Overview of ARM Architecture & Cortex-M Processors

Lecture 3

Yeongpil Cho

Hanyang University

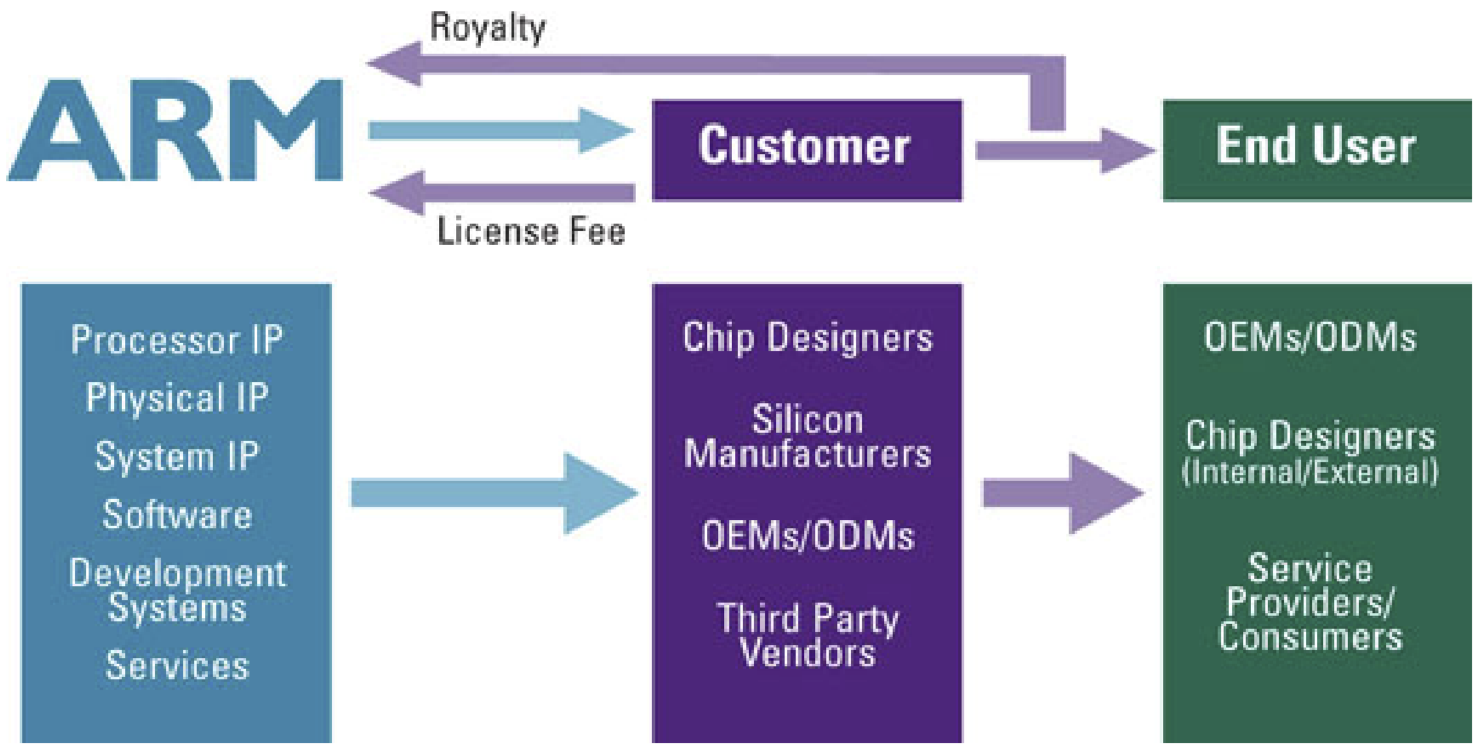
Topics

- ARM Architecture
    - Overview
- Cortex-M Processors
    - Overview
    - Programmer’s model

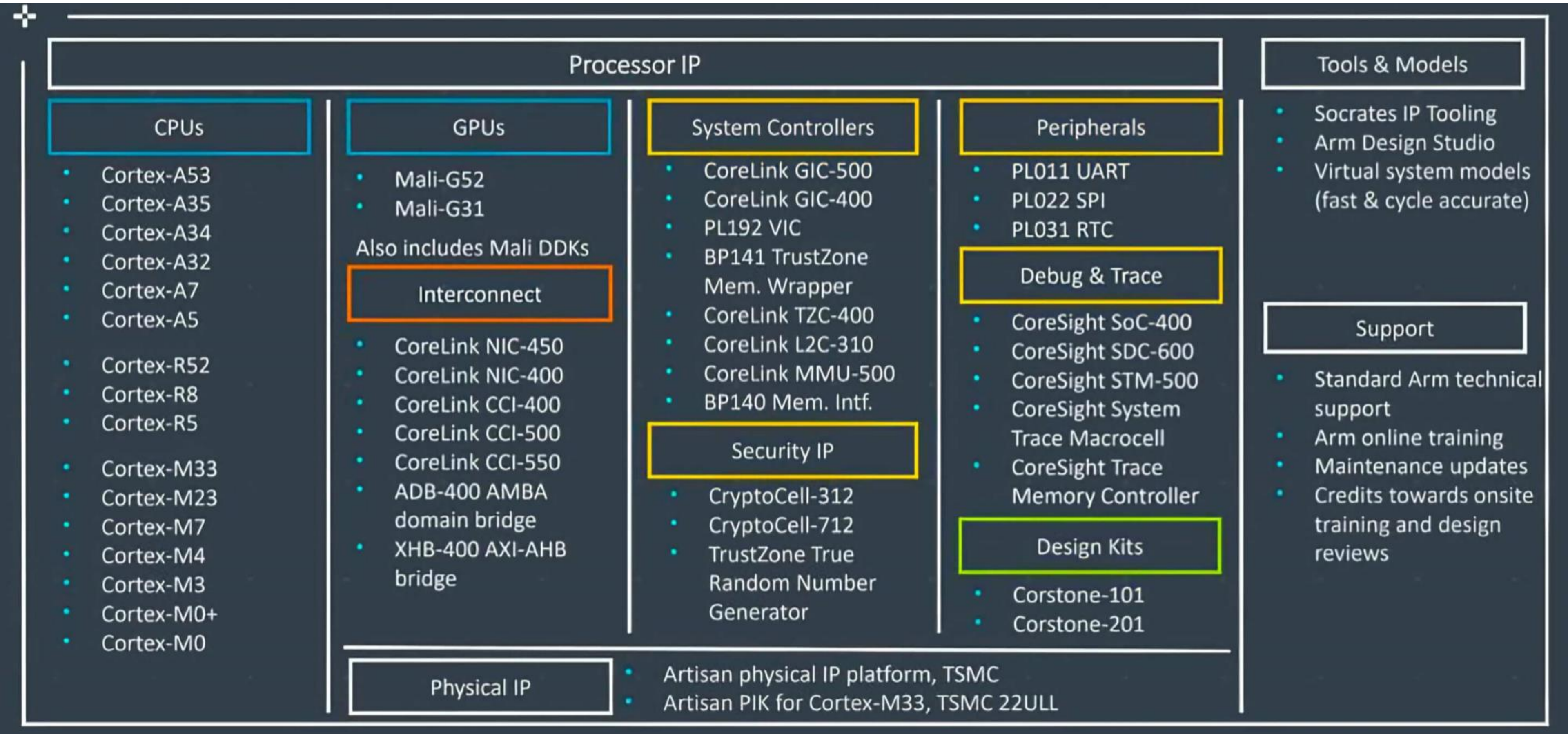
ARM Architecture Overview

ARM

- ARM is a market leader in mobile/embedded processor area.
  - As of 2020, more than 180 billions of arm-based chips have shipped.
- ARM does not manufacture silicon



ARM Product List



Processor IP

CPUs

- Cortex-A53
- Cortex-A35
- Cortex-A34
- Cortex-A32
- Cortex-A7
- Cortex-A5

- Cortex-R52
- Cortex-R8
- Cortex-R5

- Cortex-M33
- Cortex-M23
- Cortex-M7
- Cortex-M4
- Cortex-M3
- Cortex-M0+
- Cortex-M0

GPUs

- Mali-G52
- Mali-G31

Also includes Mali DDKs

Interconnect

- CoreLink NIC-450
- CoreLink NIC-400
- CoreLink CCI-400
- CoreLink CCI-500
- CoreLink CCI-550
- ADB-400 AMBA domain bridge
- XHB-400 AXI-AHB bridge

System Controllers

- CoreLink GIC-500
- CoreLink GIC-400
- PL192 VIC
- BP141 TrustZone Mem. Wrapper
- CoreLink TZC-400
- CoreLink L2C-310
- CoreLink MMU-500
- BP140 Mem. Intf.

Security IP

- CryptoCell-312
- CryptoCell-712
- TrustZone True Random Number Generator

Peripherals

- PL011 UART
- PL022 SPI
- PL031 RTC

Debug & Trace

- CoreSight SoC-400
- CoreSight SDC-600
- CoreSight STM-500
- CoreSight System Trace Macrocell
- CoreSight Trace Memory Controller

Design Kits

- Corstone-101
- Corstone-201

Physical IP

- Artisan physical IP platform, TSMC
- Artisan PIK for Cortex-M33, TSMC 22ULL

Tools & Models

- Socrates IP Tooling
- Arm Design Studio
- Virtual system models (fast & cycle accurate)

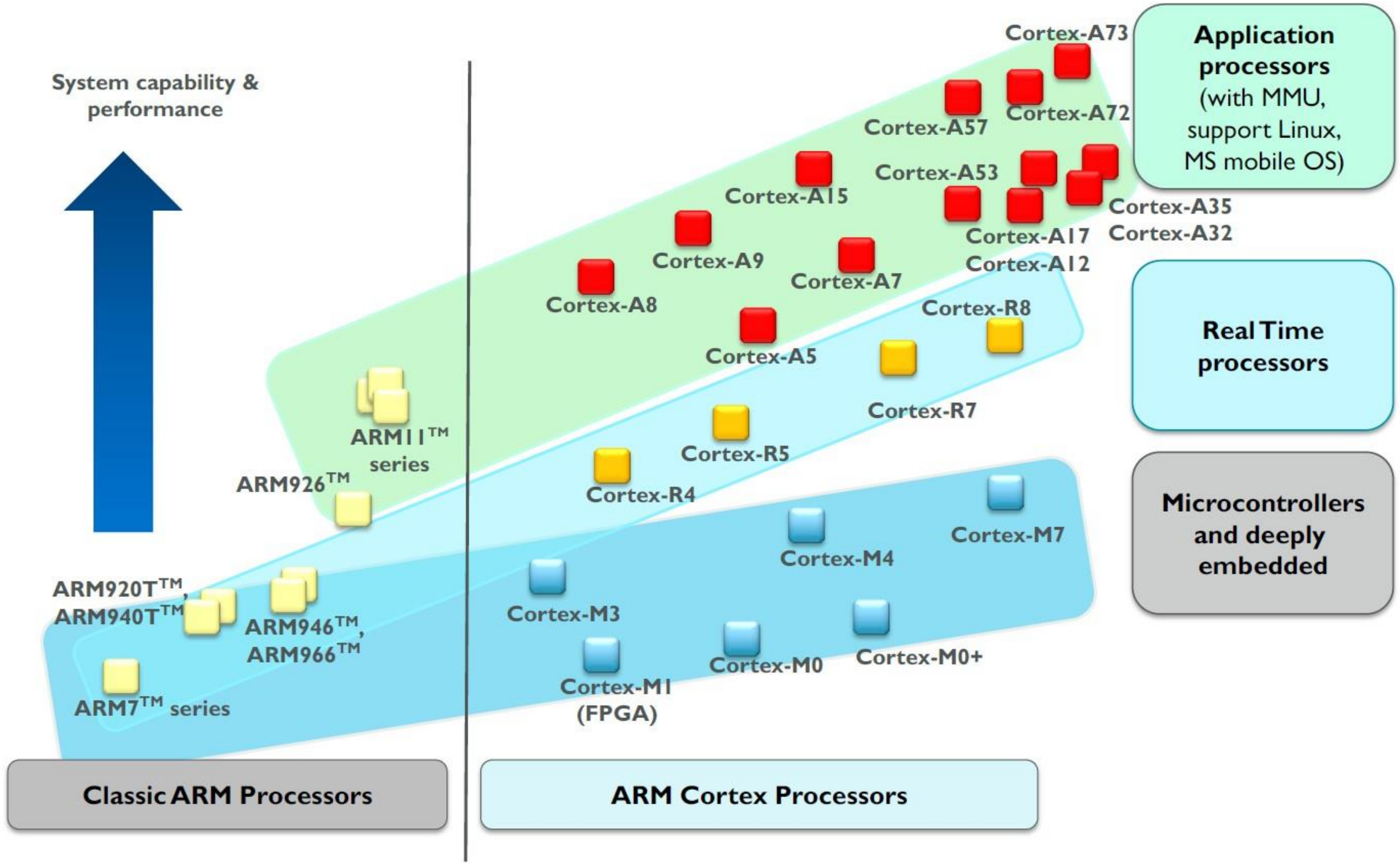
Support

- Standard Arm technical support
- Arm online training
- Maintenance updates
- Credits towards onsite training and design reviews

ARM Processor Architecture

- Based upon RISC (Reduced Instruction Set) Architecture with enhancements to meet requirements of embedded applications
    - Fixed length instructions
    - Load-store architecture
        - Memory-to-register load instructions
        - Register-to-memory store instructions
    - A large uniform register file
    - 32-bit architecture (v1-v7), 64-bit architecture (v8-v9)
    - Good performance/power

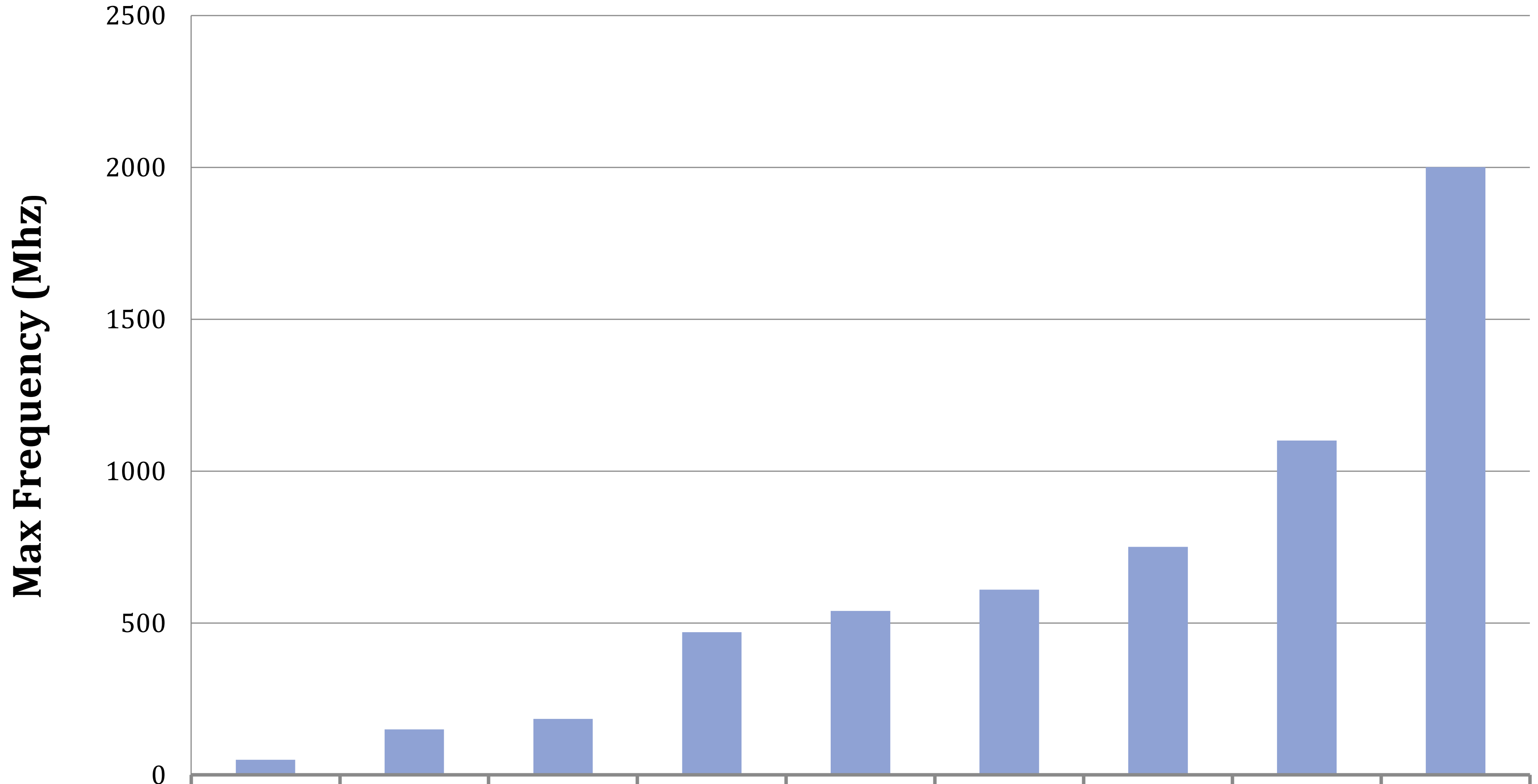
ARM Processor Family



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)

Relative Performance*

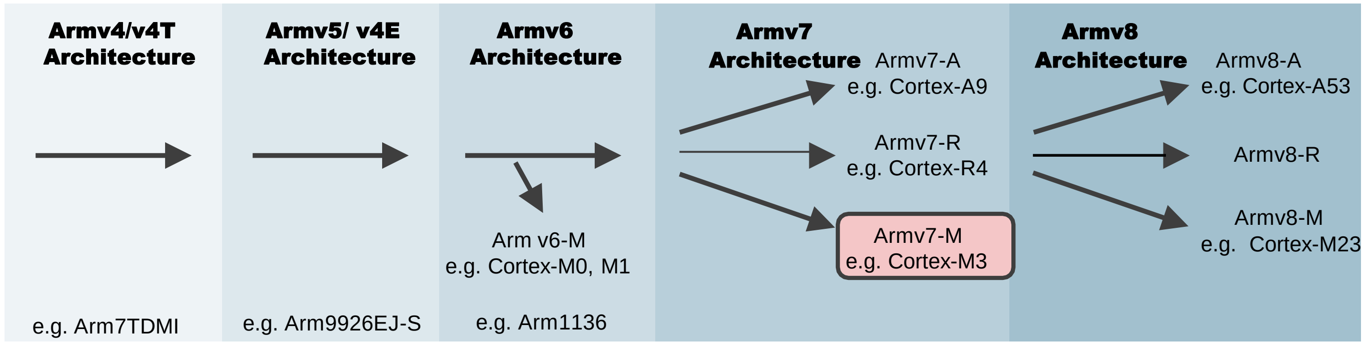


	Cortex-M0	Cortex-M3	ARM7	ARM926	ARM1026	ARM1136	ARM1176	Cortex-A8	Cortex-A9 Dual-core
Max Freq (MHz)	50	150	184	470	540	610	750	1100	2000
Min Power (mW/MHz)	0.012	0.06	0.35	0.235	0.36	0.335	0.568	0.43	0.5

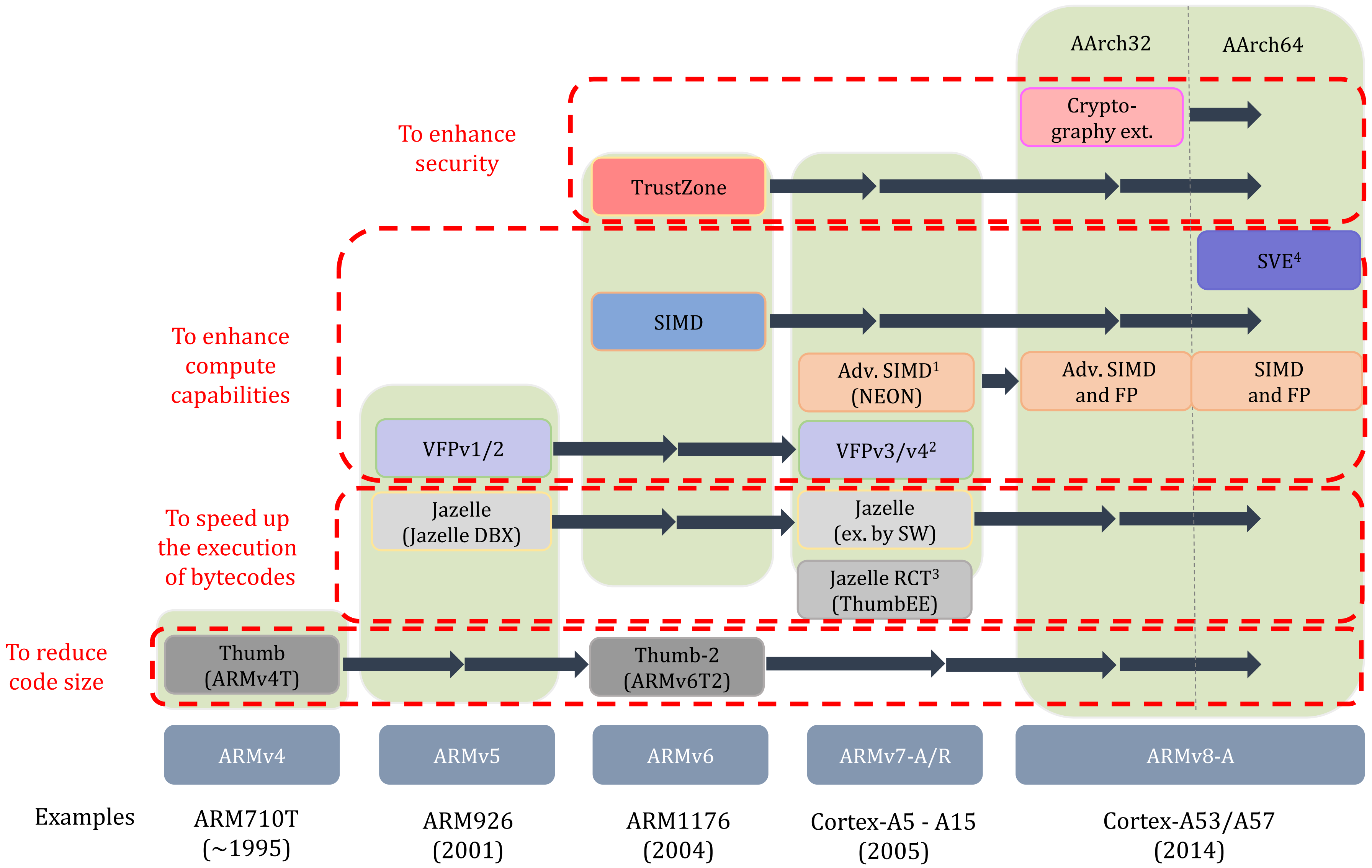
*Represents attainable speeds in 130, 90, 65, or 45nm processes

ARM processors vs. ARM architectures

- 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

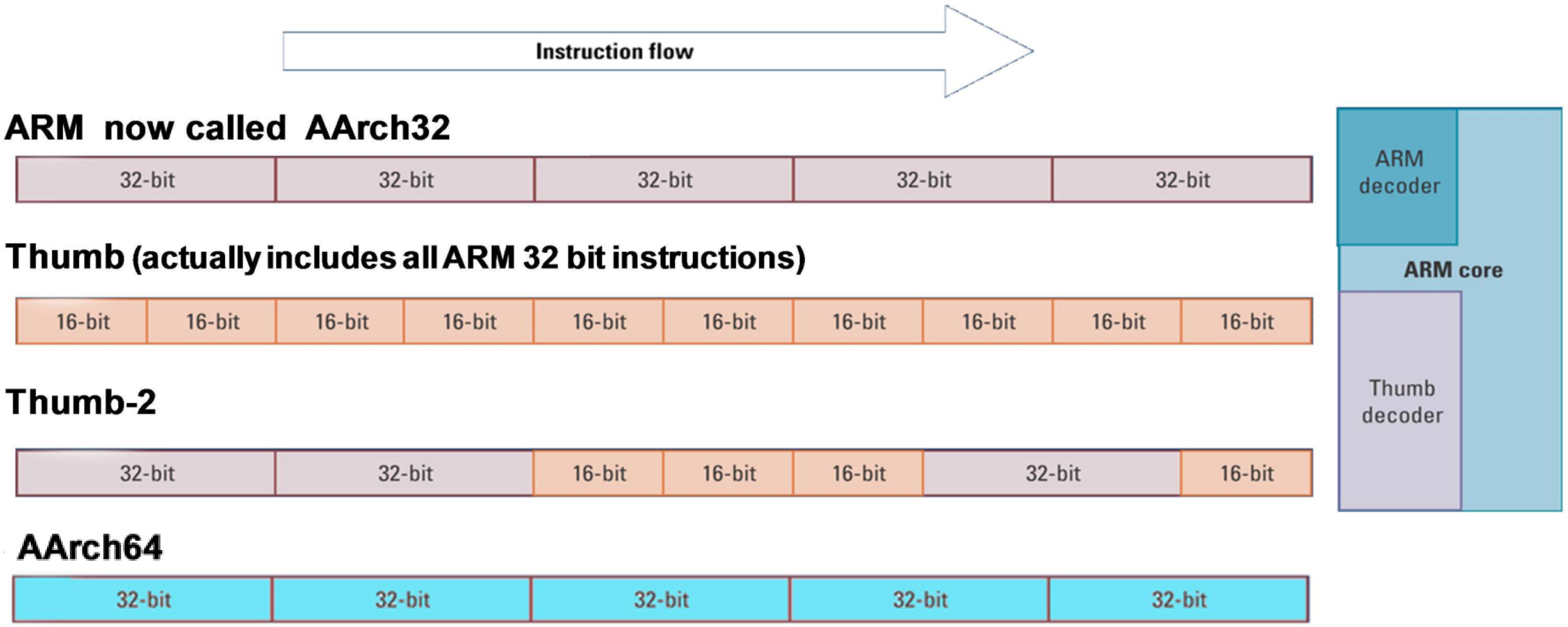


Functional Evolutions by Architectures



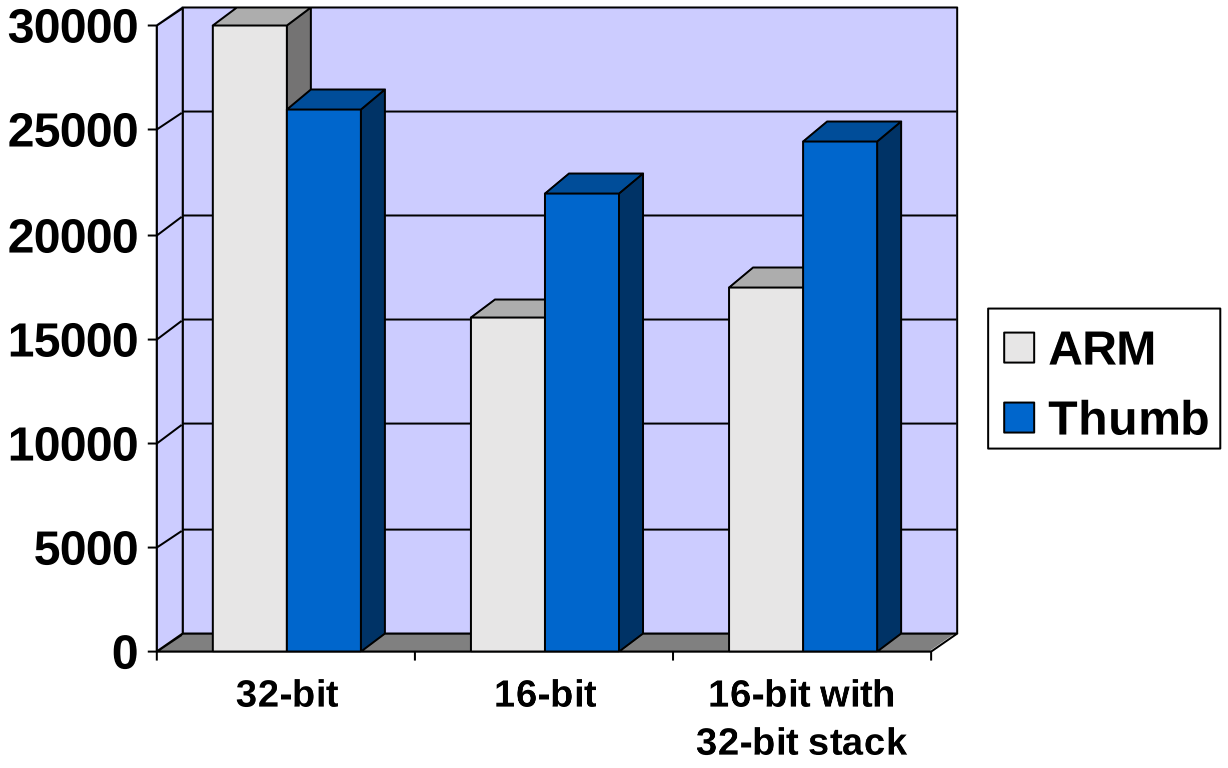
Remarks: See on the next slide.

Instruction Sets



ARM and Thumb Performance

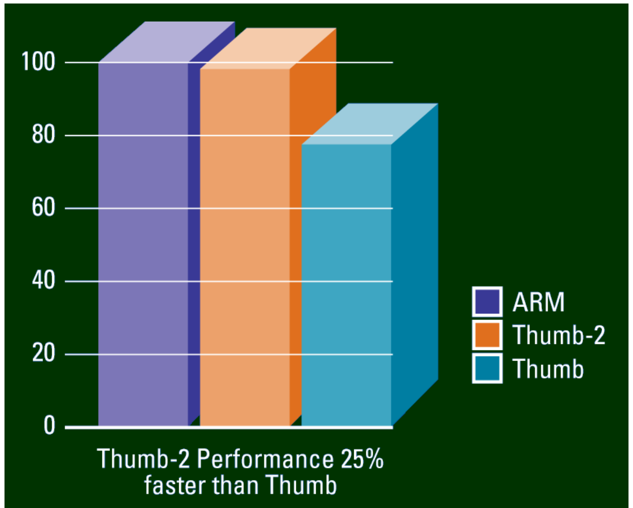
Dhrystone 2.1/sec
@ 20MHz

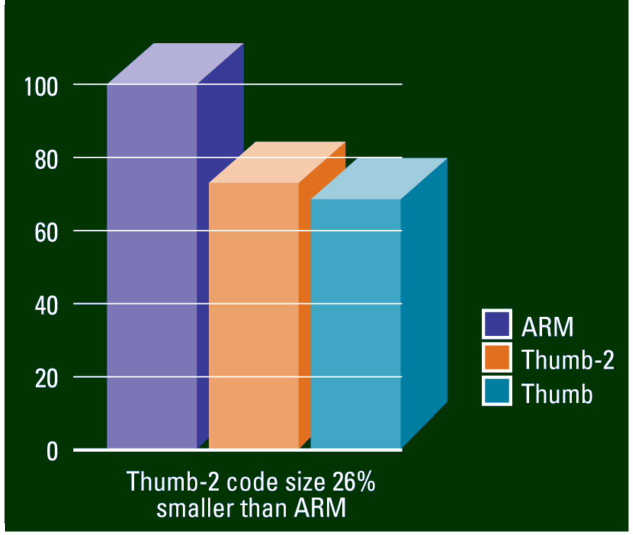


Memory width (zero wait state)

The Thumb-2 instruction set

- Variable-length instructions
  - ARM instructions are a fixed length of 32 bits
  - Thumb instructions are a fixed length of 16 bits
  - Thumb-2 instructions can be either 16-bit or 32-bit
- Thumb-2 gives approximately 25% improvement in performance over Thumb
- Thumb-2 gives approximately 26% reduction in code size over ARM





Cortex-M Processor Overview

Cortex-M Processor Family

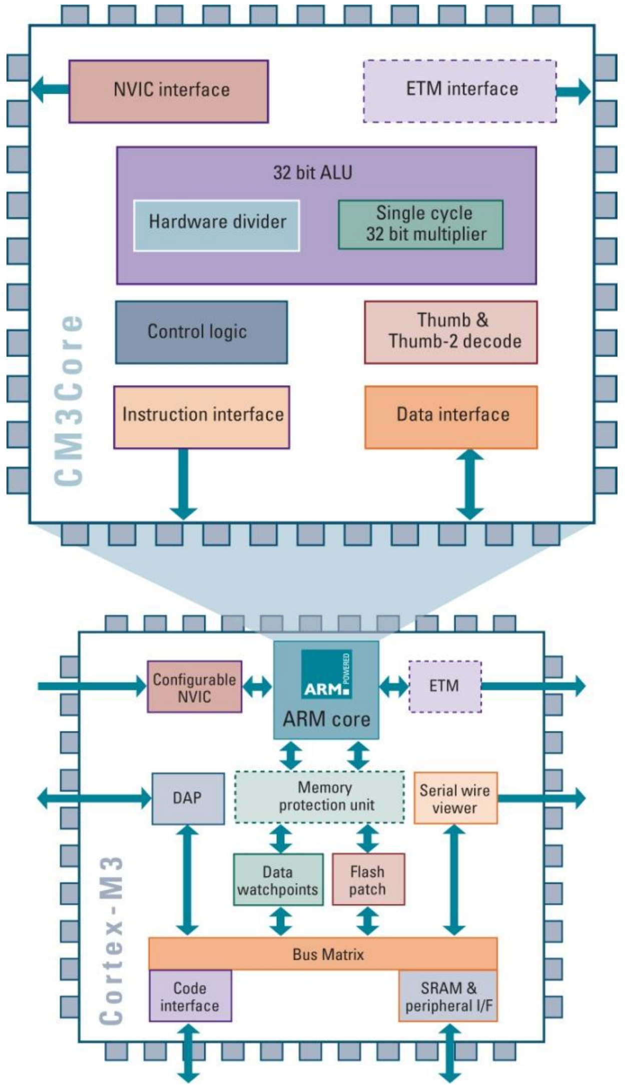
Processor	Descriptions
Cortex-M0	**A very small processor** (starting from 12K gates) for low cost, ultra low power microcontrollers and deeply embedded applications
Cortex-M0+	**The most energy-efficient processor** for small embedded system. Similar size and programmer's model to the Cortex-M0 processor, 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)
Cortex-M23	A small processor for ultra-low power and low cost designs, similar to the **Cortex-M0+** processor, but with various enhancements in instruction set and system-level features. It also supports the **TrustZone security extension**.
Cortex-M33	A mainstream processor design, similar to previous **Cortex-M3 and Cortex-M4 processors**, but with much better flexibility in system design, better energy efficiency and higher performance. It also supports the **TrustZone security extension**.

ARM Cortex-M series family

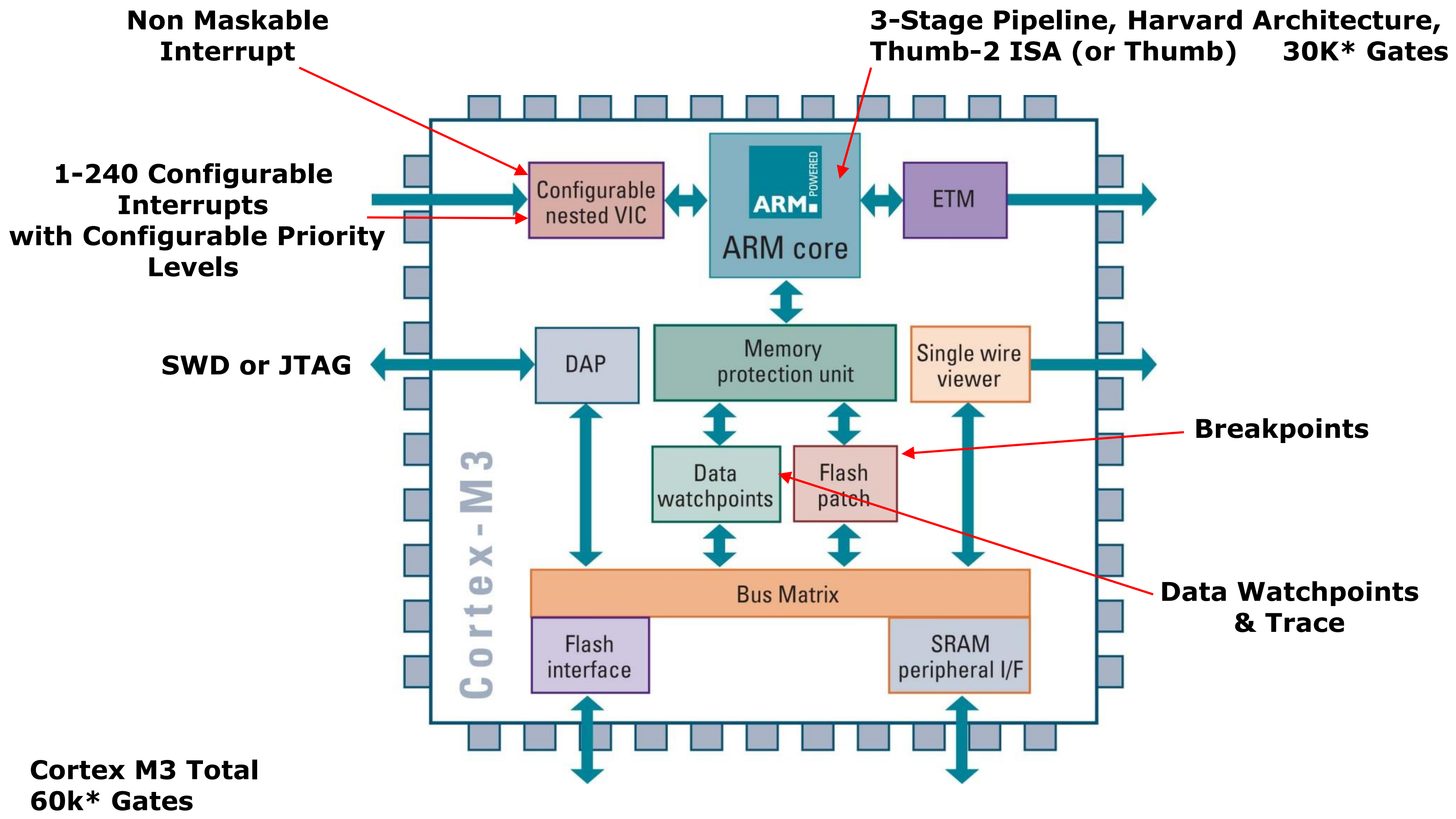
Processor	Arm Architecture	Core Architecture	Thumb®	Thumb®-2	Hardware Multiply	Hardware Divide	Saturated Math	DSP Extensions	Floating Point
Cortex-M0	Armv6-M	Von Neumann	Most	Subset	1 or 32 cycle	No	No	No	No
Cortex-M0+	Armv6-M	Von Neumann	Most	Subset	1 or 32 cycle	No	No	No	No
Cortex-M3	Armv7-M	Harvard	Entire	Entire	1 cycle	Yes	Yes	No	No
Cortex-M4	Armv7E-M	Harvard	Entire	Entire	1 cycle	Yes	Yes	Yes	Optional
Cortex-M7	Armv7E-M	Harvard	Entire	Entire	1 cycle	Yes	Yes	Yes	Optional
Cortex-M23, 33	Armv8-M	Harvard	Entire	Entire	1 cycle	Yes	Yes	Yes	Optional

Cortex-M3 Processor(1/2)

- Hierarchical processor integrating core and advanced system peripherals
- Cortex-M3 Processor (By chip manufacturers)
  - Cortex-M3 core
  - Configurable interrupt controller
    - NVIC: Nested Vectored Interrupt Controller
  - Bus matrix (ICode, Dcode, System Bus)
  - Advanced debug components(ETM…)
  - Optional MPU
- Cortex-M3 core (By ARM)
  - Harvard architecture
  - 3-stage pipeline prediction
  - Thumb®-2
  - ALU w. H/W divide and single cycle multiply



Cortex-M3 Processor(2/2)



Cortex M3 Total
60k* Gates

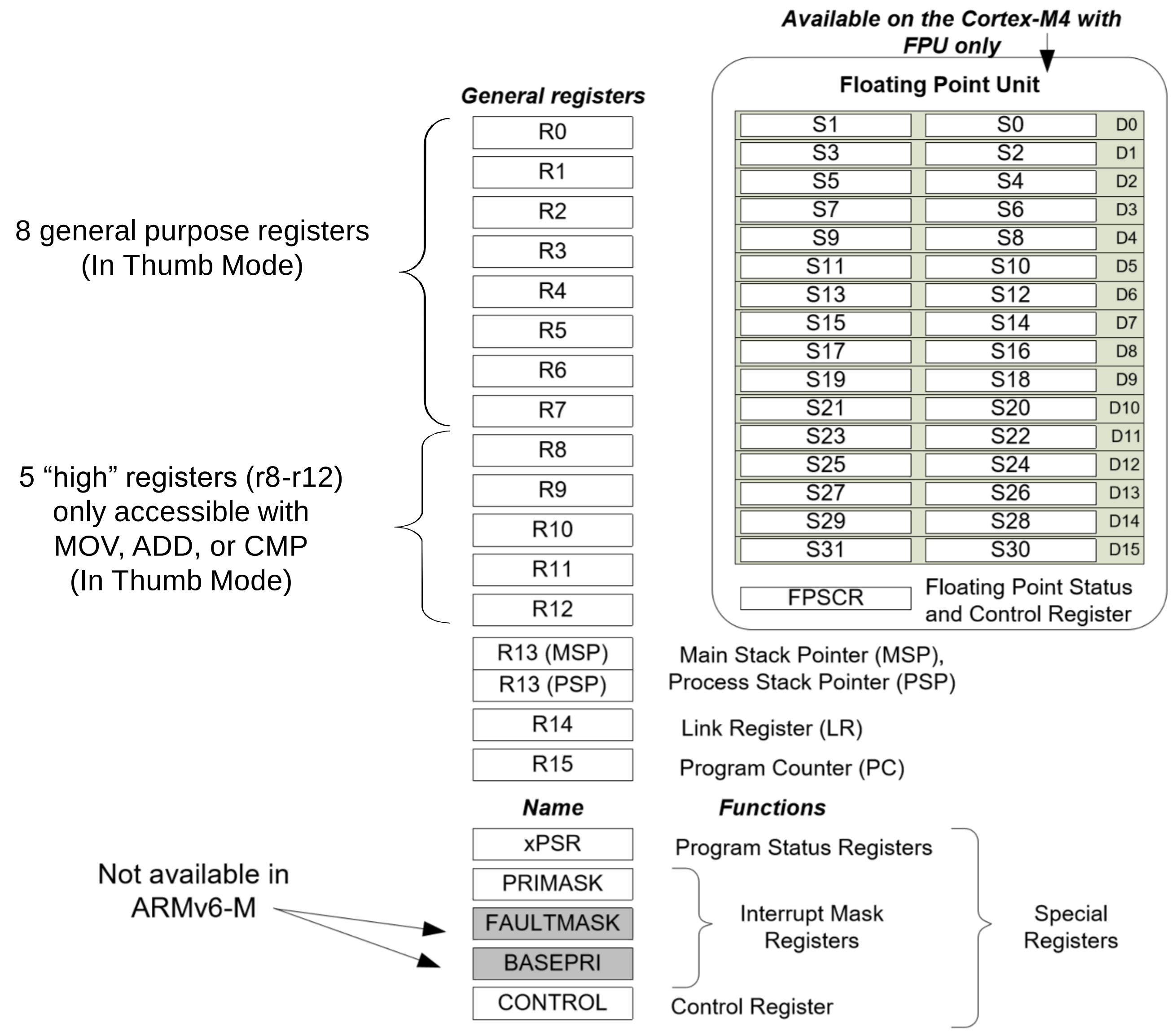
* Preliminary gate counts & power consumption based on initial implementation
Gate Counts are based on TSMC 0.18 at 50MHz
Optional ETM & MPU gate counts not included

Programmer’s Model

Cortex-M Programmer’s Model

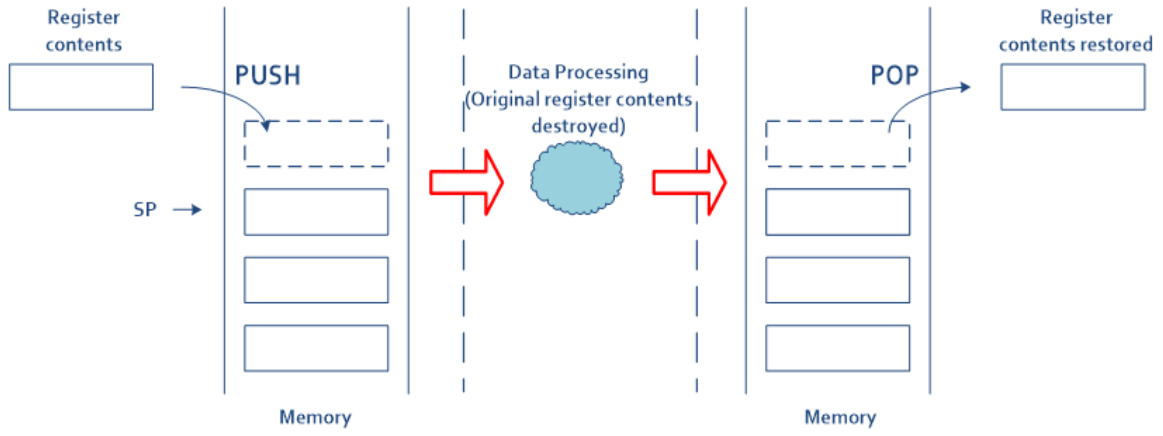
- Fully programmable in C
  - Assembly is not mandatory.
- Stack-based exception model
  - All corruptible registers automatically pushed onto stack upon exceptions in hardware
- Thumb & Thumb-2 instructions sets
- Only two processor modes
  - Thread Mode for User tasks
  - Handler Mode for exceptions
- Vector table contains addresses of exception handlers

Registers of Cortex-M Processors



Stack Pointers

- R13 is the stack pointer.


Register contents
PUSH
SP
Memory
Data Processing (Original register contents destroyed)
POP
Register contents restored
Memory

- Two stack pointers are banked so that only one is visible at a time.
- The two stack pointers are:
  - Main Stack Pointer (MSP) : This is the default stack pointer, used by the OS kernel, exception handlers, and privileged-mode programs
  - Process Stack Pointer (PSP) : Used by the user application code

Program Counter

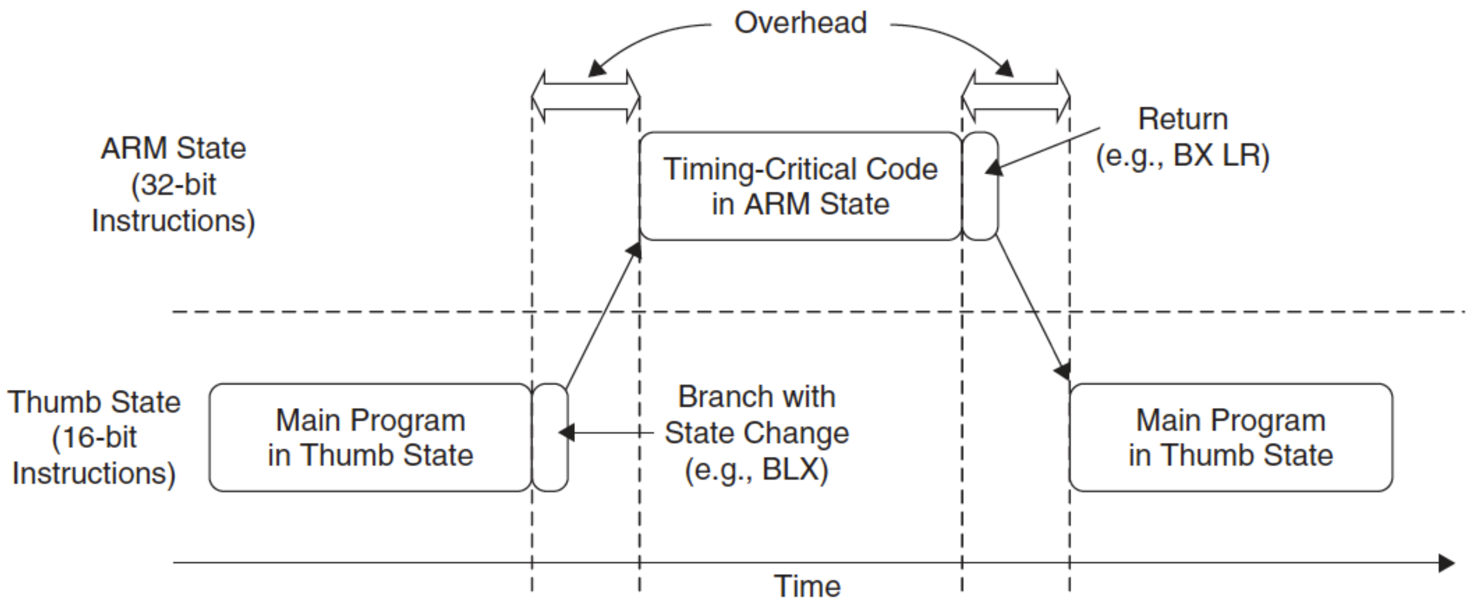
- R15 is the program counter (PC).
- When you read this register you will find that the value is different than the location of the executing instruction, due to the pipelining of the processor
  - Example:

```
0x1000 :    MOV  R0, PC       ; R0 = 0x1004
```

- When writing to the PC, it will cause a branch.
  - the LSB (Least Significant Bit) must be set to 1 to indicate the Thumb state operations (setting to 0 implies to switch to the ARM state, which will result in a fault exception in Cortex-M3)
- When reading the PC, the LSB (bit 0) is always 0.
  - Instructions are always aligned in 16-bit or 32-bit

Instruction State Switches

- The current instruction state (between ARM and Thumb/Thumb-2) is determined depending on the LSB (Least Significant bit) of the branch address. (branches: function calls, jumps, …)
  - LSB = 0 → ARM instruction set
  - LSB = 1 → Thumb/Thumb-2 instruction set


Overhead
ARM State (32-bit Instructions)
Timing-Critical Code in ARM State
Return (e.g., BX LR)
Thumb State (16-bit Instructions)
Main Program in Thumb State
Branch with State Change (e.g., BLX)
Main Program in Thumb State
Time

Link Register

- R14 is the link register (LR).
- LR is used to store the return program counter when a subroutine or function is called.
- e.g., when using the BL (Branch with Link) instruction:

```
main:                       ; main() begins here
        ...
        BL  foo             ; Call foo using Branch with Link instruction:
                            ;     PC = the address of foo()
                            ;     LR = the next instruction
        ...
foo:                        ; foo() begins here^^
        ...
        BX  LR              ; Return (i.e., LR → PC)%
```

Special Registers

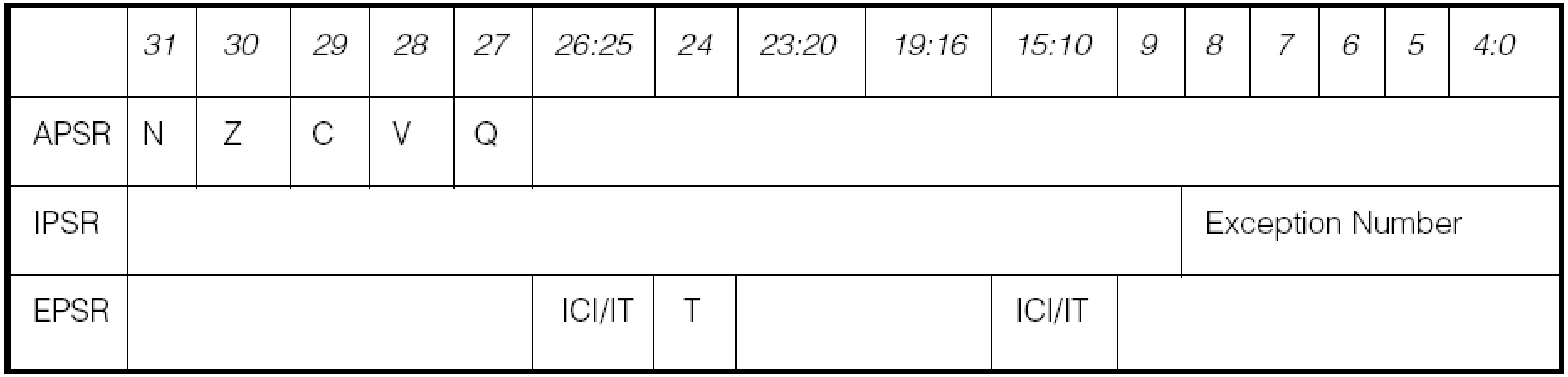
- The special registers in the Cortex-M3 processor include:
  - Program Status Registers (PSRs)
  - Interrupt Mask Registers (PRIMASK, FAULTMASK, and BASEPRI)
  - Control Register (CONTROL)
- Can only be accessed via MSR and MRS instructions
  - E.g.,

```
MRS <reg>, <special_reg>     ; Read special register
MSR <special_reg>, <reg>     ; write to special register
```

  - Note: MSR and MRS cannot have memory addresses, only registers are allowed

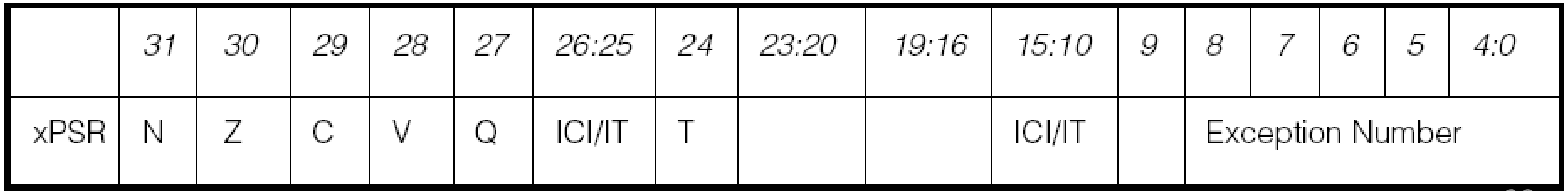
Program Status Registers (PSRs)

- The program status registers are subdivided into three status registers:
- Application/Interrupt/Execution PSR (A/I/EPSR)



	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception Number				
EPSR						ICI/IT	T			ICI/IT						

- When they are accessed as a collective item, the name xPSR is used (PSR used in program codes).



	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number				

Program Status Registers (PSRs)

- EPSR and IPSR are read-only:

```
MRS    r0, APSR           ; Read Flag state into R0
MRS    r0, IPSR           ; Read Exception/Interrupt state
MRS    r0, EPSR           ; Read Execution state
MSR    APSR, r0           ; Write Flag state
```

- Accessing xPSR:

```
MRS    r0, PSR        ; Read the combined program status word
MSR    PSR, r0        ; Write combined program state word
```

Program Status Registers (PSRs)

Bit	Description
N	Negative
Z	Zero
C	Carry/borrow
V	Overflow
Q	Sticky saturation flag
ICI/IT	Interrupt-Continuable Instruction (ICI) bits IF-THEN instruction status bit
T	Thumb state, always 1; trying to clear this bit will cause a fault exception
Exception Number	Indicates which exception the processor is handling

Bit Fields in Cortex-M3 Program Status Registers

PRIMASK, FAULTMASK and BASEPRI Registers

- These registers are used to disable exceptions.
  - NMI: Non-Maskable Interrupt

Register Name	Description
PRIMASK	A 1-bit register. When this is set, it allows NMI and the hard fault exception; all other interrupts and exceptions are masked; default is 0 (no masking)
FAULTMASK	A 1-bit register. When this is set, it allows only the NMI, and all interrupts and fault handling exceptions are disabled; default is 0
BASEPRI	A register of up to 9 bits. It defines the masking priority level. When this is set, it disables all interrupts of the same or lower level (larger priority value); default is 0

Cortex-M3 Interrupt Mask Registers

PRIMASK, FAULTMASK and BASEPRI Registers

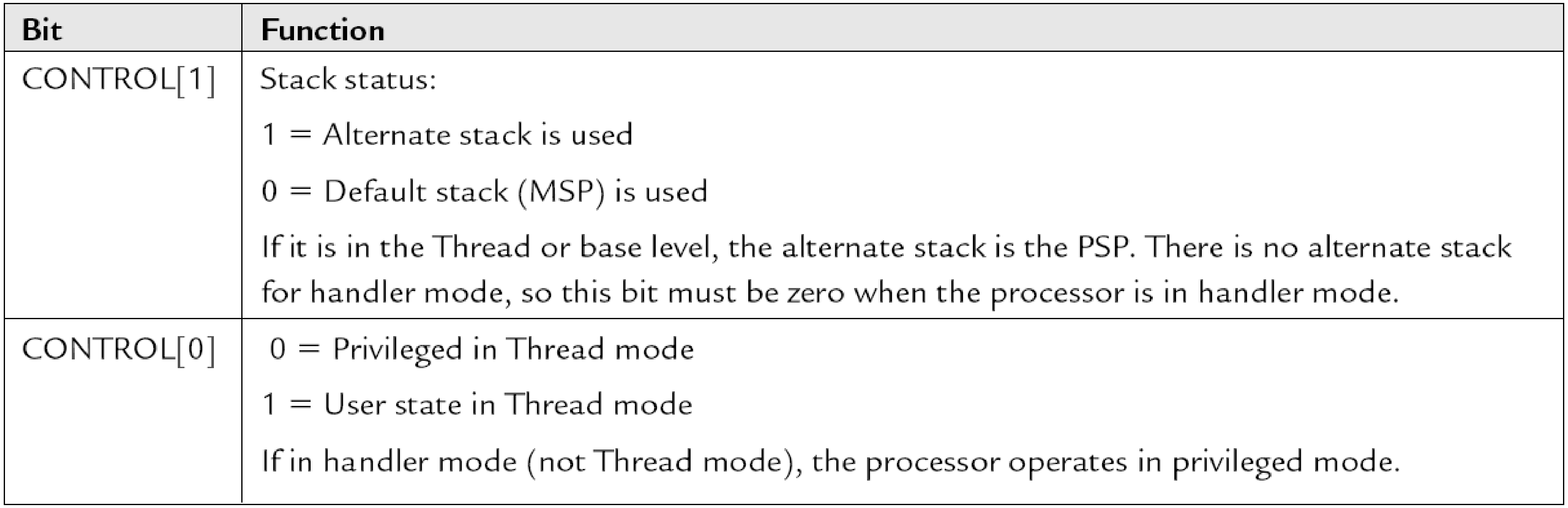
- To access the PRIMASK, FAULTMASK, and BASEPRI registers, the MRS and MSR instructions are used.
- Example:

```
MRS    r0, BASEPRI       ; Read BASEPRI register into R0
MRS    r0, PRIMASK       ; Read PRIMASK register into R0
MRS    r0, FAULTMASK     ; Read FAULTMASK register into R0
MSR    BASEPRI, r0       ; Write R0 into BASEPRI register
MSR    PRIMASK, r0       ; Write R0 into PRIMASK register
MSR    FAULTMASK, r0     ; Write R0 into FAULTMASK register
```

- PRIMASK and BASEPRI are useful for temporarily disabling interrupts in timing-critical tasks
- FAULTMASK is used by the OS kernel which cleans up a crashed task
- The PRIMASK, FAULTMASK, and BASEPRI registers cannot be set in the user access level.

The Control Register

- The Control register is used to define the privilege level and the stack pointer selection. This register has two bits.



Bit	Function
CONTROL[1]	Stack status: 1 = Alternate stack is used 0 = Default stack (MSP) is used If it is in the Thread or base level, the alternate stack is the PSP. There is no alternate stack for handler mode, so this bit must be zero when the processor is in handler mode.
CONTROL[0]	0 = Privileged in Thread mode 1 = User state in Thread mode If in handler mode (not Thread mode), the processor operates in privileged mode.

The Control Register

- CONTROL[1]
  - In Cortex-M3, the CONTROL[1] bit is always 0 (MSP) in handler mode.
  - However, in the Thread mode, it can be either 0 or 1.
  - This bit is writable only when the core is in Thread mode and privileged.
- CONTROL[0]
  - The CONTRL[0] bit is writable only in privileged level.

- To access the Control register, the MRS and MSR instructions are used:

```
MRS    r0, CONTROL        ; Read CONTROL register into R0
MSR    CONTROL, r0        ; Write R0 into CONTROL register
```

Operation Mode

- Two modes and two privilege levels.
- The operation modes determine whether the processor is running a normal program or running an exception handler.

	Privileged Level	*Unprivileged(User) Level*
When running an exception	*Handler Mode*	
When running main program	*Thread Mode*	*Thread Mode*

Operation Modes and Privilege Levels in Cortex-M3

Processor Mode

- Handler Mode
  - Used to handle exceptions.
  - The processor returns to Thread mode when it has finished exception processing.
  - In Handler mode, software execution is always privileged.
- 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.

Privilege Levels

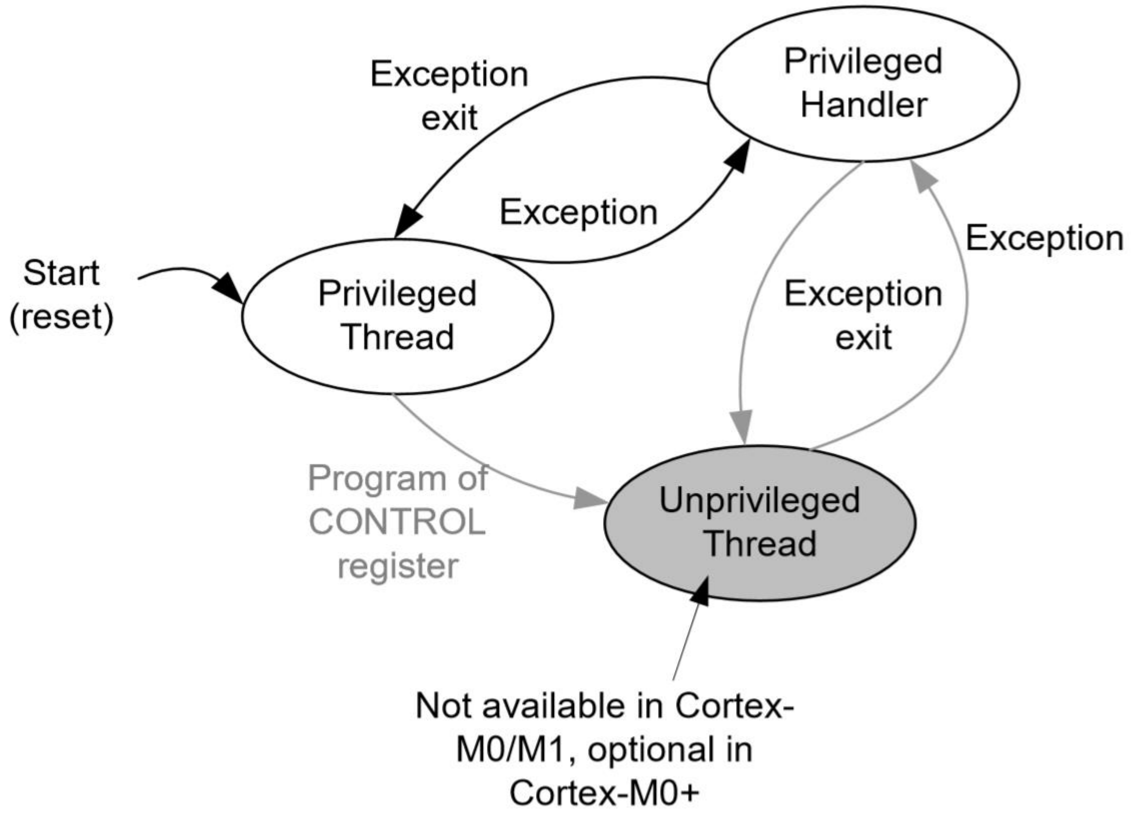
- Unprivileged (User)

  The software:

  - has limited access to the MSR (Move Register to Special Register) and MRS (Move Special Register to Register) instructions, and cannot use the CPS (Change Processor State) 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.

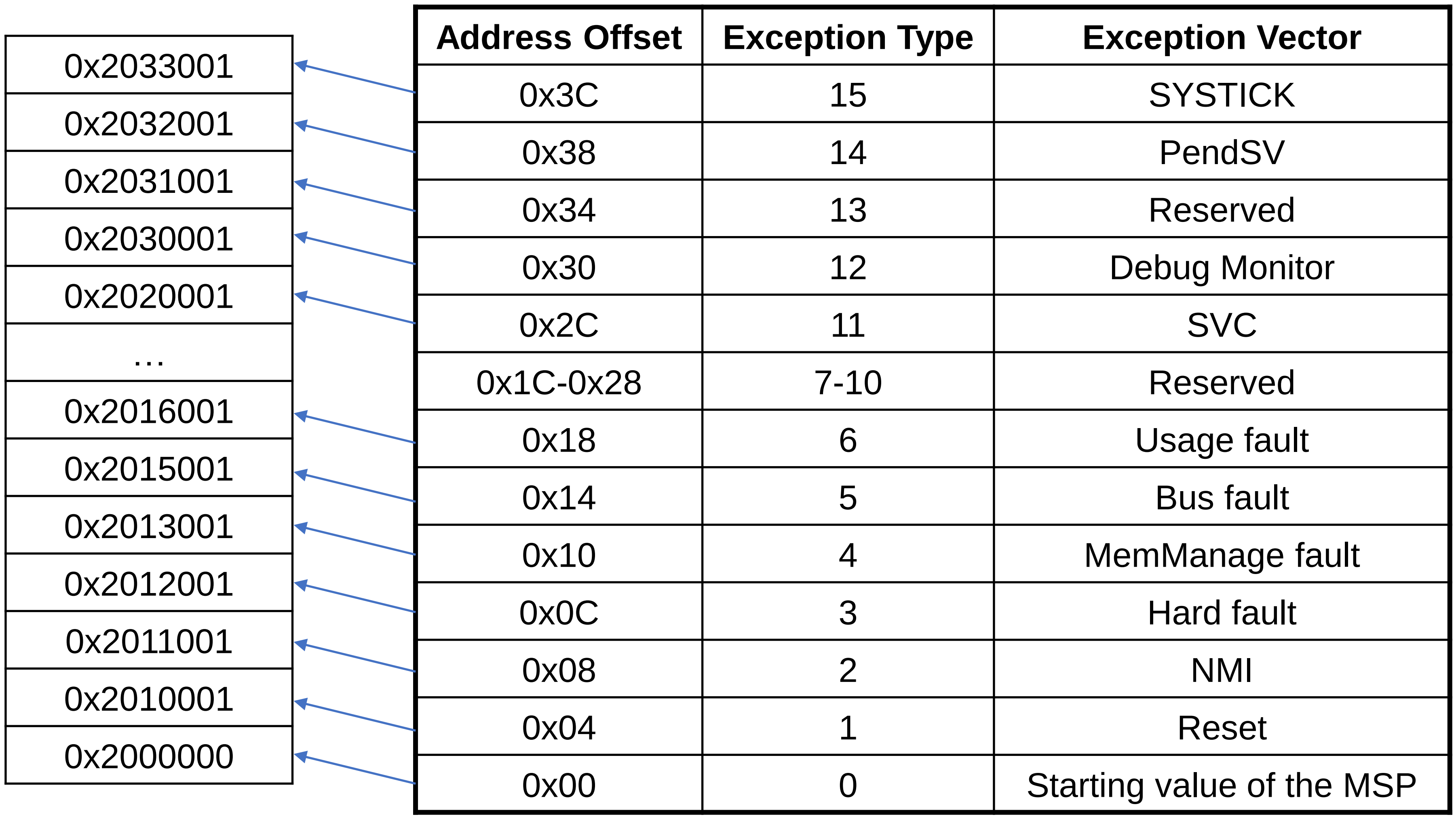
Transition of Operation Mode



Vector Tables

- The vector table is an array of word data, with each representing the starting address of the handler for one exception/interrupt type.
  - In the Cortex-M3, vector addresses in the vector table should have their LSB set to 1 to indicate that they are Thumb code.
- The base address of the vector table is relocatable (set the relocation register in the NVIC); initially, the base address is 0x0.
- Example:
  - The reset is exception type 1. The address of the reset vector is 1 times 4, which equals 0x00000004; and NMI vector (type 2) is located in 2 * 4 = 0x00000008
  - The word stored at address 0x00000000 is used as the starting value for the MSP.

Vector Tables

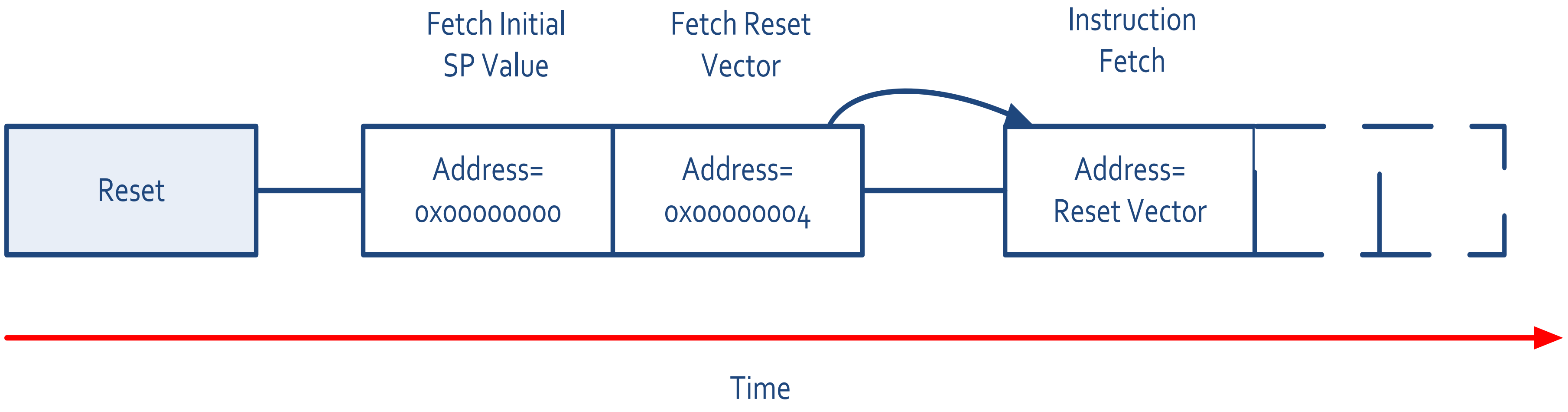


Address Offset	Exception Type	Exception Vector
0x3C	15	SYSTICK
0x38	14	PendSV
0x34	13	Reserved
0x30	12	Debug Monitor
0x2C	11	SVC
0x1C-0x28	7-10	Reserved
0x18	6	Usage fault
0x14	5	Bus fault
0x10	4	MemManage fault
0x0C	3	Hard fault
0x08	2	NMI
0x04	1	Reset
0x00	0	Starting value of the MSP

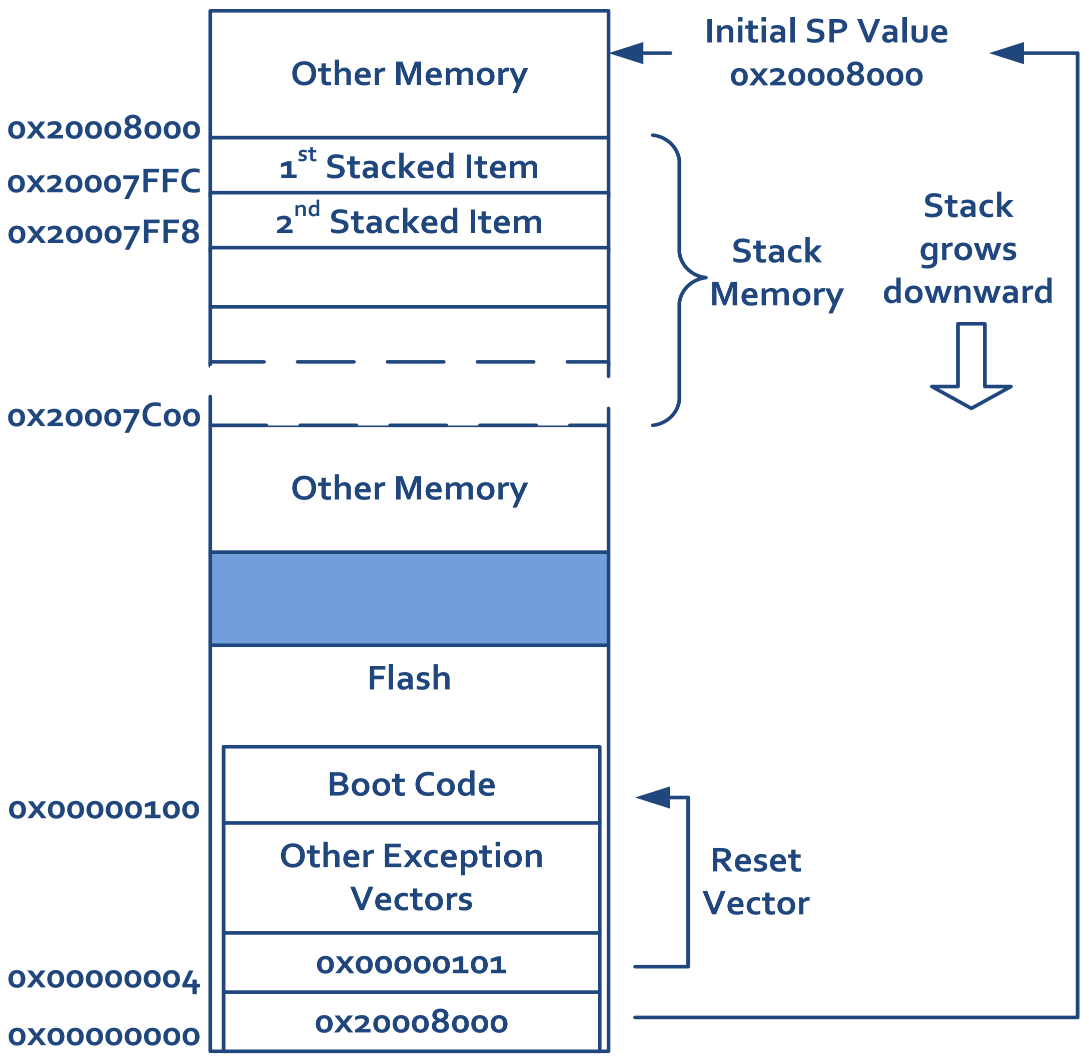
Vector Table Definition After Reset

Reset Sequence

- After the processor exits reset, it will read two words from memory:
  - Address 0x00000000: default value of R13 (MSP)
  - Address 0x00000004: Reset vector (the starting address of startup program; LSB should be set to 1 to indicate Thumb state)



Reset Sequence



Assume that the stack memory range from 0x20007C00 to 0x20007FFF

Initial Stack Pointer Value and Initial Program Counter (PC) Value Example

Summary

- ARM Architecture
  - Instruction Sets
- Cortex-M Processors
  - Programmer’s Model