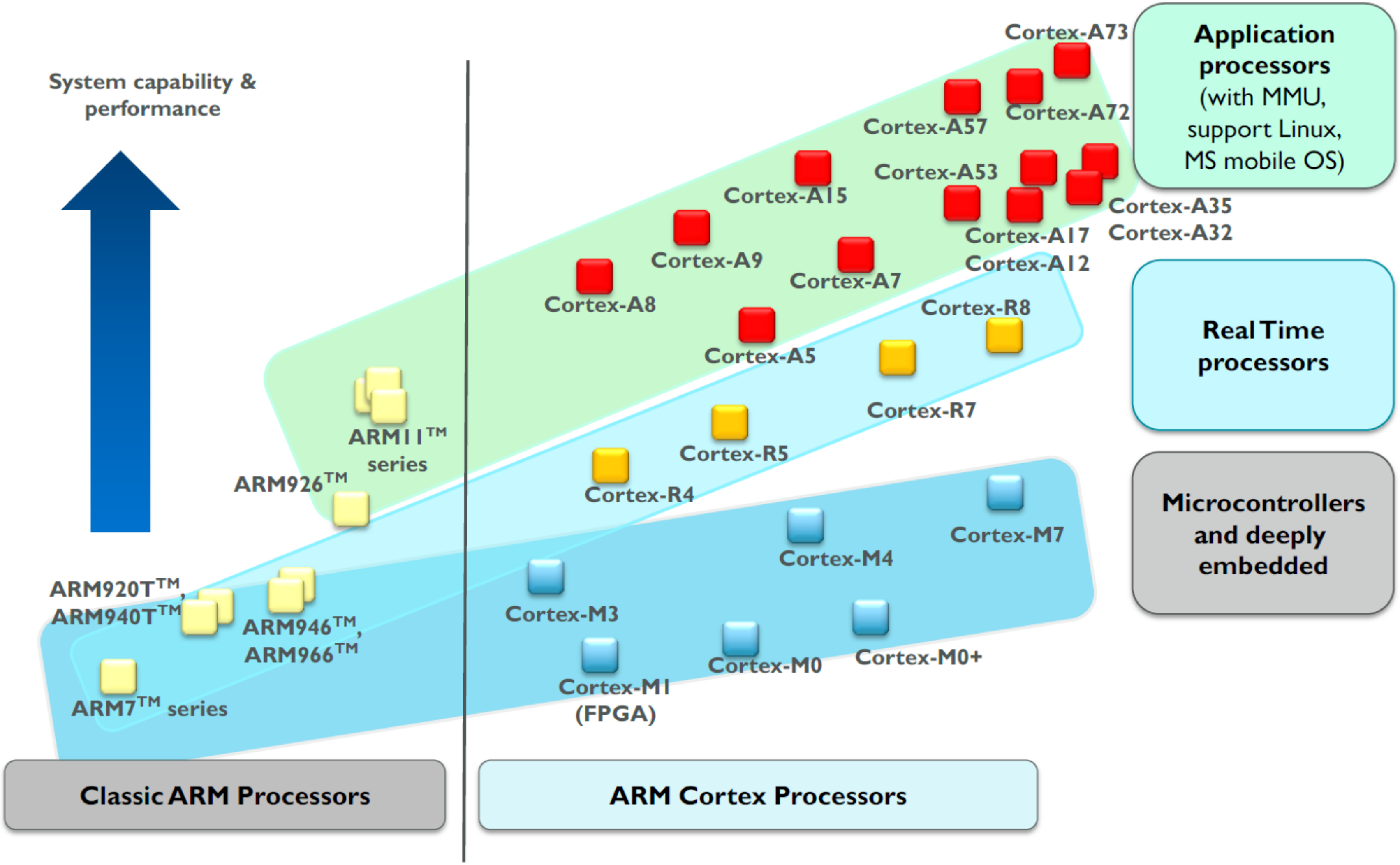
Embedded Processors



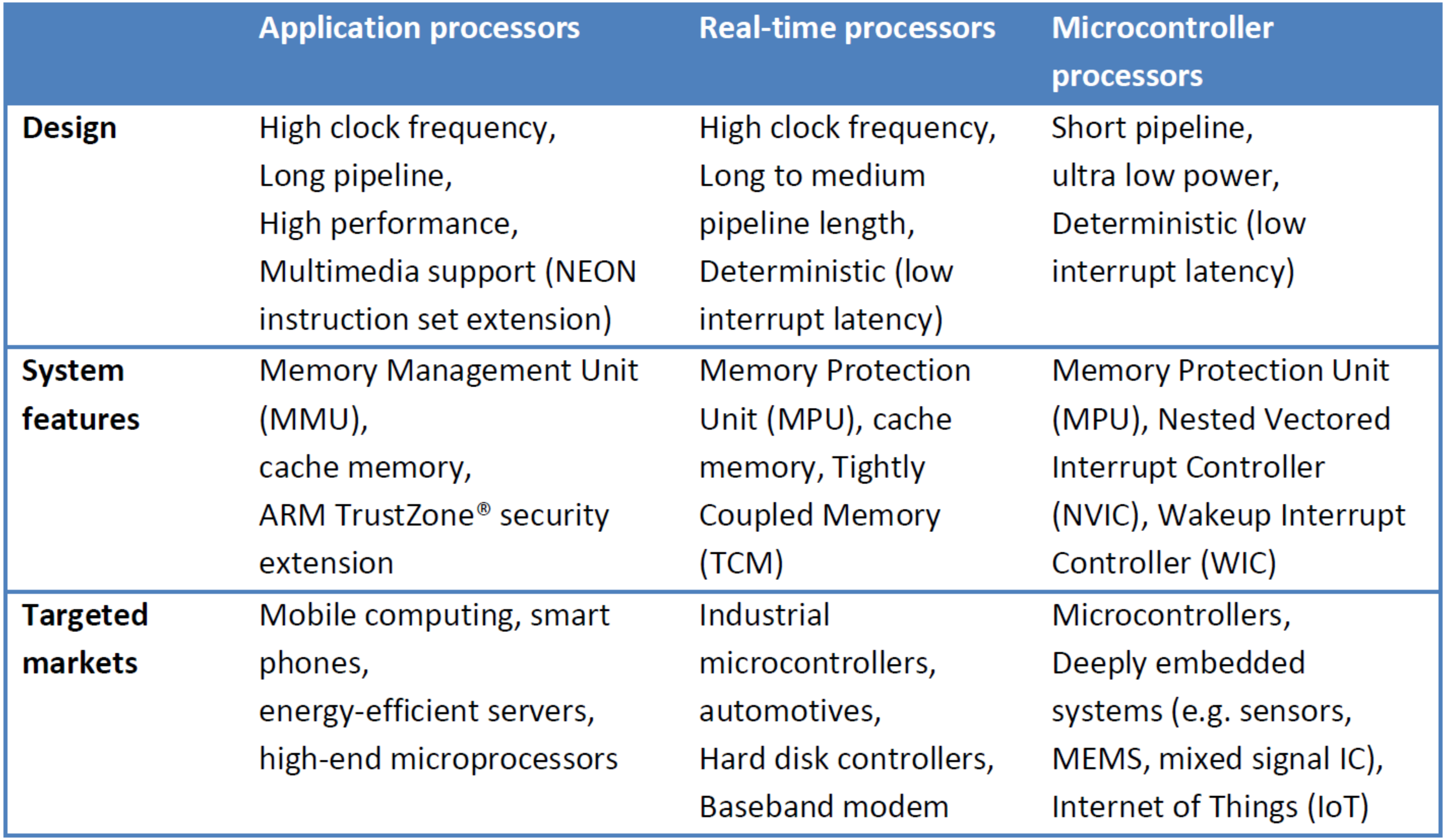
Application Processors



ARM Processor Family



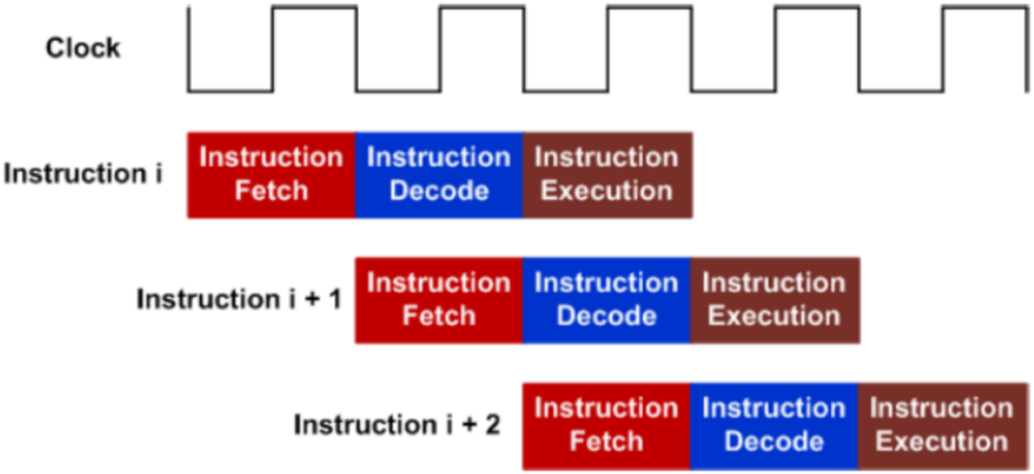
Summary of Processor Characteristics



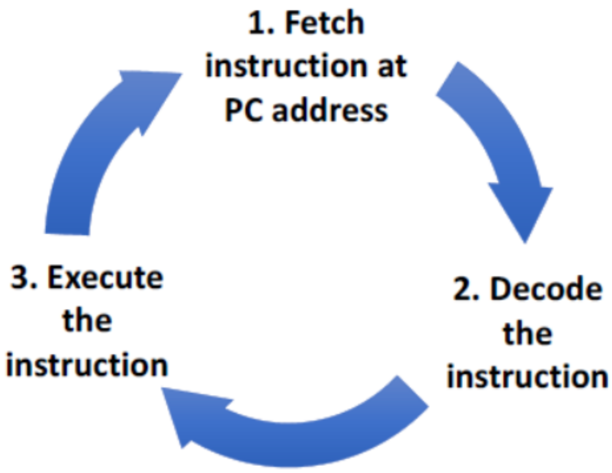
	Application processors	Real-time processors	Microcontroller processors
Design	High clock frequency, Long pipeline, High performance, Multimedia support (NEON instruction set extension)	High clock frequency, Long to medium pipeline length, Deterministic (low interrupt latency)	Short pipeline, ultra low power, Deterministic (low interrupt latency)
System features	Memory Management Unit (MMU), cache memory, ARM TrustZone® security extension	Memory Protection Unit (MPU), cache memory, Tightly Coupled Memory (TCM)	Memory Protection Unit (MPU), Nested Vectored Interrupt Controller (NVIC), Wakeup Interrupt Controller (WIC)
Targeted markets	Mobile computing, smart phones, energy-efficient servers, high-end microprocessors	Industrial microcontrollers, automotives, Hard disk controllers, Baseband modem	Microcontrollers, Deeply embedded systems (e.g. sensors, MEMS, mixed signal IC), Internet of Things (IoT)

Pipeline

- Pipelining allows hardware resources to be fully utilized
- One 32-bit instruction or **two 16-bit** instructions can be fetched.



Pipeline of 32-bit instructions


1. Fetch instruction at PC address
2. Decode the instruction
3. Execute the instruction

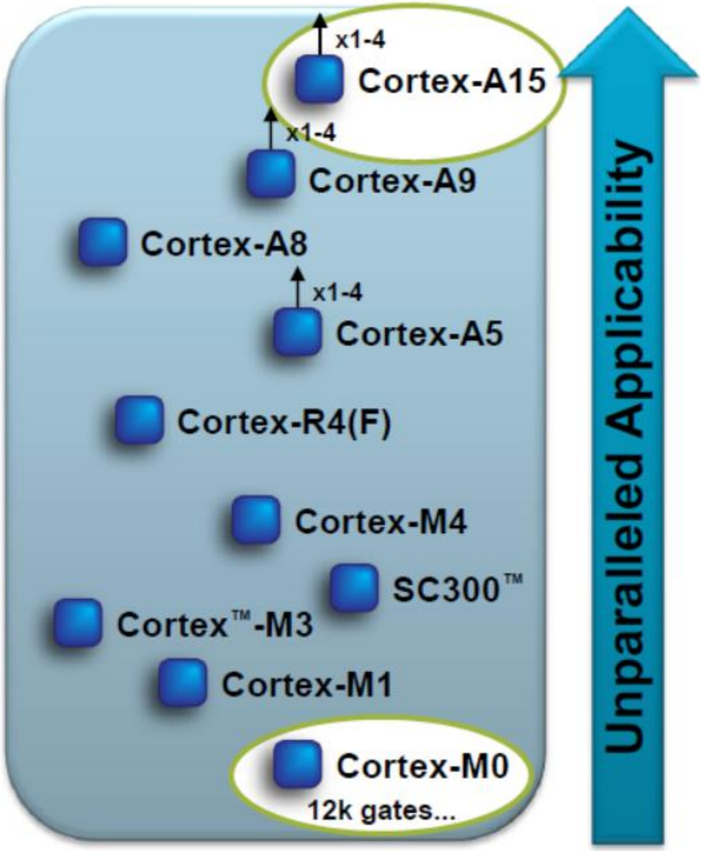
ARM Cortex Advanced Processors

Architectural innovation, compatibility across diverse application spectrum

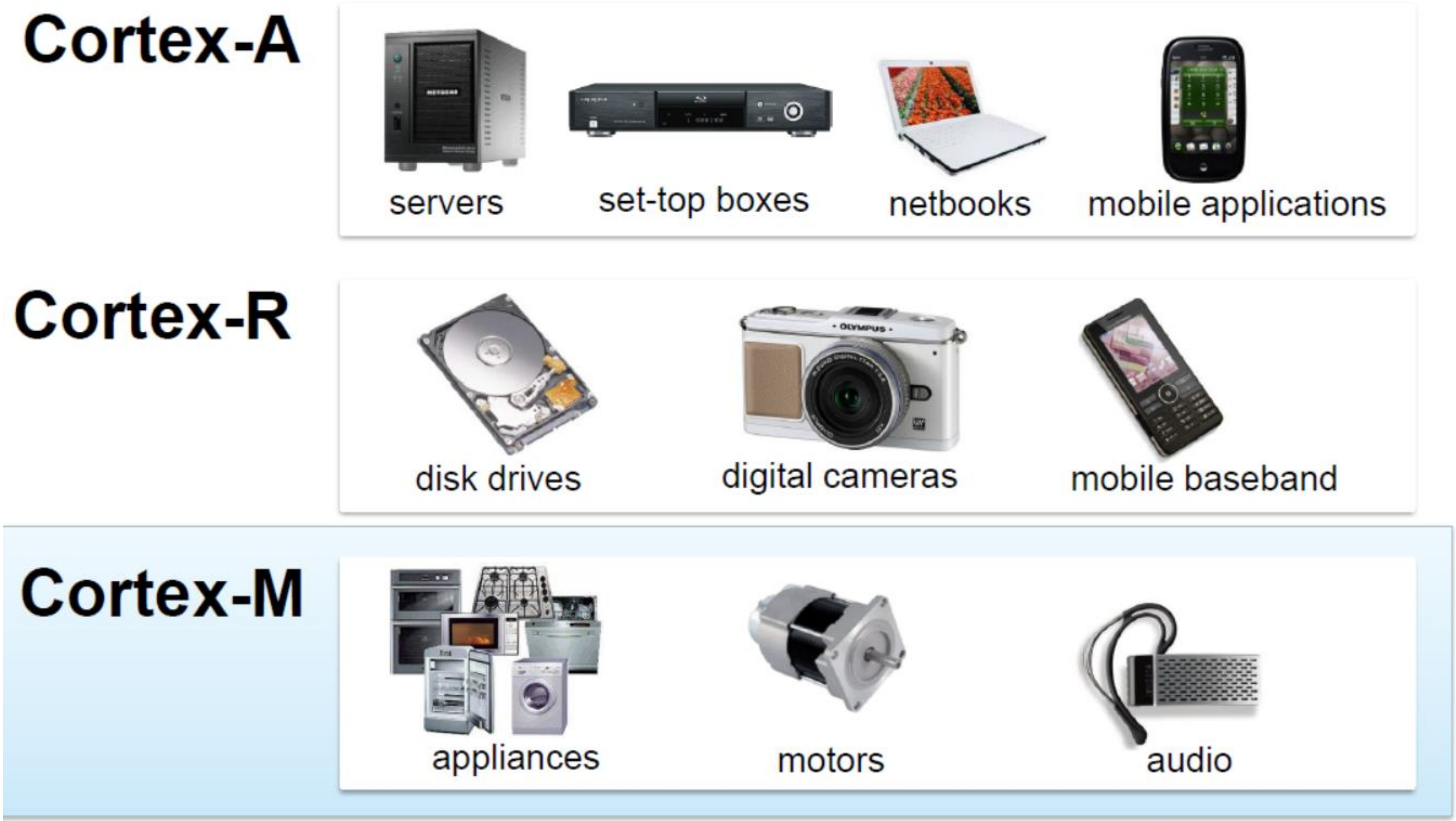
Cortex™

Low-Power Leadership from ARM®

- ARM Cortex-A family:
  - Applications processors for feature-rich OS and 3rd party applications
- ARM Cortex-R family:
  - Embedded processors for real-time signal processing, control applications
- ARM Cortex-M family:
  - Microcontroller-oriented processors for MCU, ASSP, and SoC applications



Application Examples



Cortex-A

servers

set-top boxes

netbooks

mobile applications

Cortex-R

disk drives

digital cameras

mobile baseband

Cortex-M

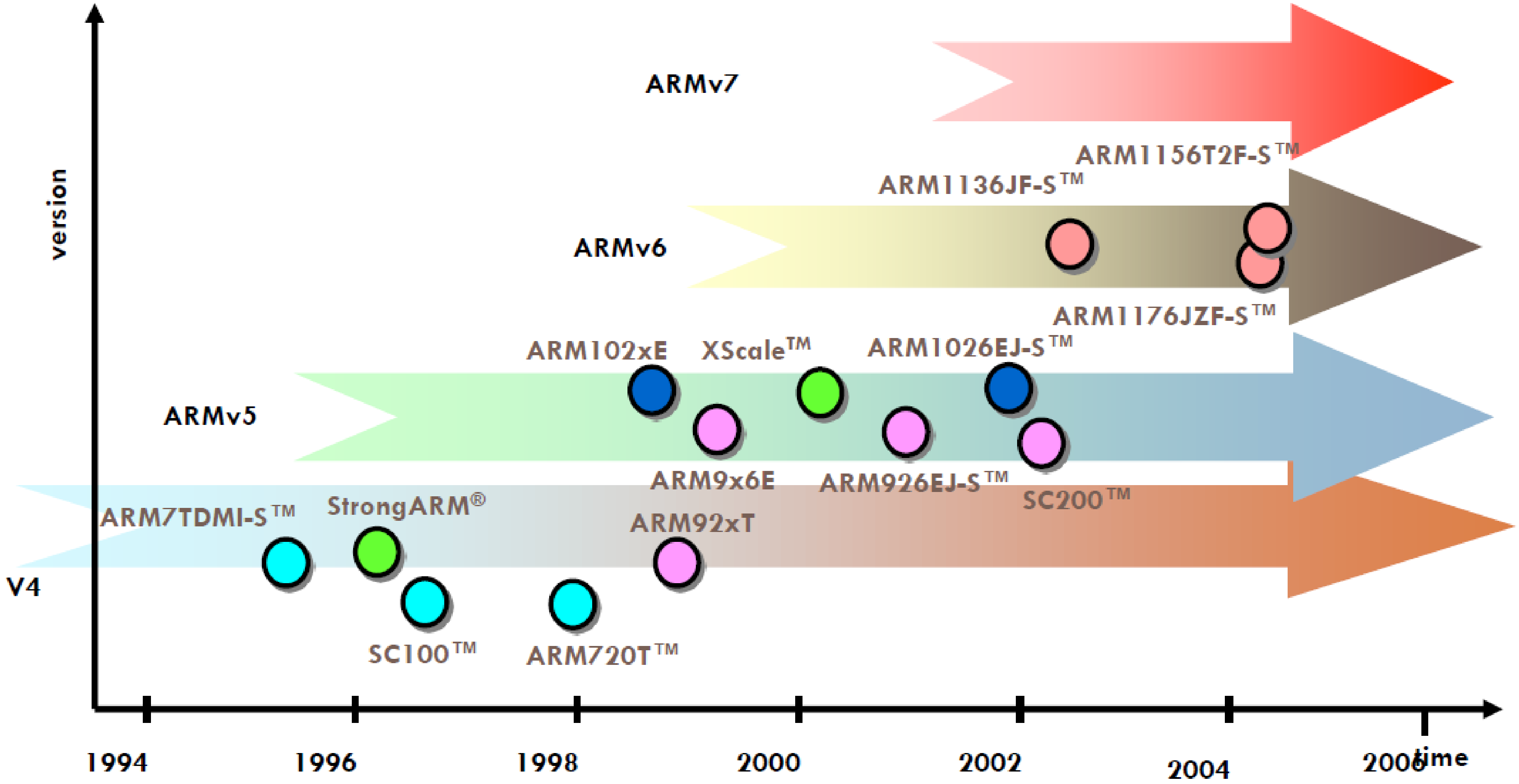
appliances

motors

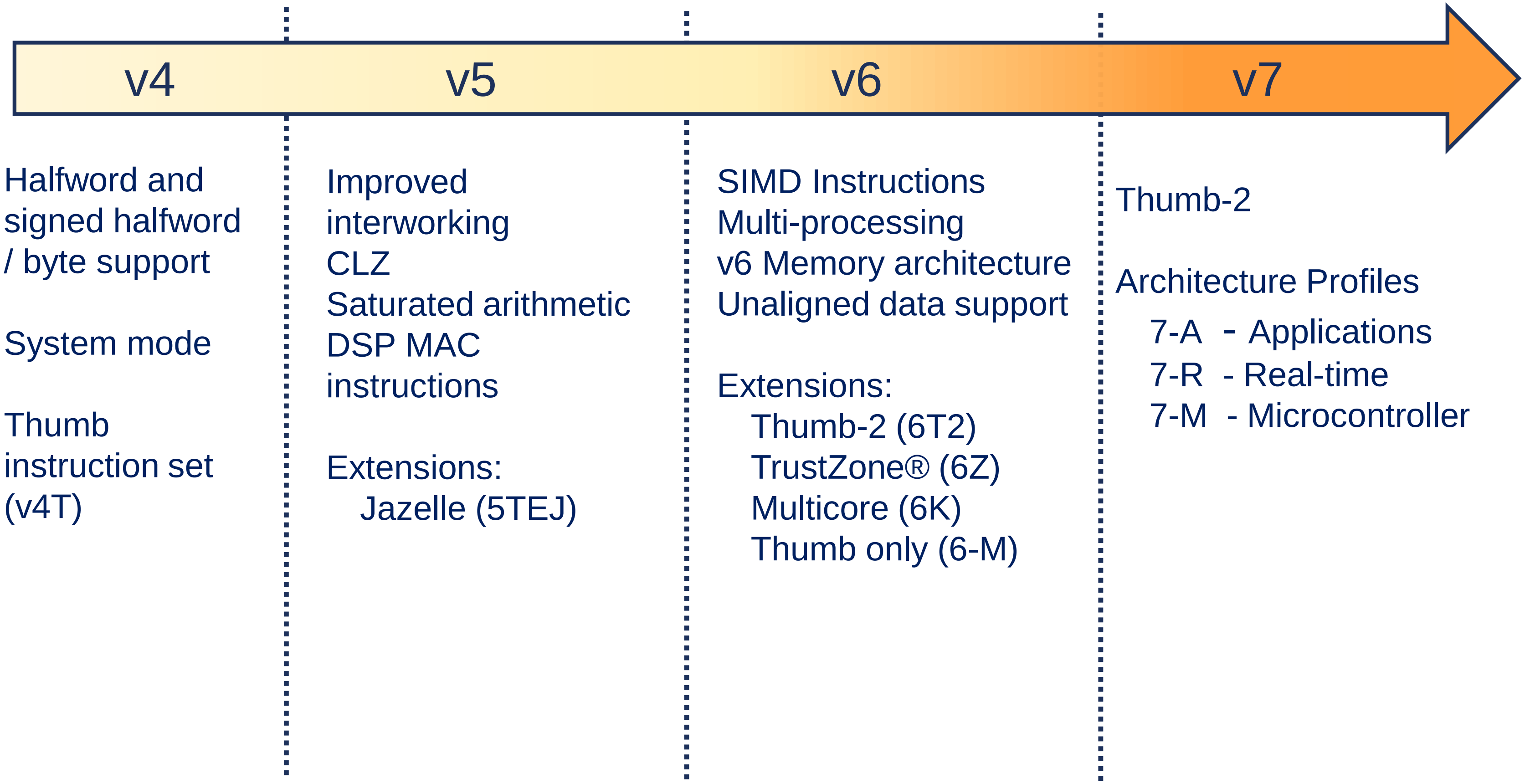
audio

ARM Architecture Overview

Architecture History



Development of the ARM Architecture



v4

Halfword and signed halfword / byte support

System mode

Thumb instruction set (v4T)

v5

Improved interworking
CLZ
Saturated arithmetic
DSP MAC instructions

Extensions:
Jazelle (5TEJ)

v6

SIMD Instructions
Multi-processing
v6 Memory architecture
Unaligned data support

Extensions:
Thumb-2 (6T2)
TrustZone® (6Z)
Multicore (6K)
Thumb only (6-M)

v7

Thumb-2

Architecture Profiles
7-A - Applications
7-R - Real-time
7-M - Microcontroller

- **Note that implementations of the same architecture can be different**
  - Cortex-A8 - architecture v7-A, with a 13-stage pipeline
  - Cortex-A9 - architecture v7-A, with an 8-stage pipeline

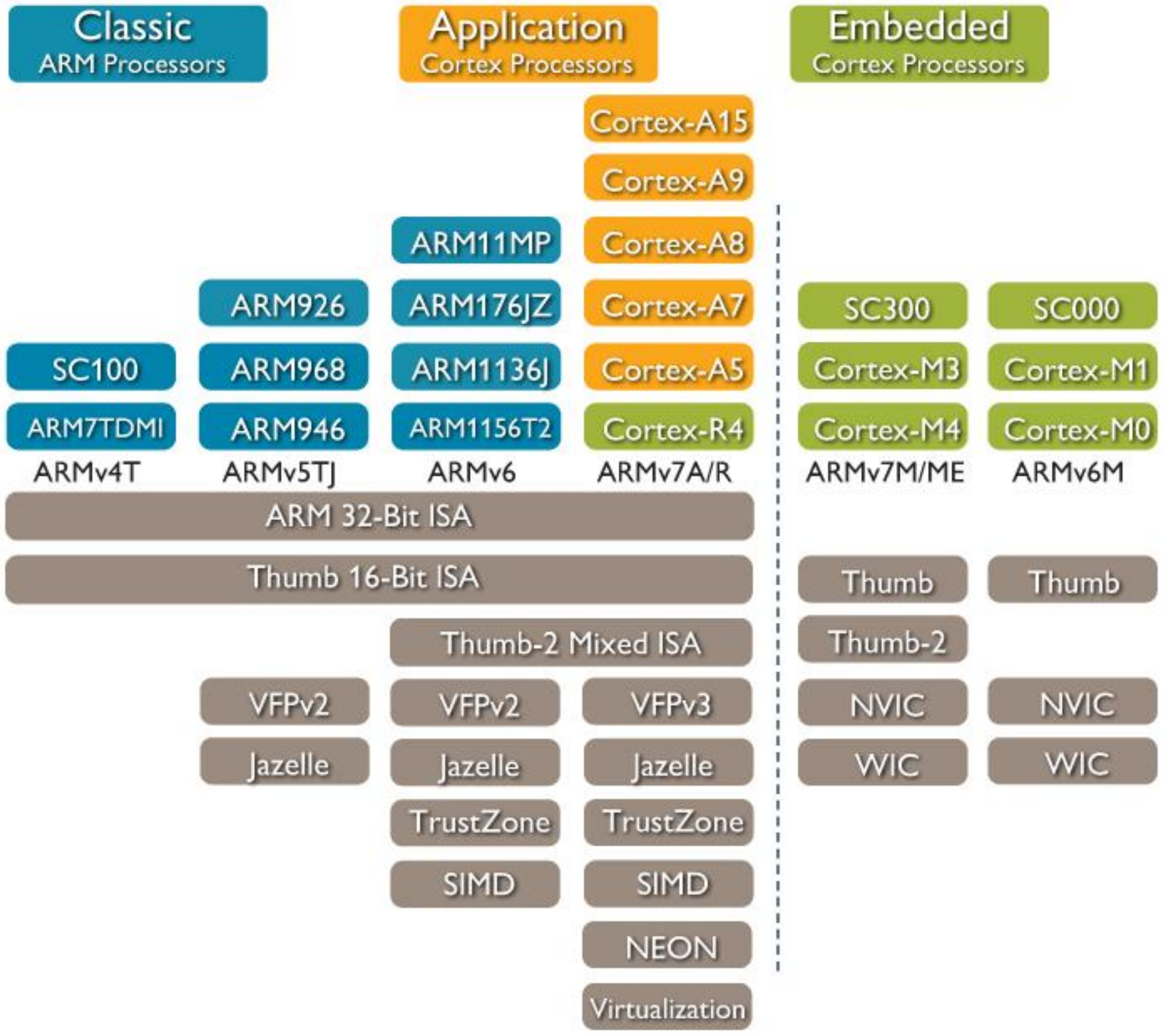
Architecture ARMv7 profiles

- **Application profile (ARMv7-A)**
  - Memory management support (MMU)
  - Highest performance at low power
    - Influenced by multi-tasking OS system requirements
  - TrustZone and Jazelle-RCT for a safe, extensible system
  - e.g. Cortex-A5, Cortex-A9

- **Real-time profile (ARMv7-R)**
  - Protected memory (MPU)
  - Low latency and predictability ‘real-time’ needs
  - Evolutionary path for traditional embedded business
  - e.g. Cortex-R4

- **Microcontroller profile (ARMv7-M, ARMv7E-M, ARMv6-M)**
  - Lowest gate count entry point
  - Deterministic and predictable behavior a key priority
  - Deeply embedded use
  - e.g. Cortex-M3

Which architecture is my processor?



Classic
ARM Processors

Application
Cortex Processors

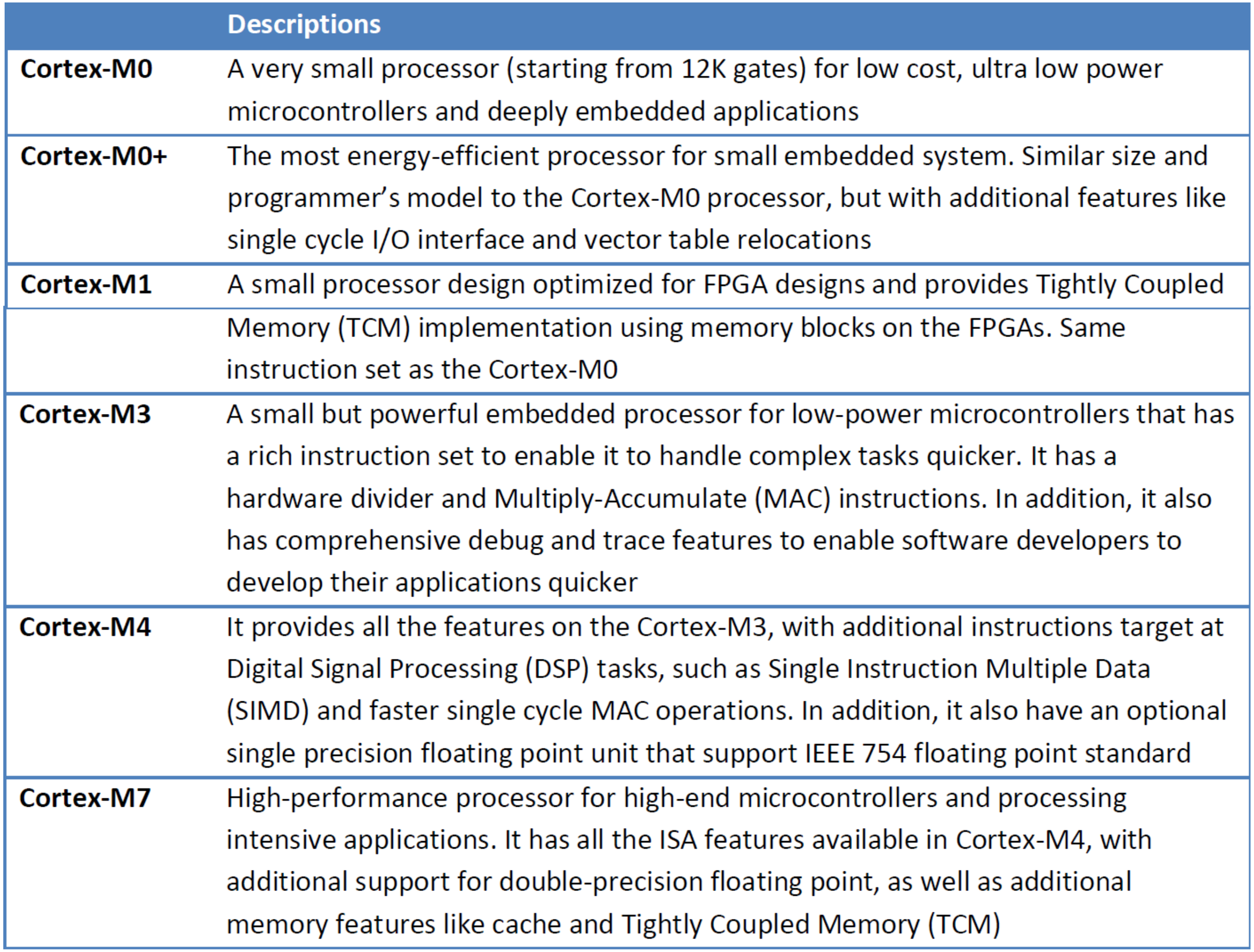
Embedded
Cortex Processors

Cortex-A15
Cortex-A9
ARM11MP
Cortex-A8
ARM926
ARM176JZ
Cortex-A7
SC300
SC000
SC100
ARM968
ARM1136J
Cortex-A5
Cortex-M3
Cortex-M1
ARM7TDMI
ARM946
ARM1156T2
Cortex-R4
Cortex-M4
Cortex-M0

ARMv4T
ARMv5TJ
ARMv6
ARMv7A/R
ARMv7M/ME
ARMv6M

ARM 32-Bit ISA
Thumb 16-Bit ISA
Thumb
Thumb
Thumb-2 Mixed ISA
Thumb-2
VFPv2
VFPv2
VFPv3
NVIC
NVIC
Jazelle
Jazelle
Jazelle
WIC
WIC
TrustZone
TrustZone
SIMD
SIMD
NEON
Virtualization

Cotex-M Processor Family



	Descriptions
Cortex-M0	A very small processor (starting from 12K gates) for low cost, ultra low power microcontrollers and deeply embedded applications
Cortex-M0+	The most energy-efficient processor for small embedded system. Similar size and programmer’s model to the Cortex-M0 processor, but with additional features like single cycle I/O interface and vector table relocations
Cortex-M1	A small processor design optimized for FPGA designs and provides Tightly Coupled Memory (TCM) implementation using memory blocks on the FPGAs. Same instruction set as the Cortex-M0
Cortex-M3	A small but powerful embedded processor for low-power microcontrollers that has a rich instruction set to enable it to handle complex tasks quicker. It has a hardware divider and Multiply-Accumulate (MAC) instructions. In addition, it also has comprehensive debug and trace features to enable software developers to develop their applications quicker
Cortex-M4	It provides all the features on the Cortex-M3, with additional instructions target at Digital Signal Processing (DSP) tasks, such as Single Instruction Multiple Data (SIMD) and faster single cycle MAC operations. In addition, it also have an optional single precision floating point unit that support IEEE 754 floating point standard
Cortex-M7	High-performance processor for high-end microcontrollers and processing intensive applications. It has all the ISA features available in Cortex-M4, with additional support for double-precision floating point, as well as additional memory features like cache and Tightly Coupled Memory (TCM)

ARMv7-M Architecture

ARMv7-M Profile Overview

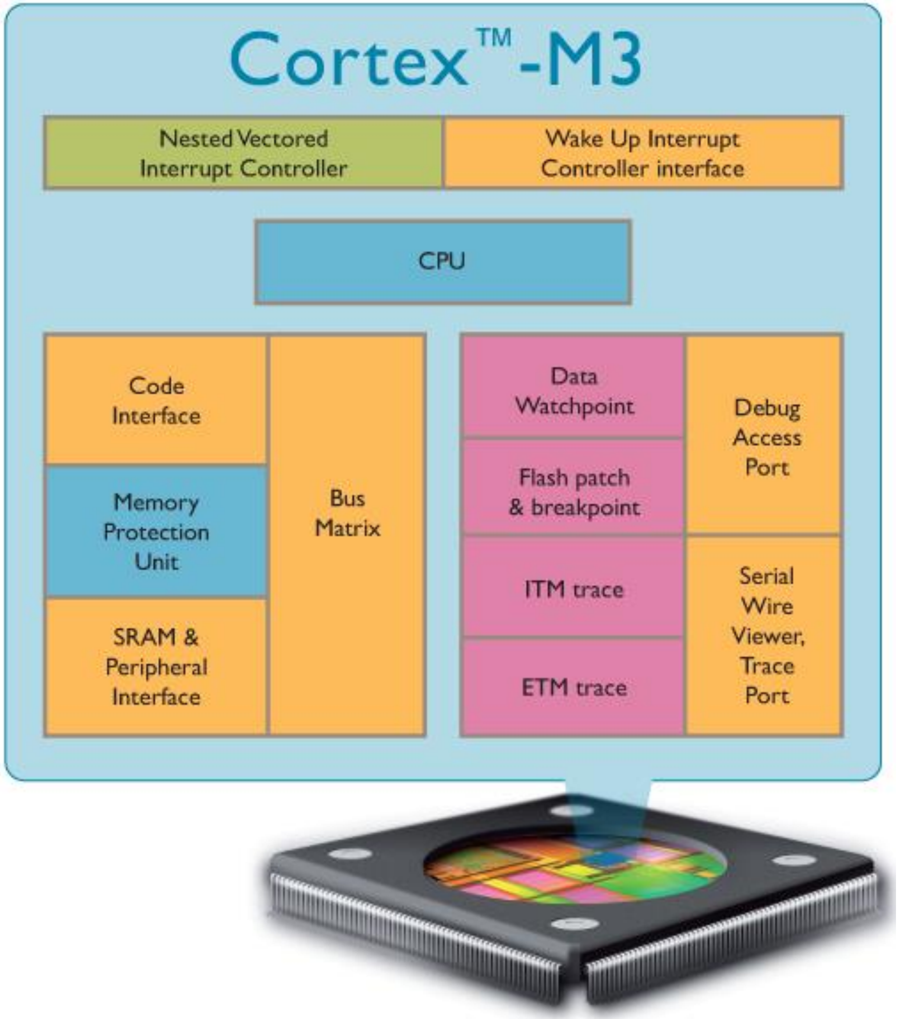
- **v7-M Cores are designed to support the microcontroller market**
  - Simpler to program – entire application can be programmed in C
  - Fewer features needed than in application processors

- **Register and ISA changes from other ARM cores**
  - No ARM instruction set support
  - Only one set of registers
  - xPSR has different bits than CPSR

- **Different modes and exception models**
  - Only two modes: Thread mode and Handler mode
  - Vector table is addresses, not instructions
  - Exceptions automatically save state (r0-r3, r12, lr, xPSR, pc) on the stack

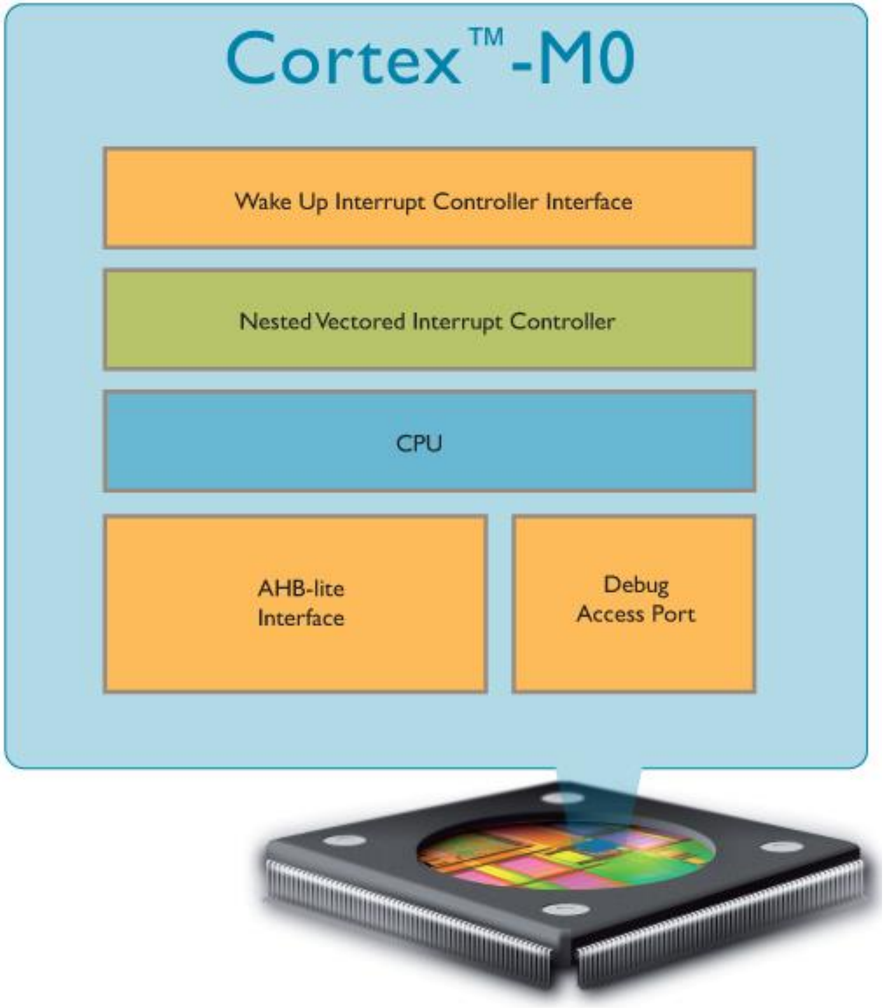
- **Different system control/memory layout**
  - Cores have a fixed memory map
  - No coprocessor 15 – controlled through memory mapped control registers

Cortex-M3



- **ARMv7-M Architecture**
  - Thumb-2 only
- **Fully programmable in C**
- **3-stage pipeline**
- **Optional MPU**
- **AHB-Lite bus interface**
- **Fixed memory map**
- **1-240 interrupts**
  - Configurable priority levels
  - Non-Maskable Interrupt support
  - Debug and Sleep control
- **Serial wire or JTAG debug**
- **Optional ETM**

Cortex-M0


Cortex™-M0
Wake Up Interrupt Controller Interface
Nested Vectored Interrupt Controller
CPU
AHB-lite Interface
Debug Access Port

- **ARMv6-M Architecture**
  - 16-bit Thumb-2 with system control instructions
- **Fully programmable in C**
- **3-stage pipeline**
- **AHB-Lite bus interface**
- **Fixed memory map**
- **1-32 interrupts**
  - Configurable priority levels
  - Non-Maskable Interrupt support
- **Low power support**
- **Core configured with or without debug**
  - Variable number of watchpoints and breakpoints

Thumb-2 Technology

- **Thumb-2 ISA was introduced in ARMv7 architecture**
  - Original16-bit Thumb instructions maintain full compatibility with existing code

+

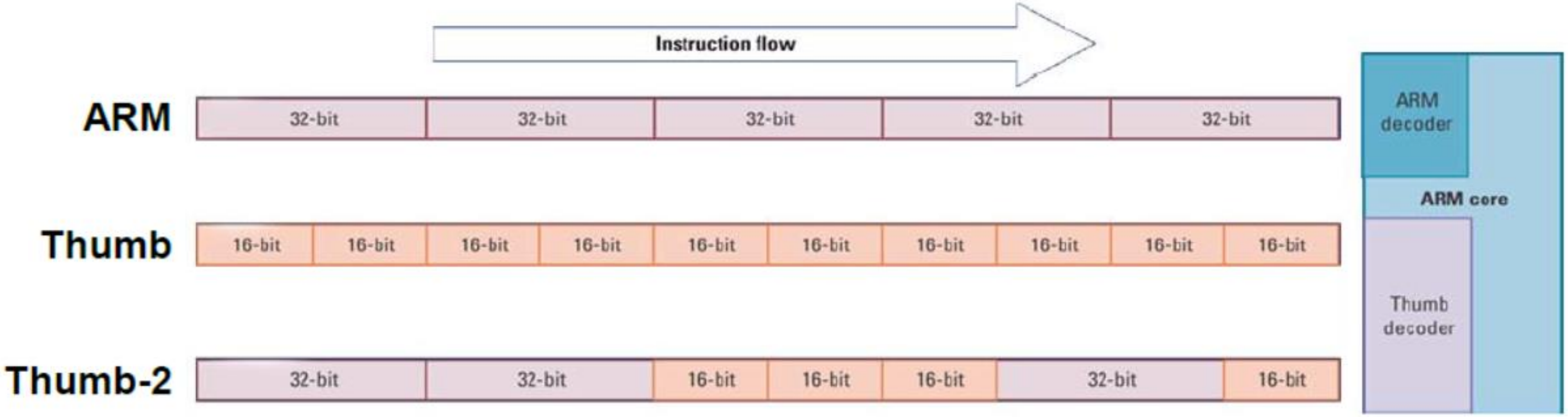
  - New 16-bit Thumb instructions for improved program flow

+

  - New 32-bit Thumb instructions for improved performance and code size.

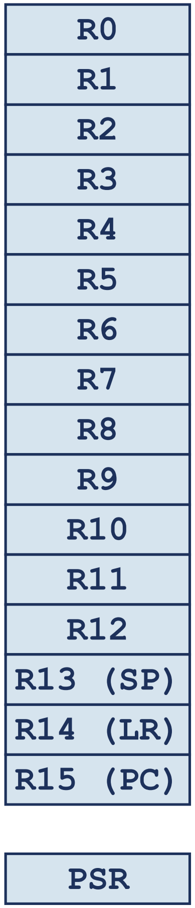
    One 32-bit instruction replaces multiple 16-bit opcodes.

    32-bit instructions are handled in the same mode ~ no interworking required



Programmer’s Model

Processor Register Set



R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13 (SP)
R14 (LR)
R15 (PC)

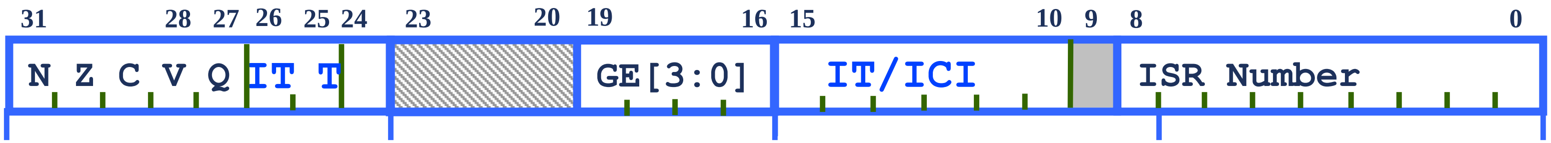
PSR

- **Registers R0-R12**
  - General-purpose registers
- **R13 is the stack pointer (SP) - 2 banked versions**
- **R14 is the link register (LR)**
- **R15 is the program counter (PC)**
- **PSR (Program Status Register)**
  - Not explicitly accessible
  - Saved to the stack on an exception
  - Subsets available as APSR, IPSR, and EPSR

Special Purpose Registers

- **Program Status Register**
  - Described in upcoming slides
- **Special Purpose Mask Registers : PRIMASK, FAULTMASK, BASEPRI**
  - Used to modify exception priorities
  - To set/clear PRIMASK and FAULTMASK, use CPS instructions
    - CPSIE i / CPSID i / CPSIE f / CPSID f
- **Special Purpose CONTROL Register**
  - 2 bits
    - Bit 0 defines Thread mode privilege
    - Bit 1 defines Thread mode stack
- **The Special Purpose Registers are not memory-mapped**
- **Accessed via specific instructions**
  - MRS – Move special purpose register to general-purpose register
  - MSR – Move general-purpose register to special purpose register

xPSR - Program Status Register



- **xPSR stored on stack during exceptions**
- **Condition code flags**
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation carry out
  - V = ALU operation overflow
  - Q = Saturated math overflow
- **IT/ICI bits**
  - Contain IF-THEN base condition code or Interrupt Continue information
- **ISR Number**
  - Stacked xPSR shows which exception was pre-empted
- **T=1**

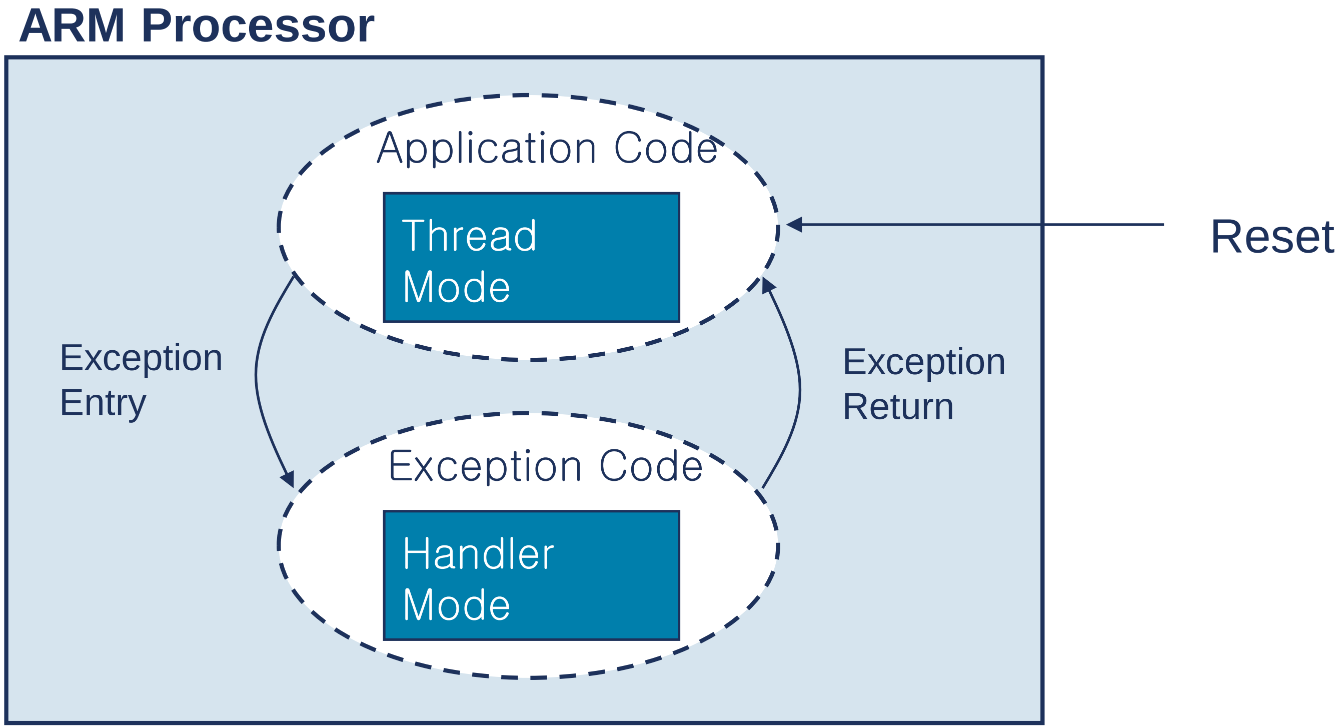
System Timer – SysTick

- **Flexible system timer**
    - 24-bit self-reloading down counter
        - Reload on count == 0
        - Optionally cause SysTick interrupt on count == 0
    - Reload register
    - Calibration value

- **Clock source is CPU clock or optional external timing reference**
    - Software selectable if provided
    - Reference pulse widths High/Low must exceed processor clock period
        - Counted by sampling on processor clock

- **Calibration Register provides value required for 10ms interval**
    - STCALIB inputs tied to appropriate value

Modes Overview



Not shown: Handler mode can also be re-entered on exception return

Processor Mode

- **Handler Mode**
  - Used to handle exceptions. The processor returns to Thread mode when it has finished exception processing.

- **Thread Mode**
  - Used to execute application software. The processor enters Thread mode when it comes out of reset.
  - In Thread mode, the CONTROL register controls whether software execution is privileged or unprivileged, see CONTROL register. In Handler mode, software execution is always privileged.

Privilege Levels

- **Unprivileged**

  The software:

  - 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.
  - Unprivileged software executes at the unprivileged level

- **Privileged**
  - The software can use all the instructions and has access to all resources.

Instruction Set Examples:

- Data Processing:

```
MOV     r2, r5                  ; r2 = r5
ADD     r5, #0x24               ; r5 = r5 + 36
ADD     r2, r3, r4, LSL #2      ; r2 = r3 + (r4 * 4)
LSL     r2, #3                  ; r2 = r2 * 8
MOVT r9, #0x1234                ; upper halfword of r9 = #0x1234
MLA     r0, r1, r2, r3          ; r0 = (r1 * r2) + r3
```

- Memory Access:

```
STRB r2, [r10, r1]              ; store lower byte in r2 at
                                  address {r10 + r1}
LDR     r0, [r1, r2, LSL #2]    ; load r0 with data at address
                                  {r1 + r2 * 4}
```

- Program Flow:

```
BL      <label>                 ; PC relative branch to <label>
                                  location, and return address
                                  stored in LR (r14)
```

Exception Handling

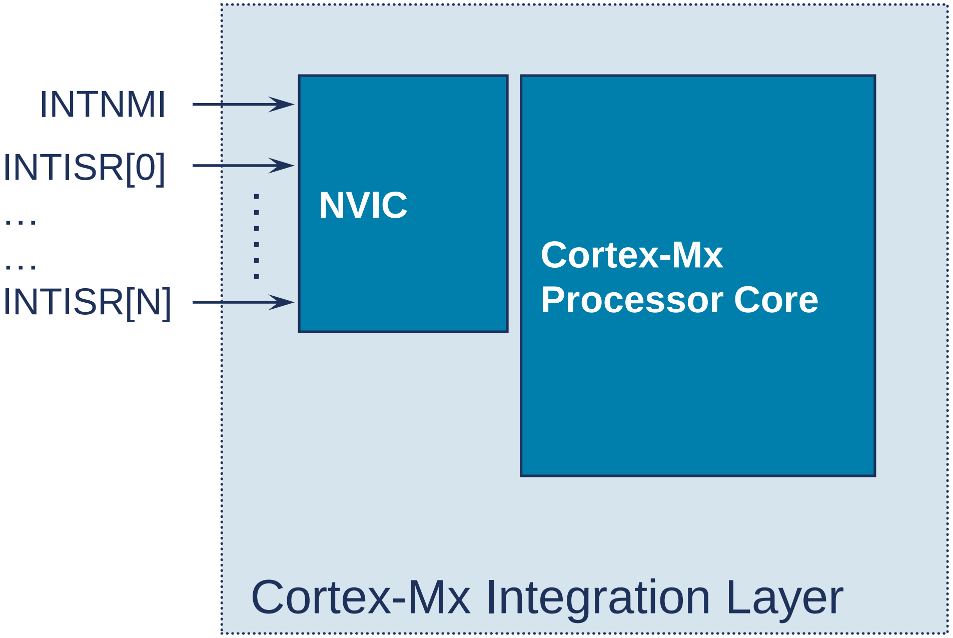
- **Exception types:**
  - Reset
  - Non-maskable Interrupts (NMI)
  - Faults
  - PendSV
  - SVCall
  - External Interrupt
  - SysTick Interrupt

- **Exceptions processed in Handler mode (except Reset)**
  - Exceptions always run privileged

- **Interrupt handling**
  - Interrupts are a sub-class of exception
  - Automatic save and restore of processor registers (xPSR, PC, LR, R12, R3-R0)
  - Allows handler to be written entirely in ‘C’

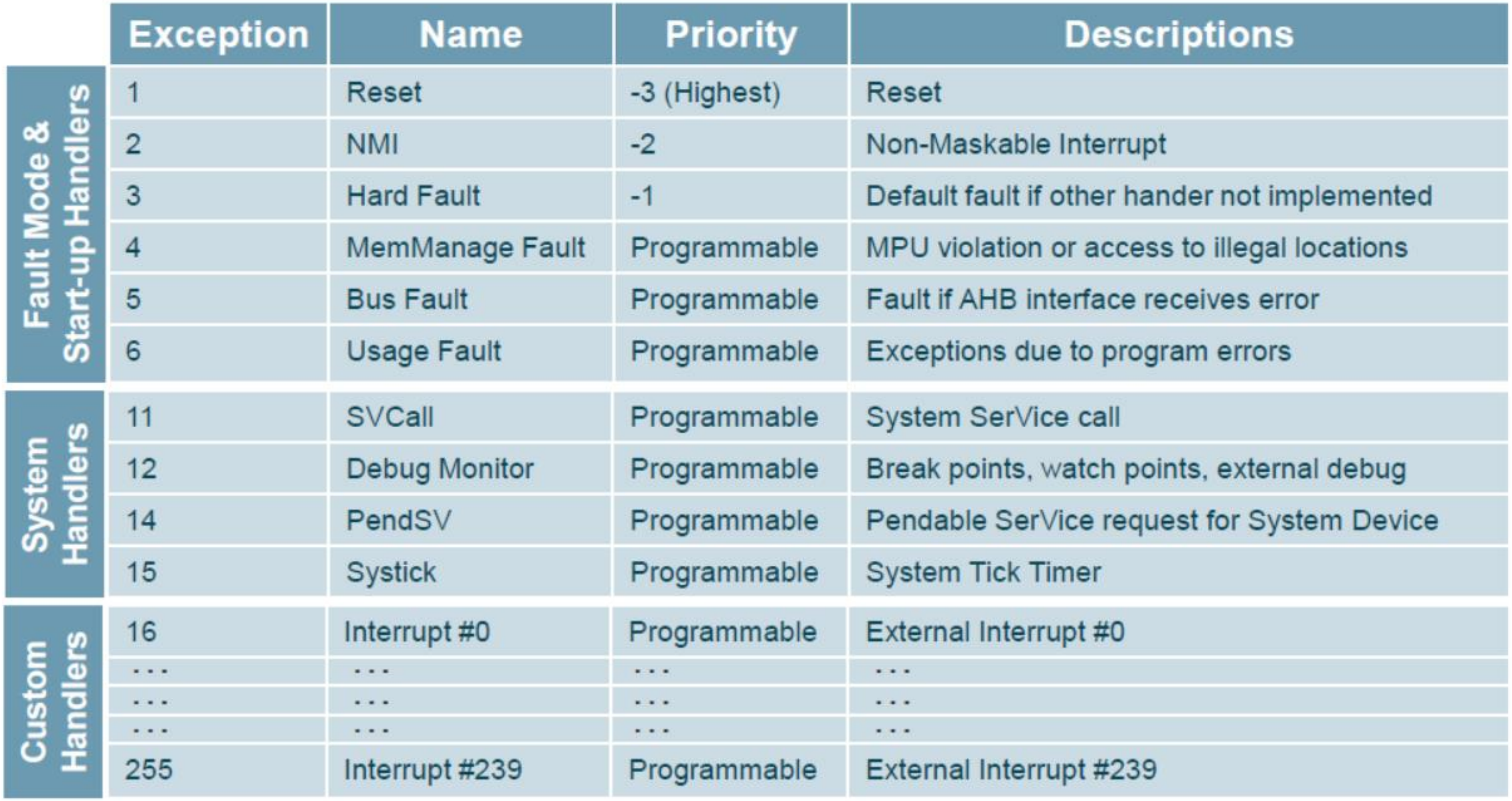
External Interrupts

- **External Interrupts handled by Nested Vectored Interrupt Controller (NVIC)**
  - Tightly coupled with processor core
- **One Non-Maskable Interrupt (NMI) supported**
- **Number of external interrupts is implementation-defined**
  - ARMv7-M supports up to 496 interrupts



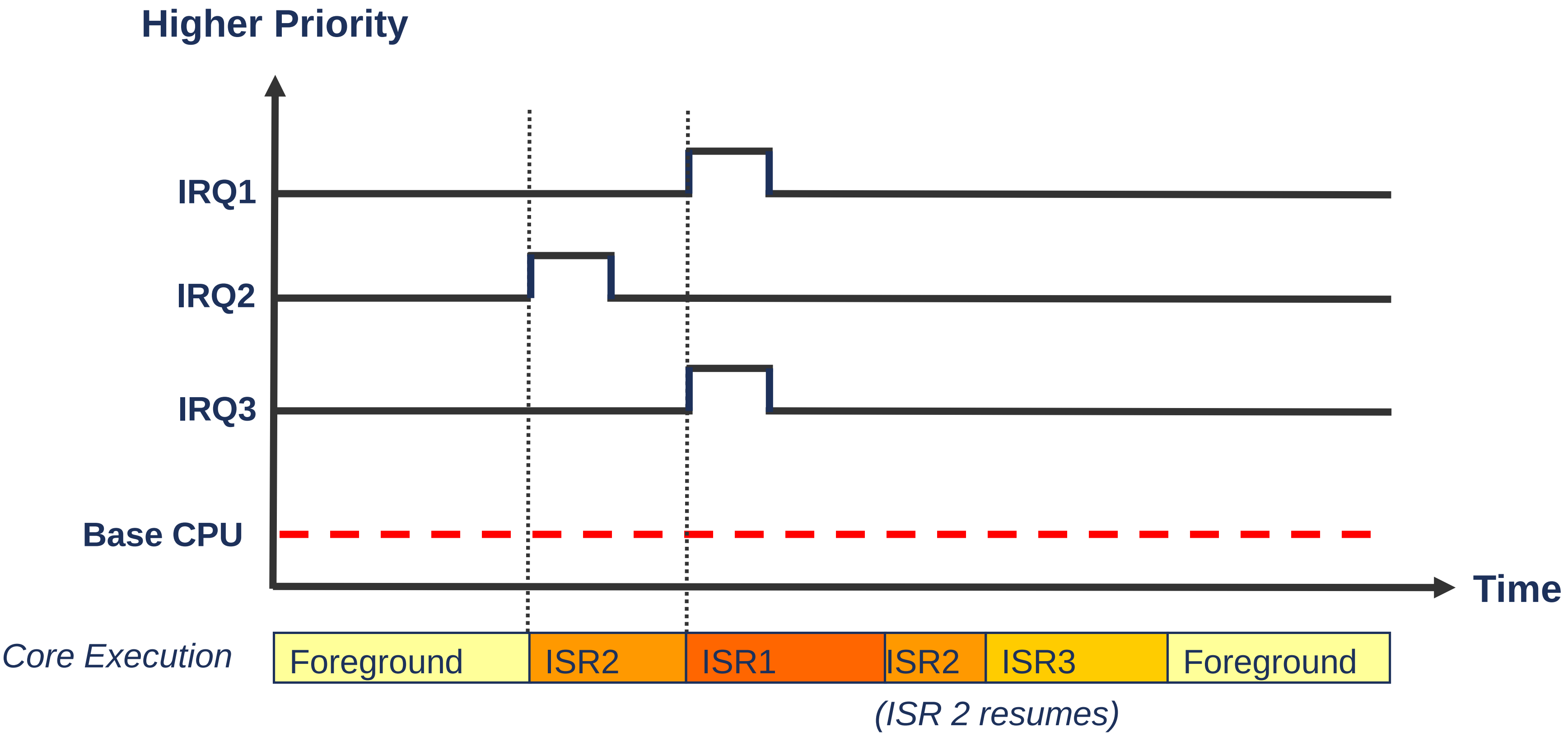
Exception & Pres-emption Ordering

- **Exception handling order is defined by programmable priority**
  - Reset, Non Maskable Interrupt (NMI) and Hard Fault have predefined pre-emption.
  - NVIC catches exceptions and pre-empts current task based on priority

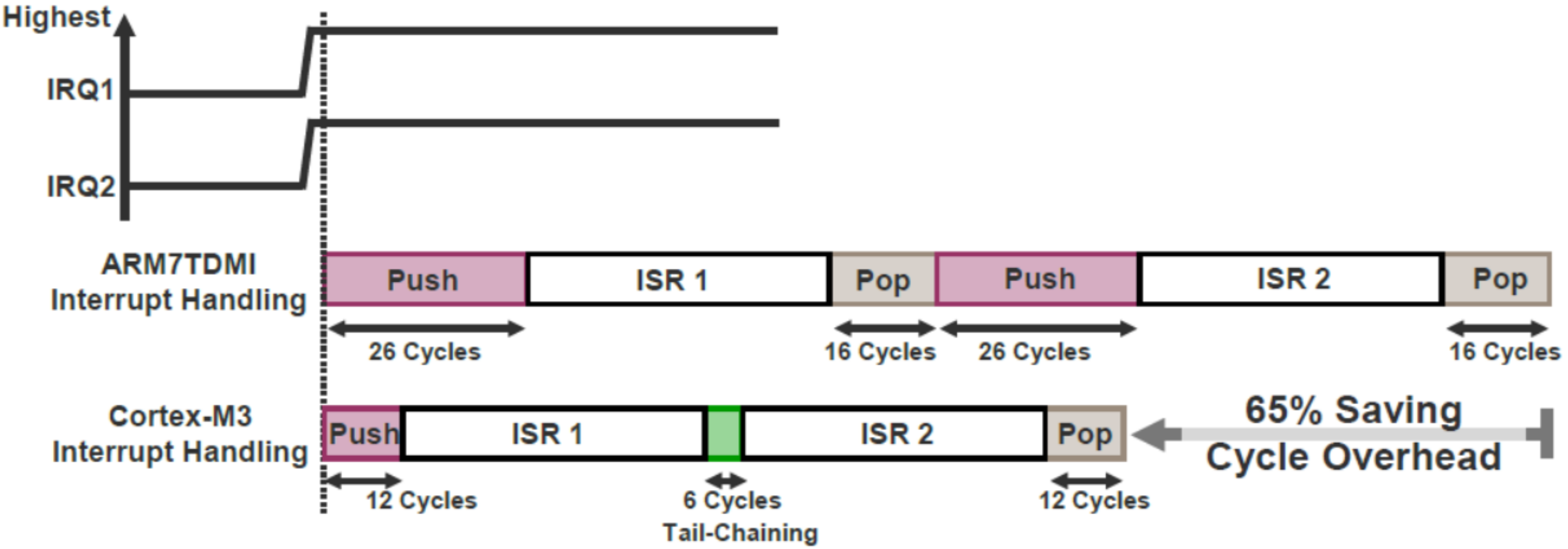


	Exception	Name	Priority	Descriptions
Fault Mode & Start-up Handlers	1	Reset	-3 (Highest)	Reset
	2	NMI	-2	Non-Maskable Interrupt
	3	Hard Fault	-1	Default fault if other hander not implemented
	4	MemManage Fault	Programmable	MPU violation or access to illegal locations
	5	Bus Fault	Programmable	Fault if AHB interface receives error
	6	Usage Fault	Programmable	Exceptions due to program errors
System Handlers	11	SVCall	Programmable	System SerVice call
	12	Debug Monitor	Programmable	Break points, watch points, external debug
	14	PendSV	Programmable	Pendable SerVice request for System Device
	15	Systick	Programmable	System Tick Timer
Custom Handlers	16	Interrupt #0	Programmable	External Interrupt #0
	...	...	...	...
	...	...	...	...
	...	...	...	...
	255	Interrupt #239	Programmable	External Interrupt #239

Exception Handling Example



Interrupt Response – Tail Chaining



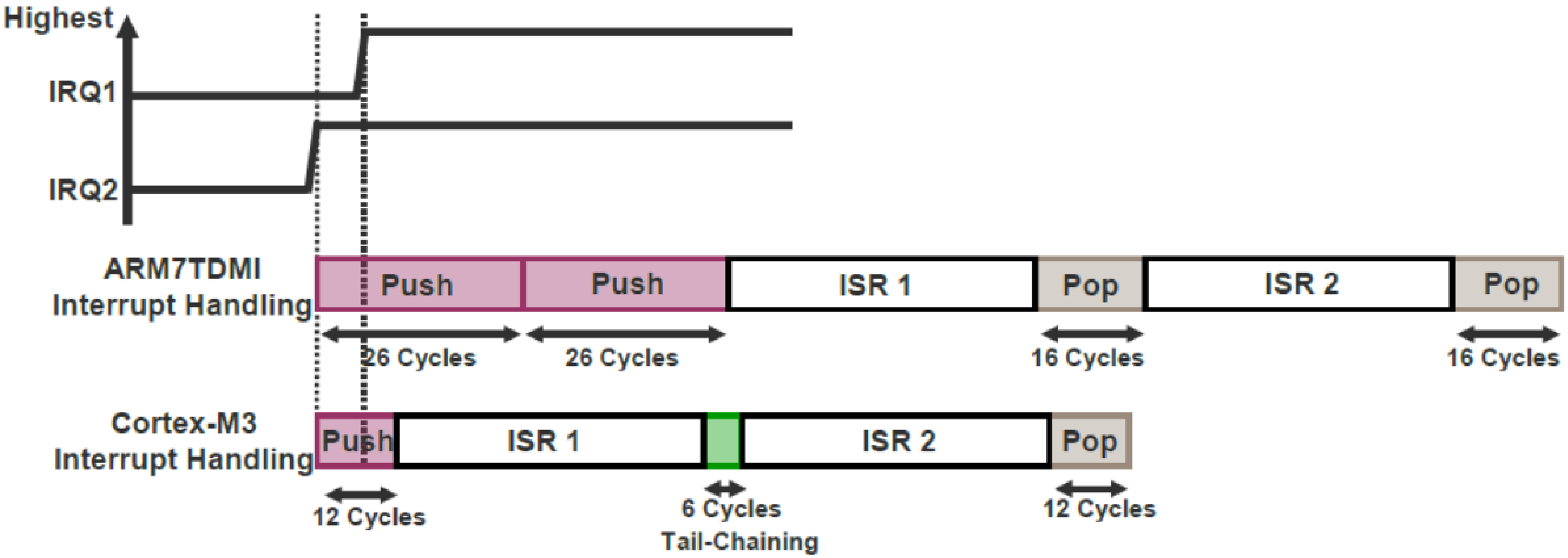
ARM7TDMI

- 26 cycles from IRQ1 to ISR1 (up to 42 cycles if in LSM)
- 42 cycles from ISR1 exit to ISR2 entry
- 16 cycles to return from ISR2

Cortex-M3

- 12 cycles from IRQ1 to ISR1 (Interruptible/Continual LSM)
- 6 cycles from ISR1 exit to ISR2 entry
- 12 cycles to return from ISR2

Interrupt Response – Late Arriving



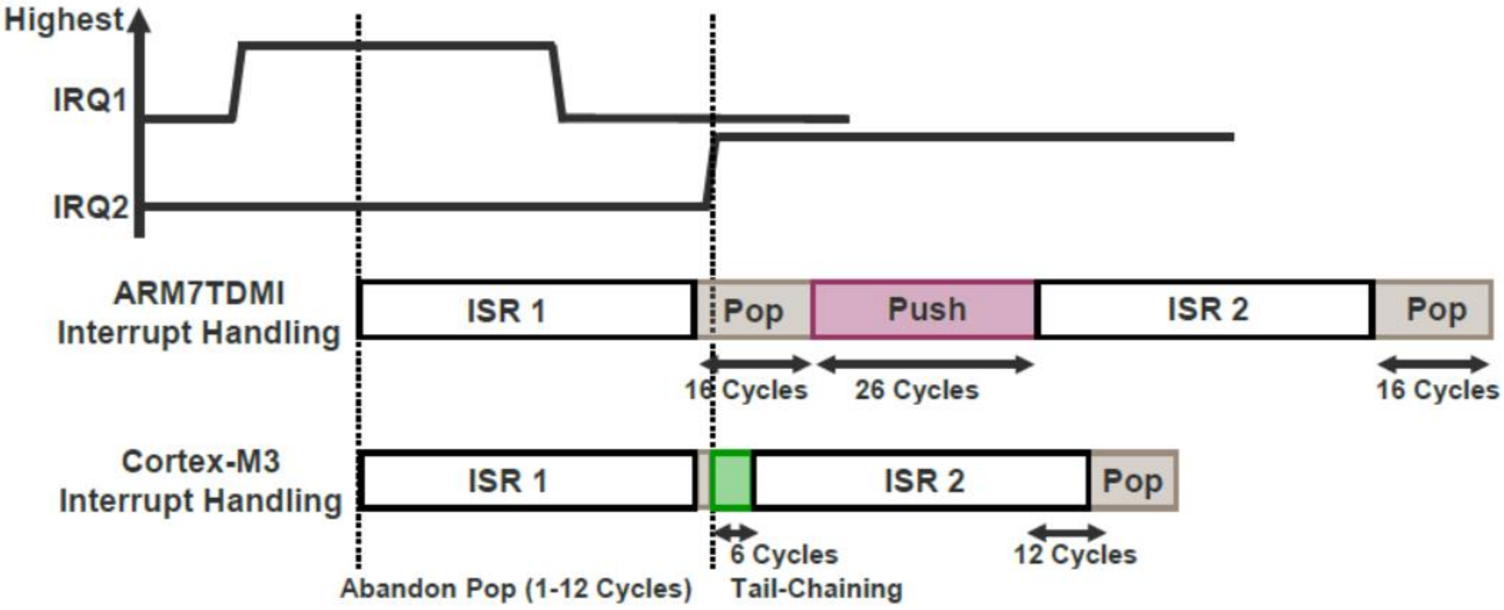
ARM7TDMI

- 26 cycles to ISR2 entered
- Immediately pre-empted by IRQ1
  Additional 26 cycles to enter ISR1.
- ISR 1 completes
  Additional 16 cycles return to ISR2.

Cortex-M3

- 12 cycles to ISR entry
- Parallel stacking & instruction fetch
- Target ISR may be changed until last cycle (PC is set)
- When IRQ1 occurs new target ISR set

Interrupt Response – Pop Pre-emption



ARM7TDMI

- Load multiple not interruptible
- Core must complete the recovery of the stack then re-stack to enter the ISR

Cortex-M3

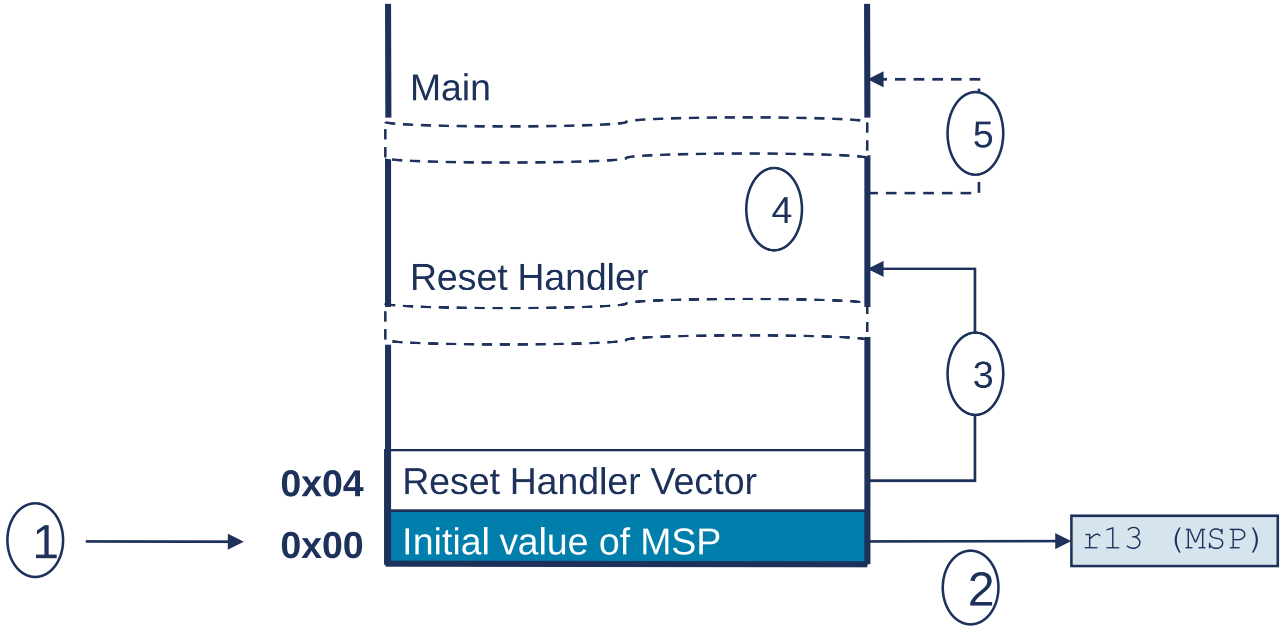
- Hardware un-stacking interruptible
- If interrupted only 6 cycles required to enter ISR2

Vector Table for ARMv7-M

- **First entry contains initial Main SP**
- **All other entries are addresses for exception handlers**
  - Must always have LSBit = 1 (for Thumb)
- **Table has up to 496 external interrupts**
  - Implementation-defined
  - Maximum table size is 2048 bytes
- **Table may be relocated**
  - Use Vector Table Offset Register
  - Still require minimal table entries at 0x0 for booting the core
- **Each exception has a vector number**
  - Used in Interrupt Control and State Register to indicate the active or pending exception type
- **Table can be generated using C code**
  - Example provided later

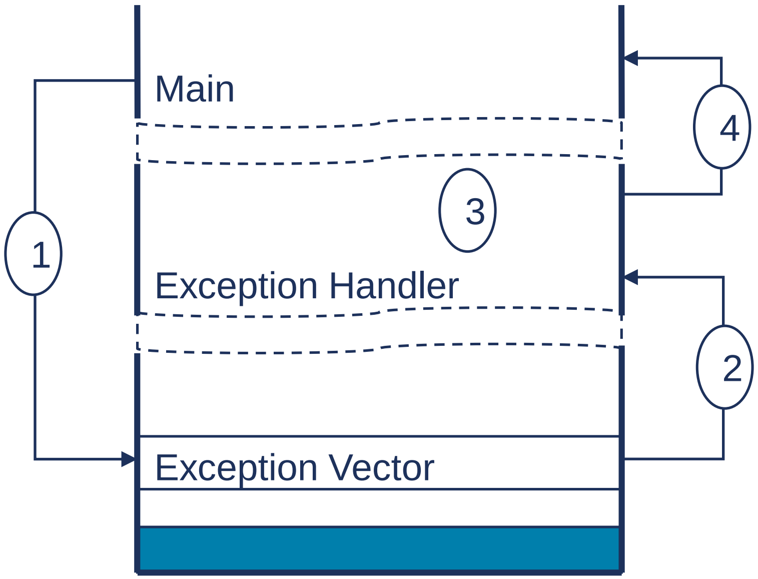
Address		Vector #
0x40 + 4*N	**External N**	16 + N
…	**…**	…
0x40	**External 0**	16
0x3C	**SysTick**	15
0x38	**PendSV**	14
0x34	**Reserved**	13
0x30	**Debug Monitor**	12
0x2C	**SVC**	11
0x1C to 0x28	**Reserved (x4)**	7-10
0x18	**Usage Fault**	6
0x14	**Bus Fault**	5
0x10	**Mem Manage Fault**	4
0x0C	**Hard Fault**	3
0x08	**NMI**	2
0x04	**Reset**	1
0x00	**Initial Main SP**	N/A

Reset Behavior



1. **A reset occurs (Reset input was asserted)**
2. **Load MSP (Main Stack Pointer) register initial value from address 0x00**
3. **Load reset handler vector address from address 0x04**
4. **Reset handler executes in Thread Mode**
5. **Optional: Reset handler branches to the main program**

Exception Behaviour



1. **Exception occurs**
   - Current instruction stream stops
   - Processor accesses vector table
2. **Vector address for the exception loaded from the vector table**
3. **Exception handler executes in Handler Mode**
4. **Exception handler returns to main**

Interrupt Service Routine Entry

- **When receiving an interrupt the processor will finish the current instruction for most instructions**
  - To minimize interrupt latency, the processor can take an interrupt during the execution of a multi-cycle instruction - see next slide
- **Processor state automatically saved to the current stack**
  - 8 registers are pushed: PC, R0-R3, R12, LR, xPSR
  - Follows ARM Architecture Procedure Calling Standard (AAPCS)
- **During (or after) state saving the address of the ISR is read from the Vector Table**
- **Link Register is modified for interrupt return**
- **First instruction of ISR executed**
  - For Cortex-M3 or Cortex-M4 the total latency is normally 12 cycles, however, interrupt late-arrival and interrupt tail-chaining can improve IRQ latency
- **ISR executes from Handler mode with Main stack**

Returning From Interrupt

- **Can return from interrupt with the following instructions when the PC is loaded with "magic" value of 0xFFFF_FFFX (same format as EXC_RETURN)**
  - LDR PC, …..
  - LDM/POP which includes loading the PC
  - BX LR (most common)
- **If no interrupts are pending, foreground state is restored**
  - Stack and state specified by EXC_RETURN is used
  - Context restore on Cortex-M3 and Cortex-M4 requires 10 cycles
- **If other interrupts are pending, the highest priority may be serviced**
  - Serviced if interrupt priority is higher than the foreground's base priority
  - Process is called Tail-Chaining as foreground state is not yet restored
  - Latency for servicing new interrupt is only 6 cycles on M3/M4 (state already saved)
- **If state restore is interrupted, it is abandoned**
  - New ISR executed without state saving (original state still intact and valid)
  - Must still fetch new vector and refill pipeline (6-cycle latency on M3/M4)

Vector Table in C

```
typedef void(* const ExecFuncPtr)(void) __irq;

#pragma arm section rodata="exceptions_area”

ExecFuncPtr exception_table[] = {
   (ExecFuncPtr)&Image$$ARM_LIB_STACK$$ZI$$Limit,    /* Initial SP */
   (ExecFuncPtr)__main,                              /* Initial PC */
   NMIException,
   HardFaultException,
   MemManageException,
   BusFaultException,
   UsageFaultException,
   0, 0, 0, 0,                         /* Reserved */
   SVCHandler,
   DebugMonitor,
   0,                                  /* Reserved */
   PendSVC,
   SysTickHandler
   /* Configurable interrupts start here...*/
};
#pragma arm section
```

The vector table at address 0x0 is minimally required to have 4 values: stack top, reset routine location, NMI ISR location, HardFault ISR location

The SVCall ISR location must be populated if the SVC instruction will be used

Once interrupts are enabled, the vector table (whether at 0 or in SRAM) must then have pointers to all enabled (by mask) exceptions

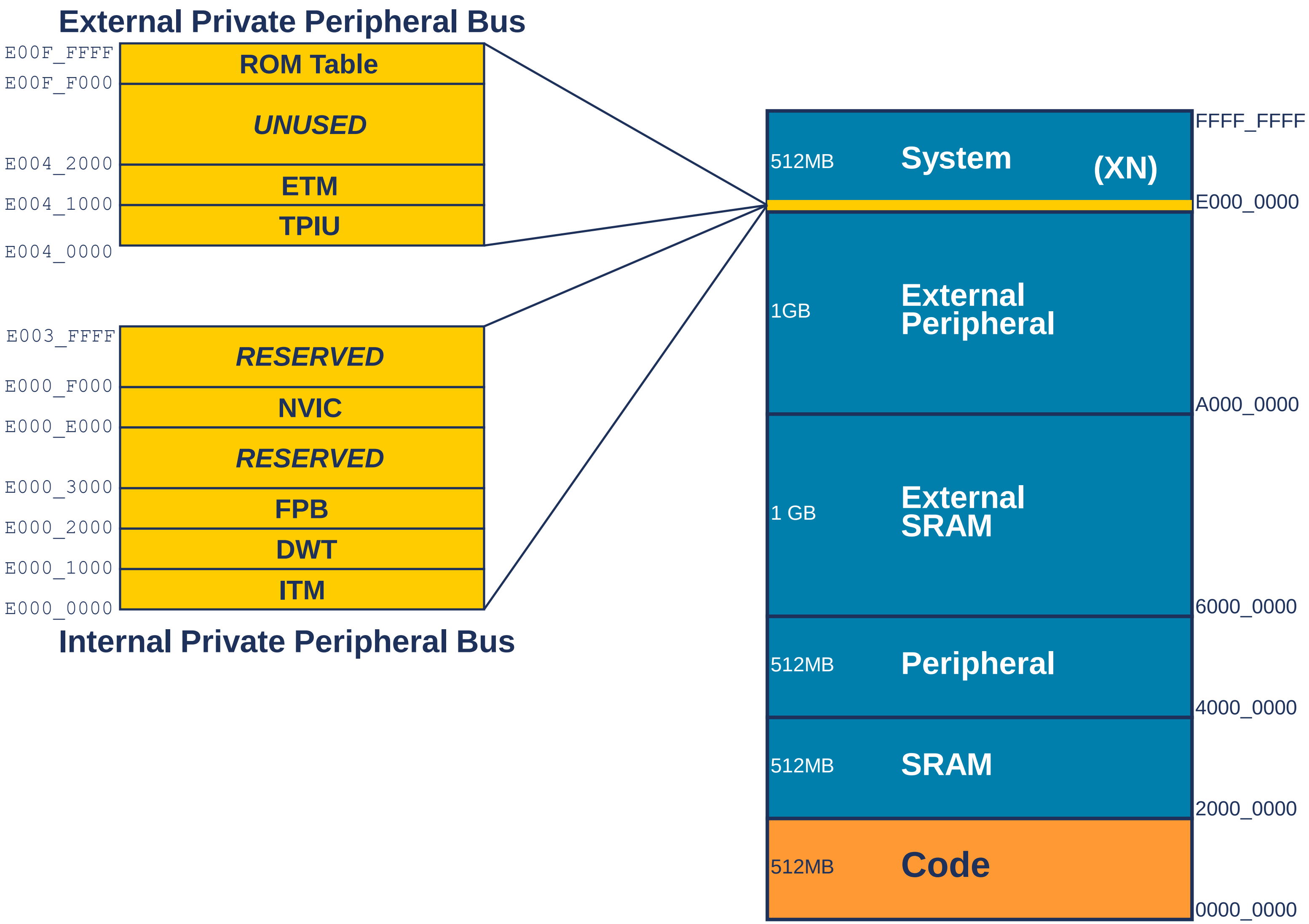
Vector Table in Assembly

```
              PRESERVE8
              THUMB
              IMPORT ||Image$$ARM_LIB_STACK$$ZI$$Limit||
              AREA    RESET, DATA, READONLY
              EXPORT  __Vectors

__Vectors     DCD     ||Image$$ARM_LIB_STACK$$ZI$$Limit||  ; Top of Stack
              DCD     Reset_Handler                ; Reset Handler
              DCD     NMI_Handler                  ; NMI Handler
              DCD     HardFault_Handler            ; Hard Fault Handler
              DCD     MemManage_Handler            ; MemManage Fault Handler
              DCD     BusFault_Handler             ; Bus Fault Handler
              DCD     UsageFault_Handler           ; Usage Fault Handler
              DCD     0, 0, 0, 0,                  ; Reserved x4
              DCD     SVC_Handler,                 ; SVCall Handler
              DCD     Debug_Monitor                ; Debug Monitor Handler
              DCD     0                            ; Reserved
              DCD     PendSV_Handler               ; PendSV Handler
              DCD     SysTick_Handler              ; SysTick Handler
              ; External vectors start here
```

Memory Systems

Processor Memory Map



External Private Peripheral Bus

E00F_FFFF
ROM Table
E00F_F000
UNUSED
E004_2000
ETM
E004_1000
TPIU
E004_0000

E003_FFFF
RESERVED
E000_F000
NVIC
E000_E000
RESERVED
E000_3000
FPB
E000_2000
DWT
E000_1000
ITM
E000_0000

Internal Private Peripheral Bus

FFFF_FFFF
512MB System (XN)
E000_0000
1GB External Peripheral
A000_0000
1 GB External SRAM
6000_0000
512MB Peripheral
4000_0000
512MB SRAM
2000_0000
512MB Code
0000_0000

Memory Types and Properties

- **There are 3 different memory types:**
  - Normal, Device and Strongly Ordered

- **Normal memory is the most flexible memory type:**
  - Suitable for different types of memory, for example, ROM, RAM, Flash and SDRAM
  - Accesses may be restarted
  - Caches and Write Buffers are permitted to work alongside Normal memory

- **Device memory is suitable for peripherals and I/O devices**
  - Caches are not permitted, but write buffers are still supported
  - Unaligned accesses are unpredictable
  - Accesses must not be restarted
    - Load/store multiple instructions should not be used to access Device memory

- **Strongly ordered memory is similar to Device memory**
  - Buffers are not supported and the PPB is marked Strongly Ordered

System Control Block

- **Memory mapped space containing registers to configure, control, and deal with interrupts, exceptions, and debug**
  - Replaces co-processor #15 in older ARM cores

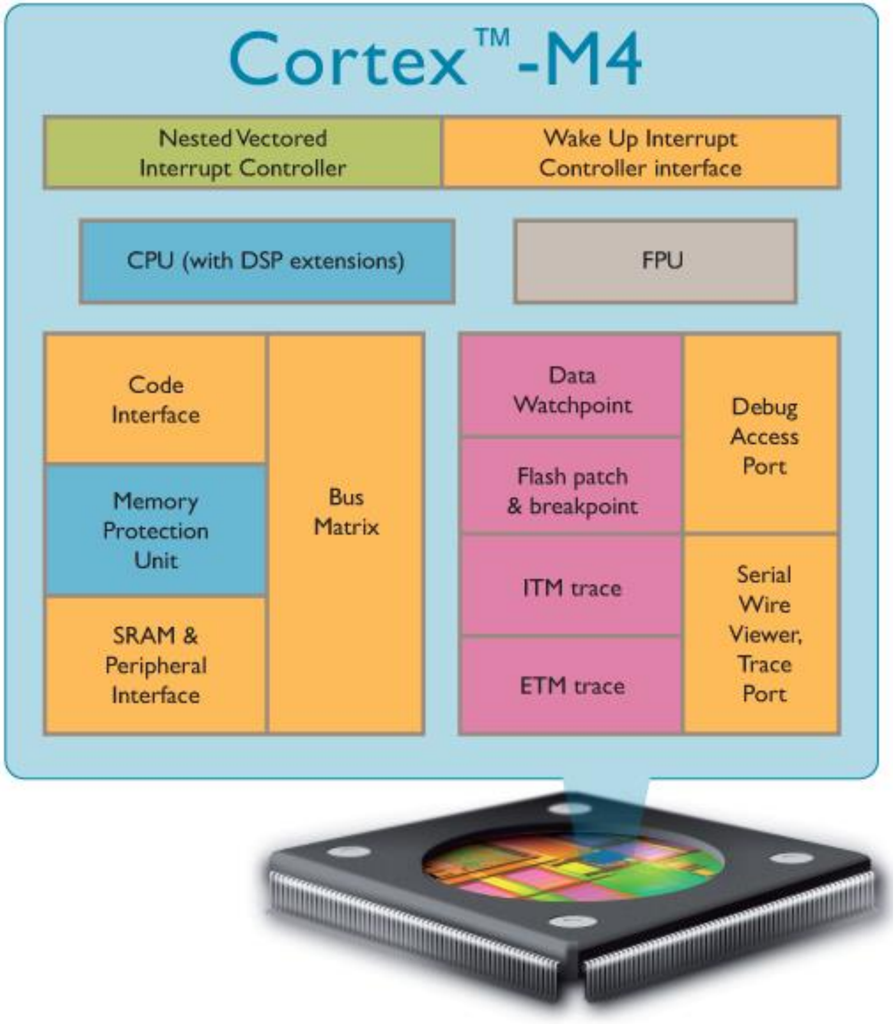
Address	Type	Reset Value	Function
0xE000E000	Read/Write	0x00000000	Master Control register - RESERVED
0xE000E004	Read Only	IMP DEFINED	Interrupt Controller Type Register
0xE000ED00	Read Only	IMP DEFINED	CPUID Base Register
0xE000ED04	Read/Write	0x00000000	Interrupt Control State Register
0xE000ED08	Read/Write	0x00000000	Vector Table Offset Register
0xE000ED0C	Read/Write	Bits[10:8] = 000	Application Interrupt/Reset Control Register

More SCB Registers

Address	Type	Reset Value	Function
0xE000ED10	Read/Write	0x00000000	System Control Register
0xE000ED14	Read/Write	0x00000000	Configuration Control Register
0xE000ED18	Read/Write	0x00000000	System Handlers 4-7 Priority Register
0xE000ED1C	Read/Write	0x00000000	System Handlers 8-11 Priority Register
0xE000ED20	Read/Write	0x00000000	System Handlers 12-15 Priority Register
0xE000ED24	Read/Write	0x00000000	System Handler Control and State Register
0xE000ED28	Read/Write	n/a - status	Configurable Fault Status Registers (3)
0xE000ED2C	Read/Write	n/a - status	HardFault Status Register
0xE000ED30	Read/Write	n/a - status	DebugFault Status Register
0xE000ED34	Read/Write	Unpredictable	MemManage Address Register
0xE000ED38	Read/Write	Unpredictable	BusFault Address Register
0xE000ED3C	Read/Write	Unpredictable	Auxiliary Fault Status Register (vendor specific)
0xE000EF00	Write Only		Software Trigger Interrupt Register

Floating Point Extensions

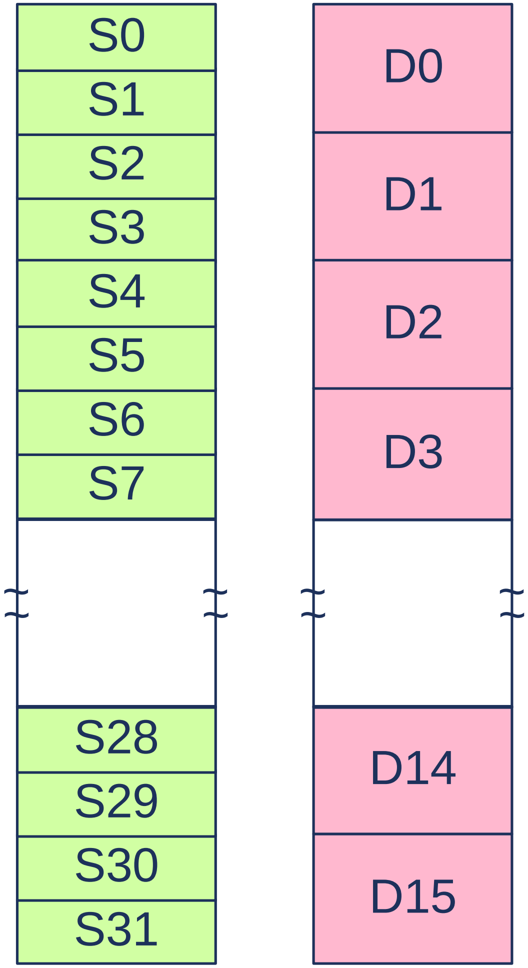
Cortex-M4


Cortex™-M4
Nested Vectored Interrupt Controller
Wake Up Interrupt Controller interface
CPU (with DSP extensions)
FPU
Code Interface
Memory Protection Unit
SRAM & Peripheral Interface
Bus Matrix
Data Watchpoint
Flash patch & breakpoint
ITM trace
ETM trace
Debug Access Port
Serial Wire Viewer, Trace Port

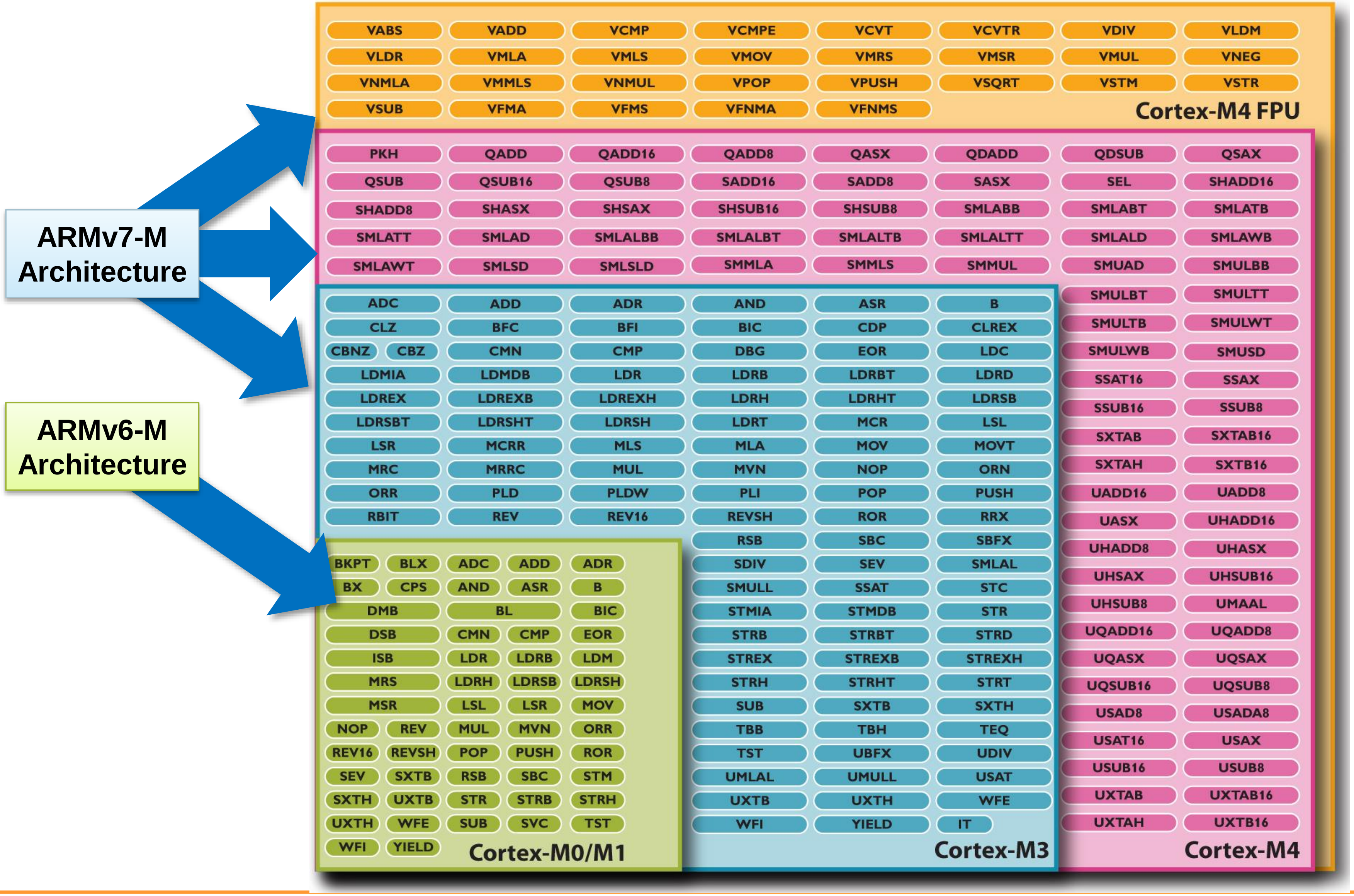
- **ARMv7E-M Architecture**
  - Thumb-2 only
  - DSP extensions
- **Optional FPU (Cortex-M4F)**
- **Otherwise, same as Cortex-M3**
- **Implements full Thumb-2 instruction set**
  - Saturated math (e.g. QADD)
  - Packing and unpacking (e.g. UXTB)
  - Signed multiply (e.g. SMULTB)
  - SIMD (e.g. ADD8)

Cortex-M4F Floating Point Registers

- **FPU provides a further 32 single-precision registers**
- **Can be viewed as either**
  - 32 x 32-bit registers
  - 16 x 64-bit doubleword registers
  - Any combination of the above



Binary Upwards Compatibility



ARMv7-M Architecture

ARMv6-M Architecture

Cortex-M4 FPU

VABS VADD VCMP VCMPE VCVT VCVTR VDIV VLDM
VLDR VMLA VMLS VMOV VMRS VMSR VMUL VNEG
VNMLA VMMLS VNMUL VPOP VPUSH VSQRT VSTM VSTR
VSUB VFMA VFMS VFNMA VFNMS

Cortex-M4

PKH QADD QADD16 QADD8 QASX QDADD QDSUB QSAX
QSUB QSUB16 QSUB8 SADD16 SADD8 SASX SEL SHADD16
SHADD8 SHASX SHSAX SHSUB16 SHSUB8 SMLABB SMLABT SMLATB
SMLATT SMLAD SMLALBB SMLALBT SMLALTB SMLALTT SMLALD SMLAWB
SMLAWT SMLSD SMLSLD SMMLA SMMLS SMMUL SMUAD SMULBB
SMULBT SMULTT
SMULTB SMULWT
SMULWB SMUSD
SSAT16 SSAX
SSUB16 SSUB8
SXTAB SXTAB16
SXTAH SXTB16
UADD16 UADD8
UASX UHADD16
UHADD8 UHASX
UHSAX UHSUB16
UHSUB8 UMAAL
UQADD16 UQADD8
UQASX UQSAX
UQSUB16 UQSUB8
USAD8 USADA8
USAT16 USAX
USUB16 USUB8
UXTAB UXTAB16
UXTAH UXTB16

Cortex-M3

ADC ADD ADR AND ASR B
CLZ BFC BFI BIC CDP CLREX
CBNZ CBZ CMN CMP DBG EOR LDC
LDMIA LDMDB LDR LDRB LDRBT LDRD
LDREX LDREXB LDREXH LDRH LDRHT LDRSB
LDRSBT LDRSHT LDRSH LDRT MCR LSL
LSR MCRR MLS MLA MOV MOVT
MRC MRRC MUL MVN NOP ORN
ORR PLD PLDW PLI POP PUSH
RBIT REV REV16 REVSH ROR RRX
RSB SBC SBFX
SDIV SEV SMLAL
SMULL SSAT STC
STMIA STMDB STR
STRB STRBT STRD
STREX STREXB STREXH
STRH STRHT STRT
SUB SXTB SXTH
TBB TBH TEQ
TST UBFX UDIV
UMLAL UMULL USAT
UXTB UXTH WFE
WFI YIELD IT

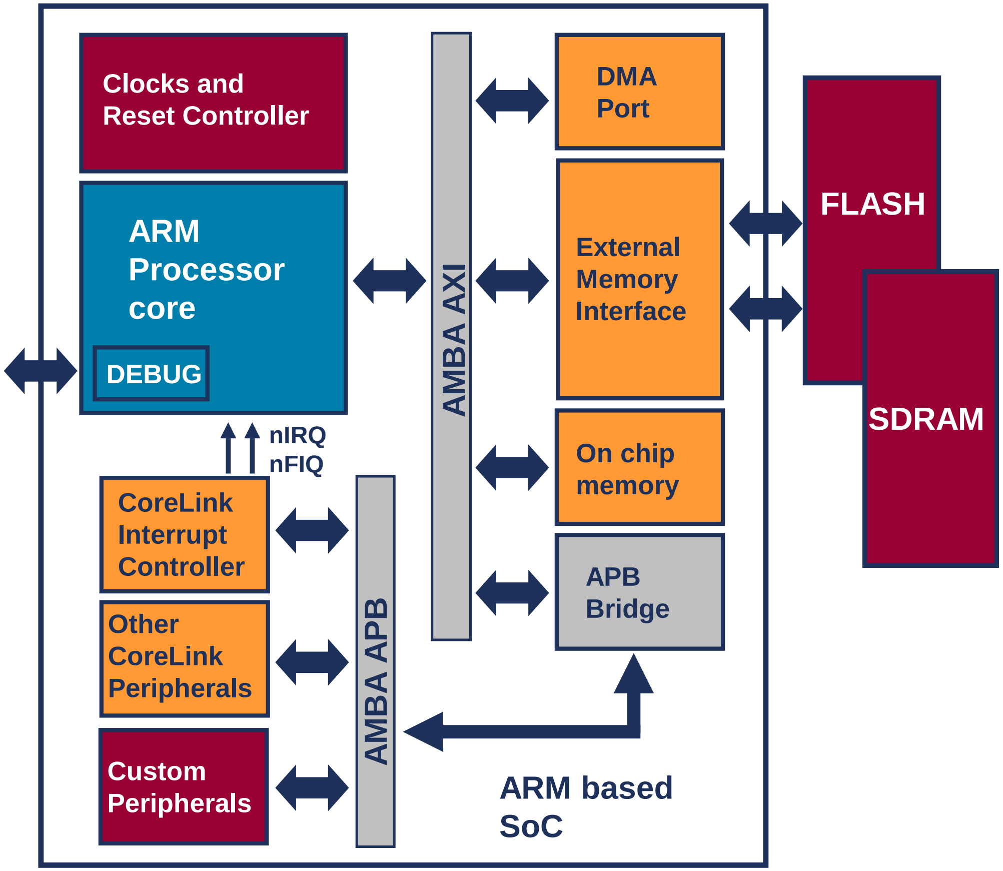
Cortex-M0/M1

BKPT BLX ADC ADD ADR
BX CPS AND ASR B
DMB BL BIC
DSB CMN CMP EOR
ISB LDR LDRB LDM
MRS LDRH LDRSB LDRSH
MSR LSL LSR MOV
NOP REV MUL MVN ORR
REV16 REVSH POP PUSH ROR
SEV SXTB RSB SBC STM
SXTH UXTB STR STRB STRH
UXTH WFE SUB SVC TST
WFI YIELD

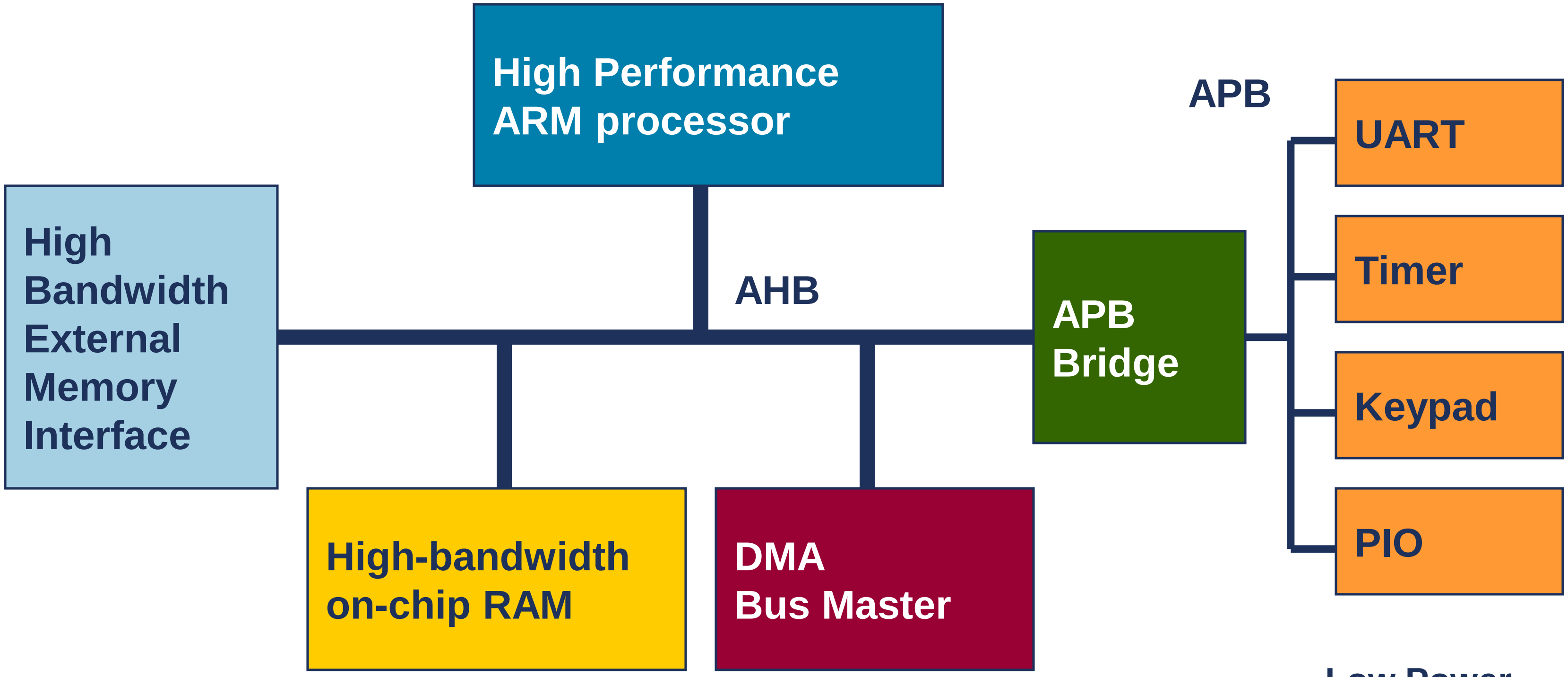
ARM System Design

Example ARM-based system

- **ARM core deeply embedded within an SoC**
  - External debug and trace via JTAG or CoreSight interface
- **Design can have both external and internal memories**
  - Varying width, speed and size – depending on system requirements
- **Can include ARM licensed CoreLink peripherals**
  - Interrupt controller, since core only has two interrupt sources
  - Other peripherals and interfaces
- **Can include on-chip memory from ARM Artisan Physical IP Libraries**
- **Elements connected using AMBA (Advanced Microcontroller Bus Architecture)**



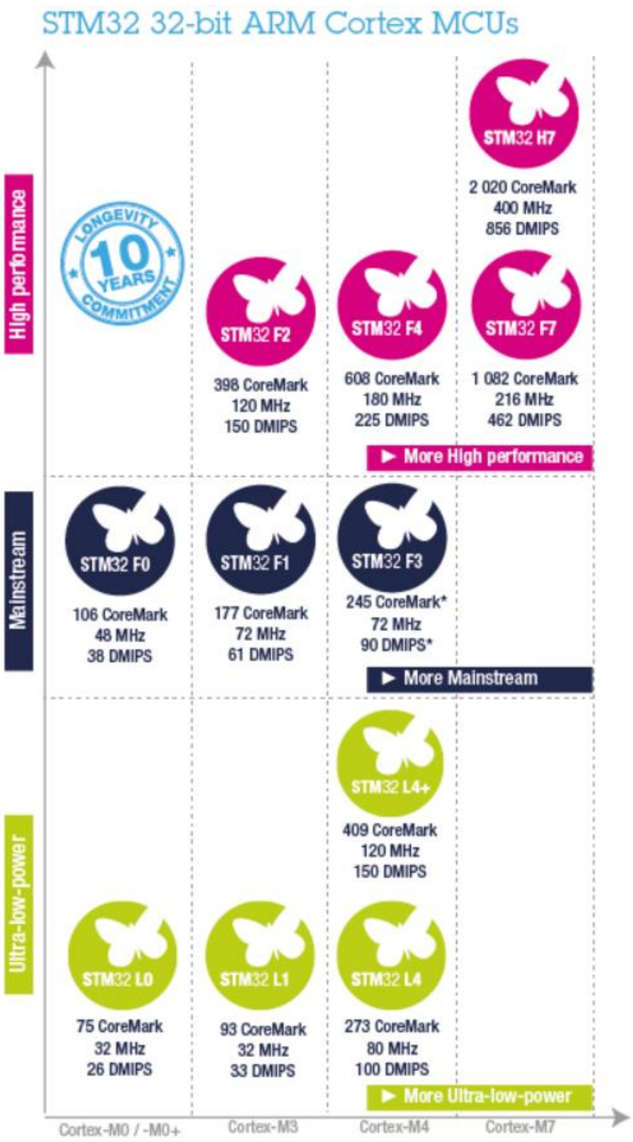
An Example AMBA System



Low Power
Non-pipelined
Simple Interface

High Performance
Pipelined
Burst Support
Multiple Bus Masters

STM32 32-bit ARM Cortex MCUs



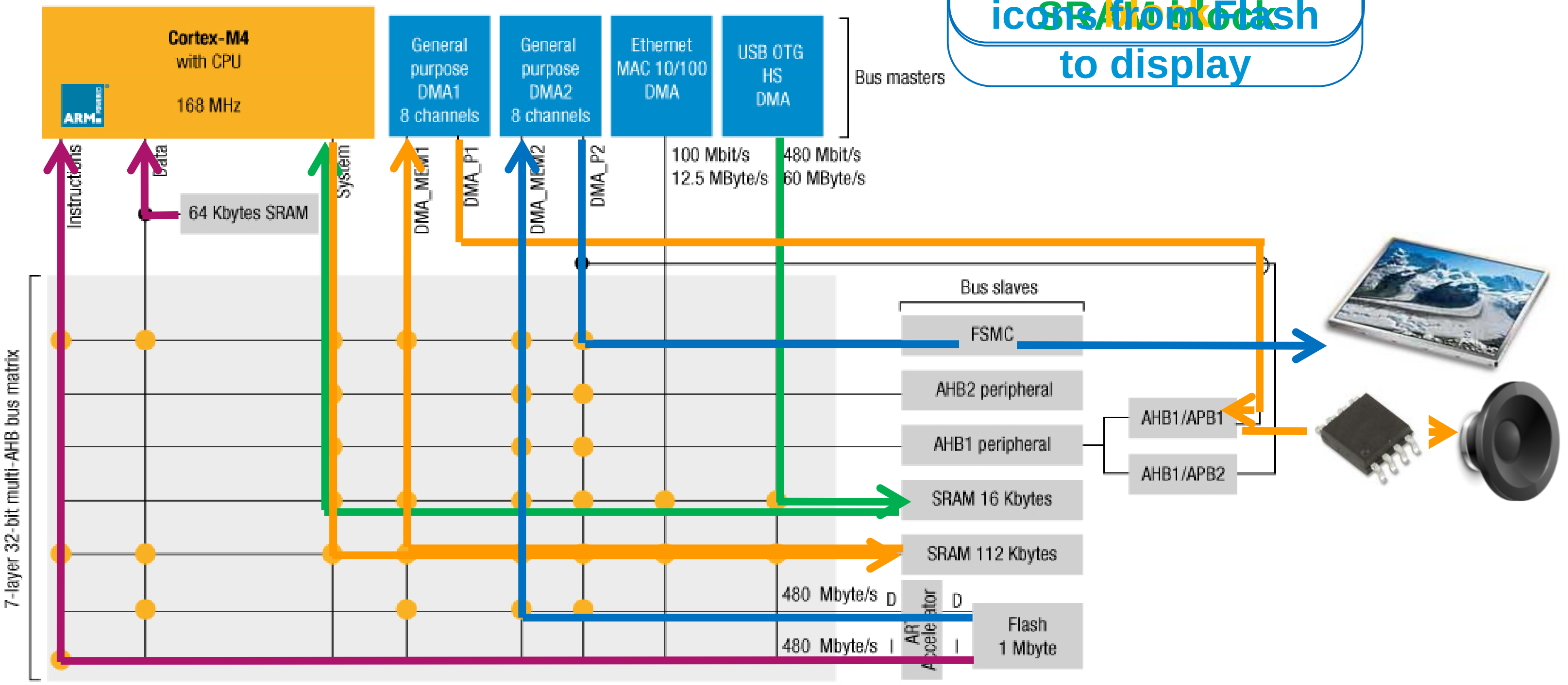
STM32 F4 series

High-performance Cortex™-M4 MCU

Real-time performance

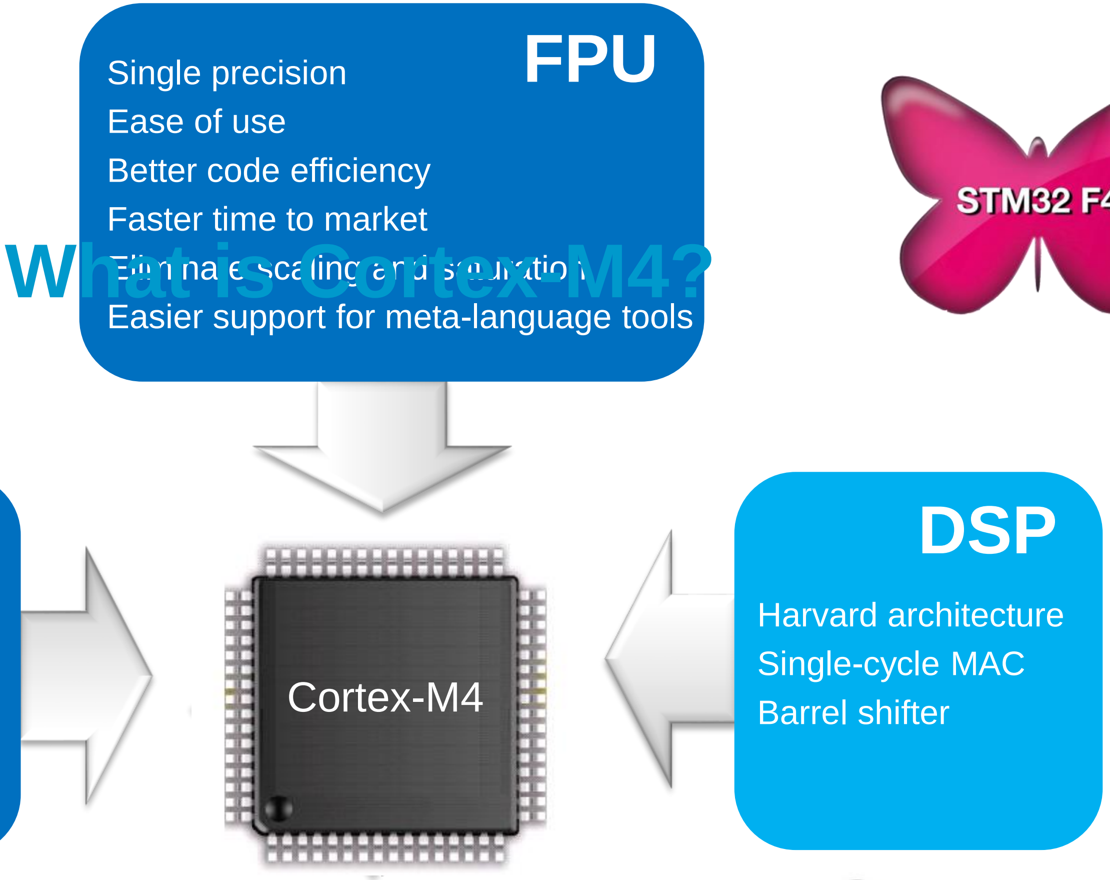
32-bit multi-AHB bus matrix

User interface: DMA transfers of the graphical icons from Flash to display



STM32 F4 series
High-performance digital signal controller

What is Cortex-M4?



FPU

Single precision

Ease of use

Better code efficiency

Faster time to market

Eliminate scaling and saturation

Easier support for meta-language tools

MCU

Ease of use of C programming

Interrupt handling

Ultra-low power

Cortex-M4

DSP

Harvard architecture

Single-cycle MAC

Barrel shifter

STM32 product series

4 product series

Common core peripherals and architecture:

- Communication peripherals: USART, SPI, I²C
- Multiple general-purpose timers
- Integrated reset and brown-out warning
- Multiple DMA
- 2x watchdogs
Real-time clock
- Integrated regulator
PLL and clock circuit
- External memory interface (FSMC)
- Dual 12-bit DAC
- Up to 3x 12-bit ADC (up to 0.41 µs)
- Main oscillator and 32 kHz oscillator
- Low-speed and high-speed internal RC oscillators
- -40 to +85 °C and up to 105 °C operating temperature range
- Low voltage 2.0 to 3.6 V or 1.65/1.7 to 3.6 V (depending on series)
5.0 V tolerant I/Os
- Temperature sensor

+

STM32 F4 series - High performance with DSP (STM32F405/415/407/417)

168 MHz Cortex-M4 with DSP and FPU	Up to 192-Kbyte SRAM	Up to 1-Mbyte Flash	2x USB 2.0 OTG FS/HS	3-phase MC timer	2x CAN 2.0B	SDIO 2x I²S audio Camera IF	Ethernet IEEE 1588	Crypto/hash processor and RNG

STM32 F4

STM32 F2 series - High performance (STM32F205/215/207/217)

120 MHz Cortex-M3 CPU	Up to 128-Kbyte SRAM	Up to 1-Mbyte Flash	2x USB 2.0 OTG FS/HS	3-phase MC timer	2x CAN 2.0B	SDIO 2x I²S audio Camera IF	Ethernet IEEE 1588	Crypto/hash processor and RNG

STM32 F2

STM32 F1 series - Connectivity line (STM32F105/107)

72 MHz Cortex-M3 CPU	Up to 64-Kbyte SRAM	Up to 256-Kbyte Flash	USB 2.0 OTG FS	3-phase MC timer	2x CAN 2.0B	2x I²S audio	Ethernet IEEE 1588

STM32 F1 series - Performance line (STM32F103)

72 MHz Cortex-M3 CPU	Up to 96-Kbyte SRAM	Up to 1-Mbyte Flash	USB FS device	3-phase MC timer	CAN 2.0B	SDIO 2x I²S

STM32 F1 series - USB Access line (STM32F102)

48 MHz Cortex-M3 CPU	Up to 16-Kbyte SRAM	Up to 128-Kbyte Flash	USB FS device

STM32 F1

STM32 F1 series - Access line (STM32F101)

36 MHz Cortex-M3 CPU	Up to 80-Kbyte SRAM	Up to 1-Mbyte Flash

STM32 F1 series - Value line (STM32F100)

24 MHz Cortex-M3 CPU	Up to 32-Kbyte SRAM	Up to 512-Kbyte Flash	3-phase MC timer	CEC

STM32 L1 series - Ultra-low-power (STM32F151/152)

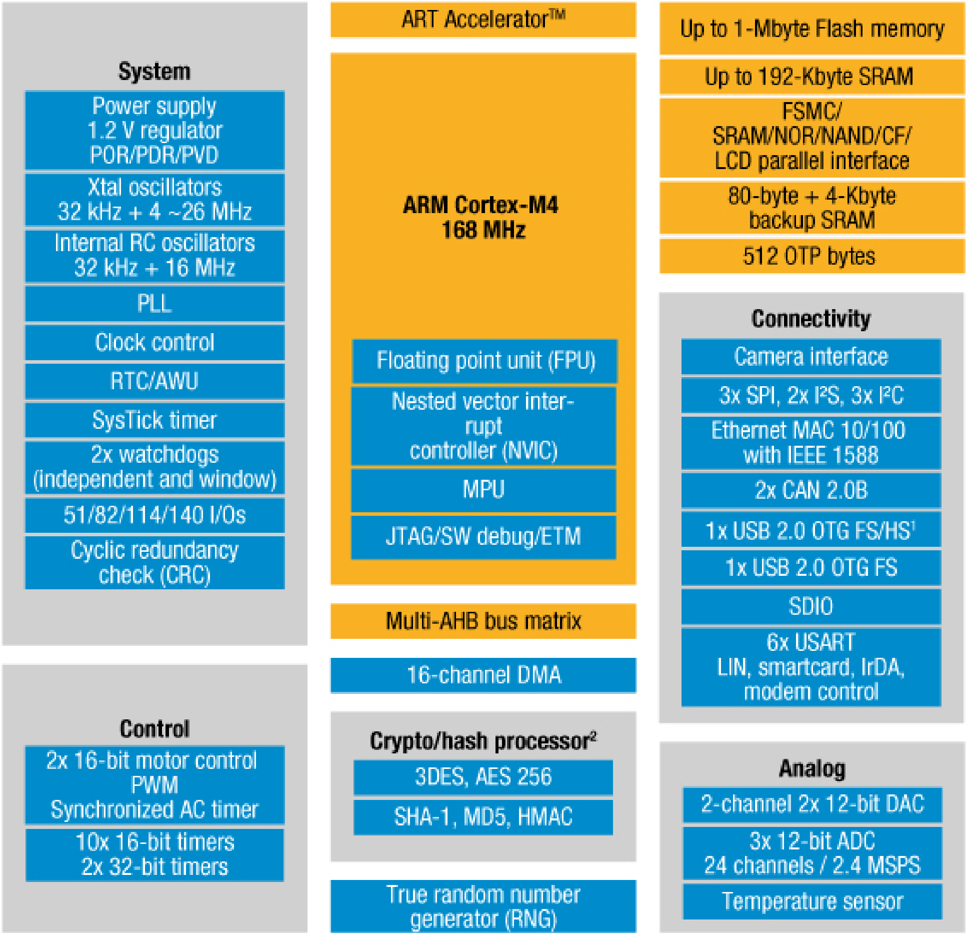
32 MHz Cortex-M3 CPU	Up to 48-Kbyte SRAM	Up to 384-Kbyte Flash	USB FS device	Data EEPROM up to 12 Kbytes	LCD 8x40 4x44	Comparator	BOR MSI VScal	AES 128-bit

STM32 L1

STM32 F4 block diagram

Feature highlight

- 168 MHz Cortex-M4 CPU
  - Floating point unit (FPU)
  - ART Accelerator ™
  - Multi-level AHB bus matrix
- 1-Mbyte Flash, 192-Kbyte SRAM
- 1.7 to 3.6 V supply
- RTC: <1 µA typ, sub second accuracy
- 2x full duplex I²S
- 3x 12-bit ADC 0.41 µs/2.4 MSPS
- 168 MHz timers



System
Power supply
1.2 V regulator
POR/PDR/PVD
Xtal oscillators
32 kHz + 4 ~26 MHz
Internal RC oscillators
32 kHz + 16 MHz
PLL
Clock control
RTC/AWU
SysTick timer
2x watchdogs
(independent and window)
51/82/114/140 I/Os
Cyclic redundancy
check (CRC)

Control
2x 16-bit motor control
PWM
Synchronized AC timer
10x 16-bit timers
2x 32-bit timers

ART Accelerator™

ARM Cortex-M4
168 MHz
Floating point unit (FPU)
Nested vector inter-
rupt
controller (NVIC)
MPU
JTAG/SW debug/ETM

Multi-AHB bus matrix

16-channel DMA

Crypto/hash processor[2]
3DES, AES 256
SHA-1, MD5, HMAC

True random number
generator (RNG)

Up to 1-Mbyte Flash memory
Up to 192-Kbyte SRAM
FSMC/
SRAM/NOR/NAND/CF/
LCD parallel interface
80-byte + 4-Kbyte
backup SRAM
512 OTP bytes

Connectivity
Camera interface
3x SPI, 2x I²S, 3x I²C
Ethernet MAC 10/100
with IEEE 1588
2x CAN 2.0B
1x USB 2.0 OTG FS/HS[1]
1x USB 2.0 OTG FS
SDIO
6x USART
LIN, smartcard, IrDA,
modem control

Analog
2-channel 2x 12-bit DAC
3x 12-bit ADC
24 channels / 2.4 MSPS
Temperature sensor

Notes:

1. HS requires an external PHY connected to the ULPI interface
2. Crypto/hash processor on STM32F417 and STM32F415