The ARM Cortex-M0 Processor Architecture Part-1

Module Syllabus

- ARM Architectures and Processors
  - What is ARM Architecture
  - ARM Processors Families
  - ARM Cortex-M Series Family
  - Cortex-M0 Processor
  - ARM Processor Vs. ARM Architectures
- ARM Cortex-M0 Processor
  - Cortex-M0 Processor Overview
  - Cortex-M0 Block Diagram
  - Cortex-M0 Registers
- Cortex-M0 Memory Map
- Cortex-M0 Exception Handling

ARM Architectures and ARM Processors

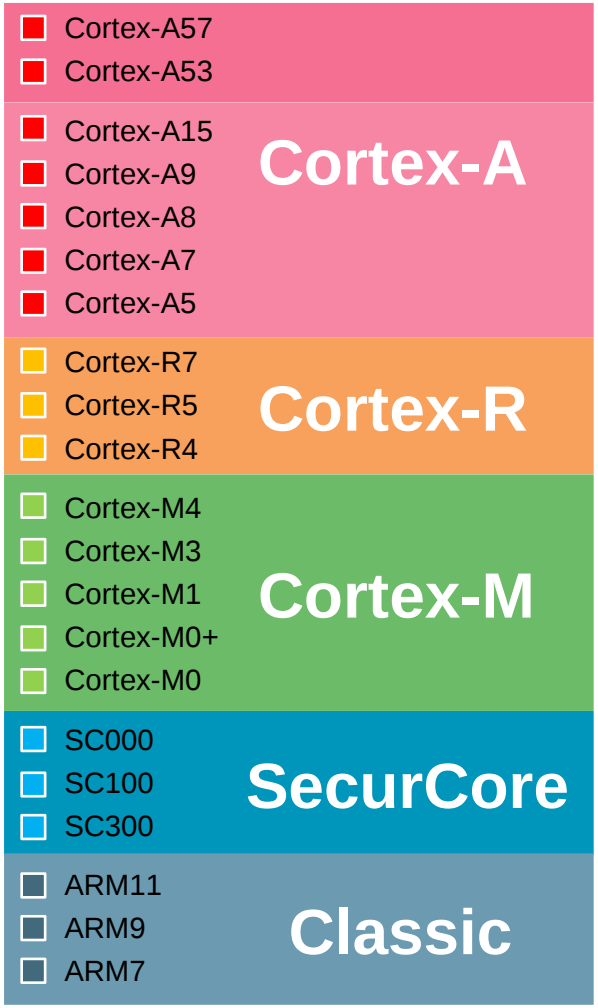
What is ARM Architecture

- ARM architecture is a family of RISC-based processor architectures
  - Well-known for its power efficiency;
  - Hence widely used in mobile devices, such as smartphones, and tablets
  - Designed and licensed to a wide eco-systems by ARM.

- ARM Holdings
  - The company designs ARM-based processors;
  - Does not manufacture, but licenses designs to semiconductor partners who fabricate and sell to their customers;
  - Also offer other designs available, such as physical IPs, graphics cores, and development tools.

ARM Processor Families

- Cortex-A series (Application)
  - High performance processors for open Operating Systems;
  - Applications include smartphones, digital TV, smart books, home gateways;
- Cortex-R series (Real-time)
  - Exceptional performance for real-time applications;
  - Applications include automotive braking systems, powertrains;
- Cortex-M series (Microcontroller)
  - Cost-sensitive solutions for deterministic microcontroller applications;
  - Applications include microcontrollers, mixed signal devices, smart sensors, automotive body electronics and airbags;
- SecurCore series
  - High security applications.
- Previous classic processors
  - Include ARM7, ARM9, ARM11 families

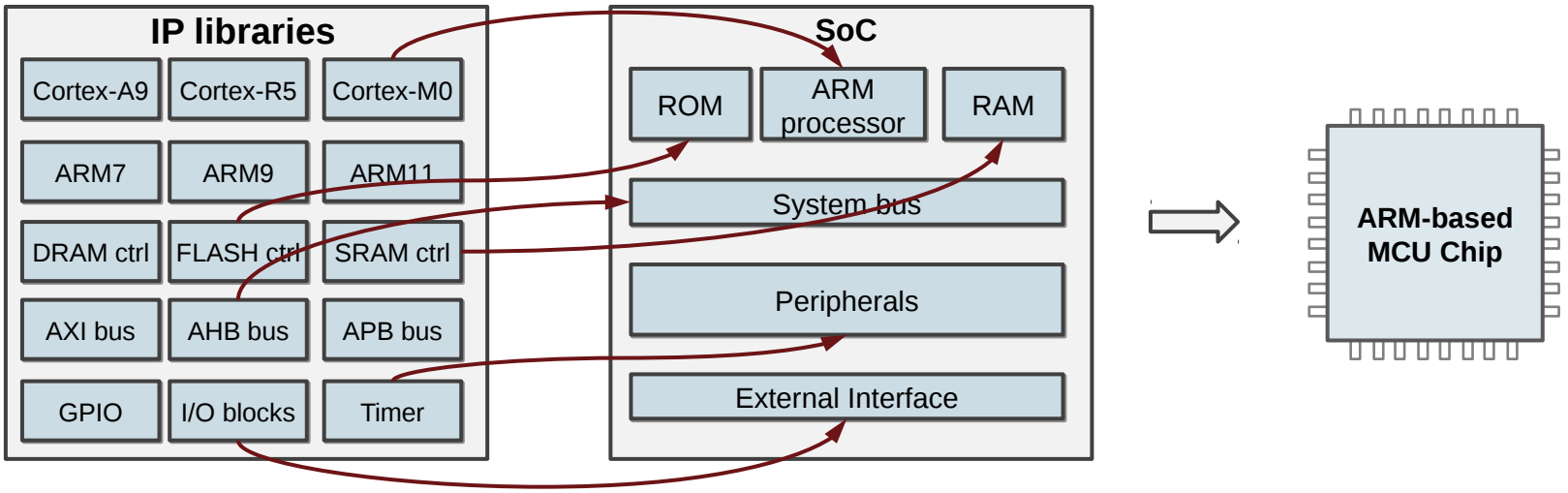


Cortex
Low-Power Leadership from ARM

As of Sept 2013

Design an ARM-based SoC

- Select a set of IP cores from ARM or other third-party IP vendors;
- Integrate IP cores into a single chip design;
- Give design to semiconductor foundries for chip fabrication.



ARM Cortex-M Series Family

- Cortex-M series: Cortex-M0, M0+, M1, M3, M4.
- Energy-efficiency
  - Lower energy costs, longer battery life
- Smaller code
  - Lower silicon costs
- Ease of use
  - Faster software development and reuse
- Embedded applications
  - Smart metering, human interface devices, automotive and industrial control systems, white goods, consumer products and medical instrumentation





As of Sept 2013

ARM Cortex-M Series Family

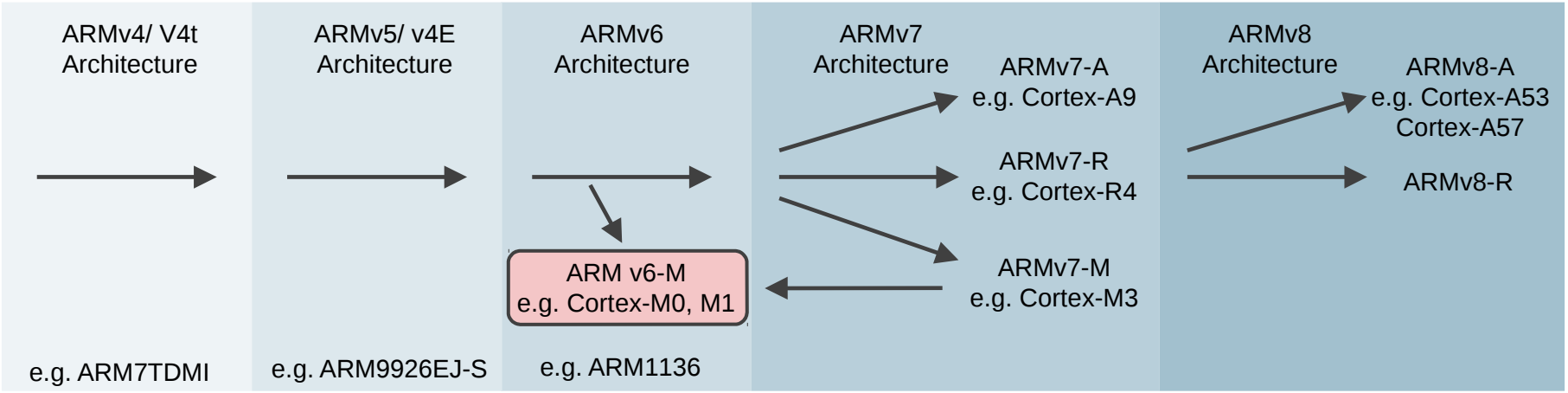
Processor	ARM Architecture	Core Architecture	Thumb®	Thumb®-2	Hardware Multiply	Hardware Divide	Saturated Math	DSP Extensions	Floating Point
Cortex-M0	ARMv6-M	Von Neumann	Most	Subset	1 or 32 cycle	No	No	No	No
Cortex-M0+	ARMv6-M	Von Neumann	Most	Subset	1 or 32 cycle	No	No	No	No
Cortex-M1	ARMv6-M	Von Neumann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

Cortex-M0 Processor

- The smallest ARM processor
  - Exceptionally small silicon area
  - Ultra-low gate count (approx. 12k gates at minimum configuration)
- High code density
  - Fundamental base of 16-bit Thumb instructions
  - Additional powerful 32-bit instructions
- Lower power
  - 16µW/MHz (90LP process, minimal configuration)
- Simplicity
  - Only 56 instructions
  - C friendly
  - More deterministic response time
- Uses ARMv6-M Architecture

ARM Processor Vs. ARM Architectures

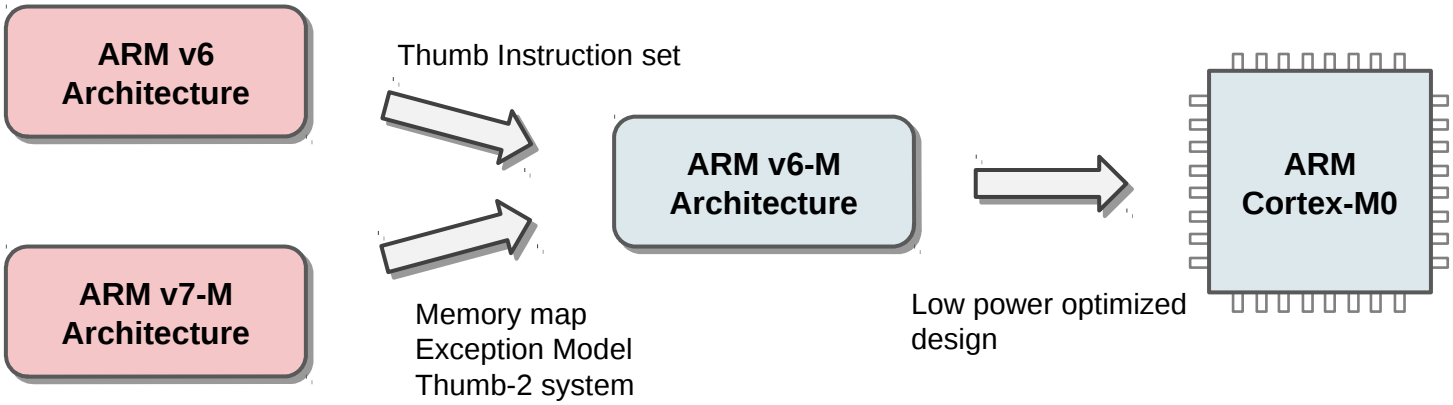
- ARM architecture
  - Describes the details of instruction set, programmer's model, exception model, and memory map;
  - Documented in the Architecture Reference Manual;
- ARM processor
  - Developed using one of the ARM architectures;
  - More implementing details, such as timing information and implementation-related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



As of Sept 2013

ARM Processor Vs. ARM Architectures

- Cortex-M0: v6-M
  - ARMv6 architecture's thumb instruction set;
  - ARMv7-M architecture memory map, exception model, and thumb-2 system;
  - Low power optimised design.

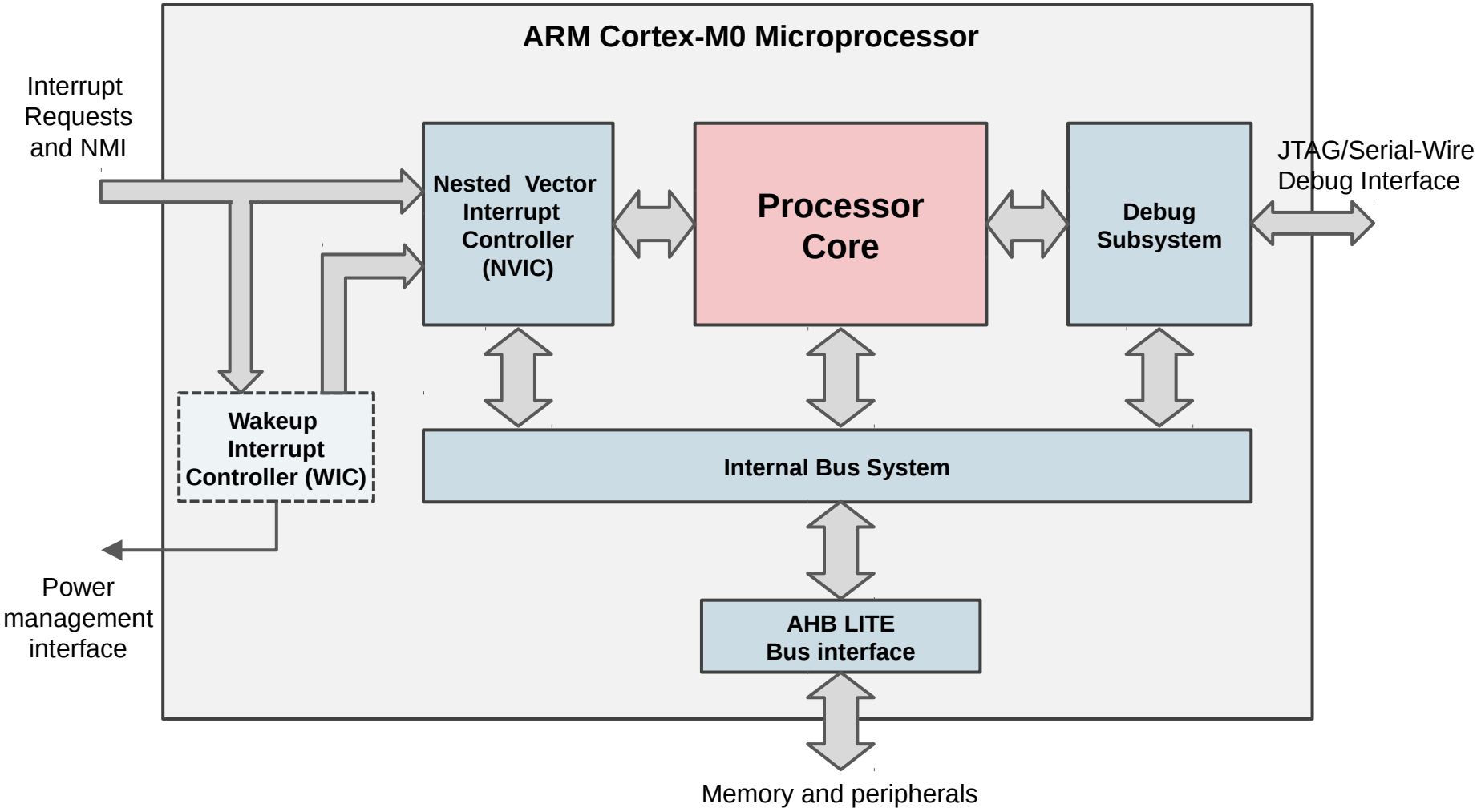


ARM Cortex-M0 Processor Overview

Cortex-M0 Overview

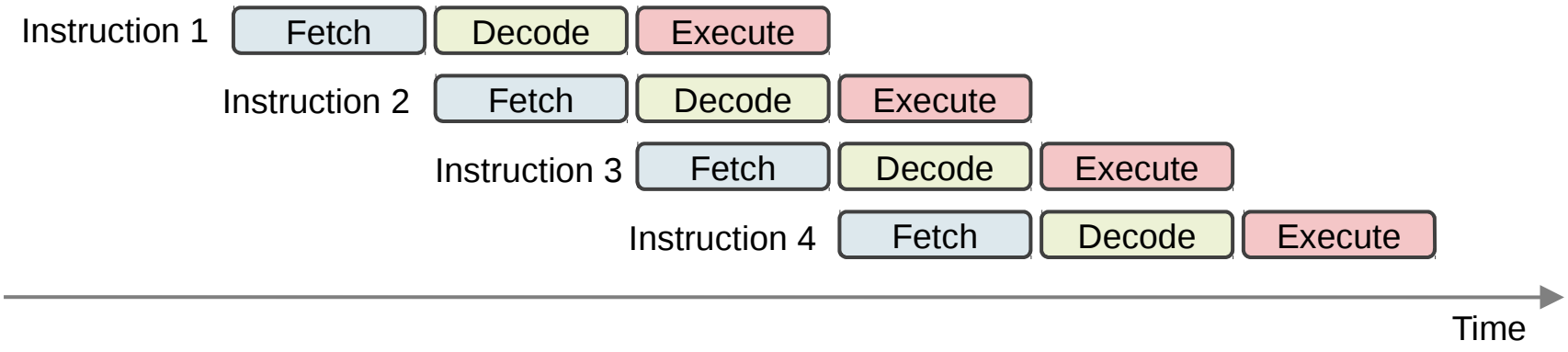
- 32-bit Reduced Instruction Set Computing (RISC) processor
- Von-Neumann architecture
  - Both data and instructions share a single bus interface;
- Instruction set
  - 56 instructions as a subset of Thumb-1 (16-bit) and Thumb-2 (16/ 32-bit);
- Supported Interrupts
  - Non-maskable Interrupt (NMI) + 1 to 32 physical interrupts
- Supports Sleep Modes

Cortex-M0 Block Diagram



Cortex-M0 Block Diagram

- Processor core
  - Contains internal registers, the ALU, data path, and some control logics;
  - Three-stage pipeline: fetch, decode, and execution;
  - Registers include sixteen 32-bit registers for both general and special usages.
- Nested Vectored Interrupt Controller (NVIC)
  - Up to 32 interrupt request signals and a non-maskable interrupt (NMI);
  - Automatically handles nested interrupts, such as comparing priorities between interrupt requests and the current priority level;



Cortex-M0 Block Diagram

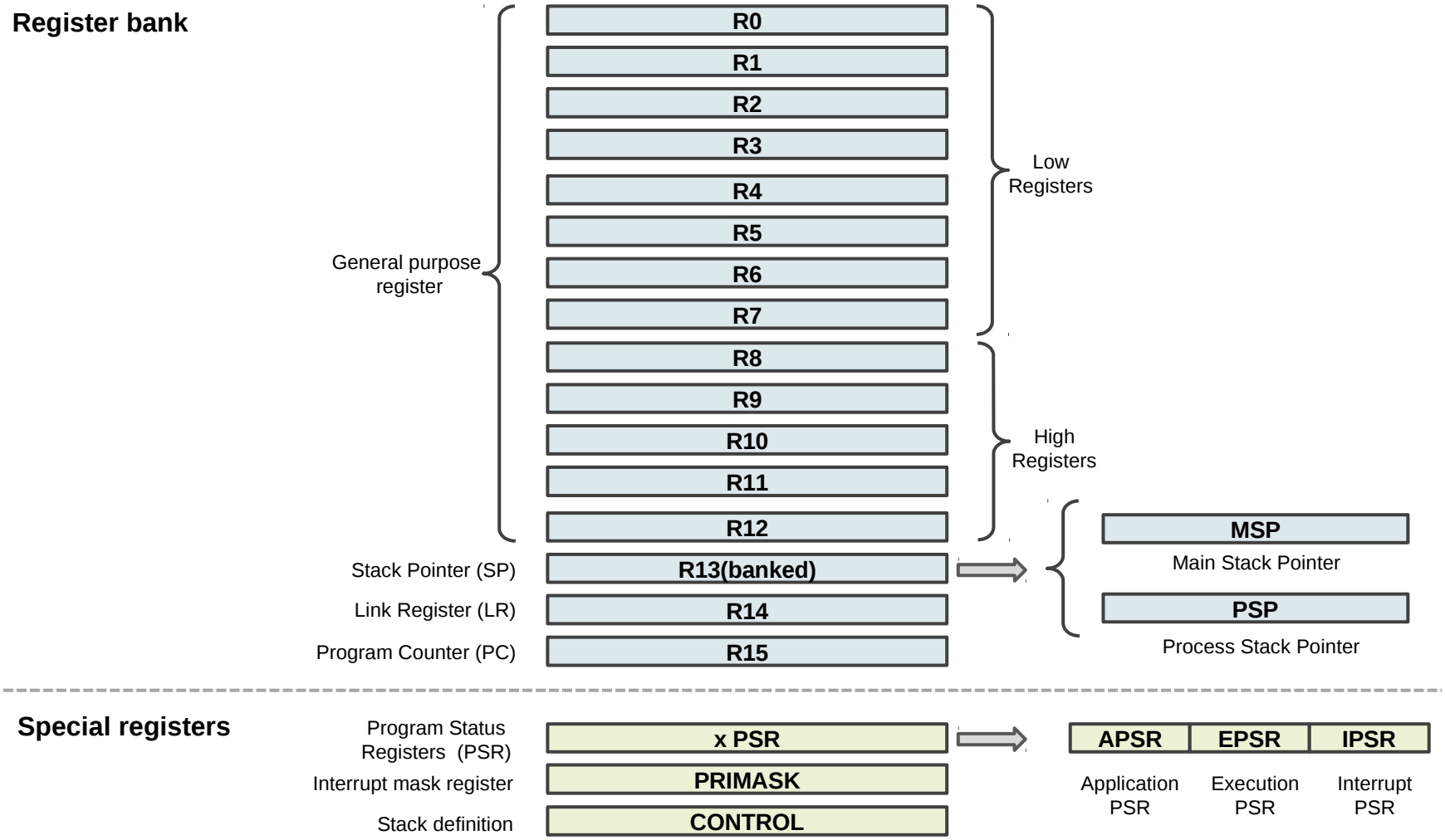
- Bus system
  - Includes the internal bus system, the data path in the processor core, and the AHB LITE interface unit;
  - All 32 bits wide;
  - AHB LITE is an on-chip bus protocol for many ARM processors and widely used in IC design industry.
- Debug subsystem
  - Handles debug control, program breakpoints, and data watchpoints;
  - When a debug event occurs, it can put the processor core in a halted state, where developers can analyse the status of the processor at that point, such as register values and flags.
- Wakeup Interrupt Controller (WIC) (optional)
  - For low-power applications, the microcontroller can enter sleep mode by shutting down most of the components.
  - When an interrupt request is detected, the WIC can inform the power management unit to power up the system.

ARM Cortex-M0 Processor Registers

Cortex-M0 Registers

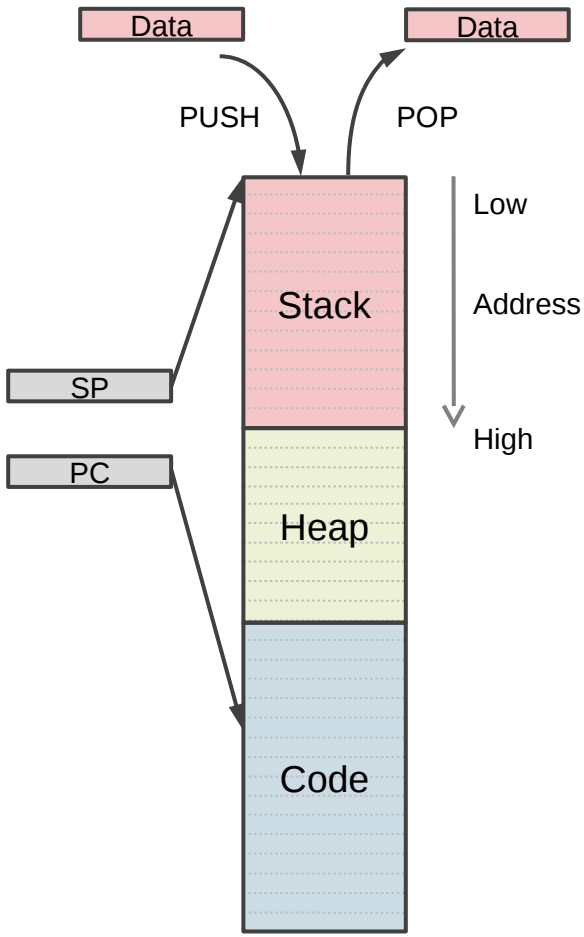
- Processor registers
  - The internal registers are used to store and process temporary data within the processor core;
  - All registers are inside the processor core hence can be accessed more quickly;
  - Load-store architecture
    - To process a data in the memory, they have to be loaded from the memory to a register, processed inside the processor, and then written back to the memory if needed;
- Cortex-M0 register
  - Register bank
    - Sixteen 32-bit registers (thirteen are used for general-purpose);
  - Special registers;

Cortex-M0 Registers



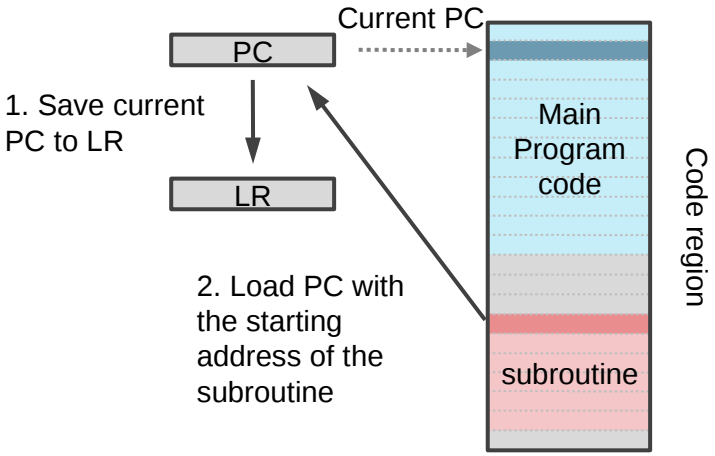
Cortex-M0 Registers

- R0 – R12: general purpose registers
  - Low registers (R0 – R7) can be accessed by any instruction;
  - High registers (R8 – R12) sometimes cannot be accessed by some Thumb instructions;
- R13: Stack Pointer (SP)
  - Records the current address of the stack
  - Used for saving the context of a program while switching between tasks
  - Cortex-M0 has two SPs: *Main SP*, used in applications that require privileged access e.g. OS kernel, and exception handlers, and *Process SP*, used in base-level application code (when not running an exception handler)
- Program Counter (PC)
  - Records the address of the current instruction code;
  - Automatically incremented by 4 at each operation (for 32-bit instruction code), except branching operations;
  - A branching operation, such as function calls, will change the PC to a specific address, meanwhile save the current PC to the Link Register (LR);

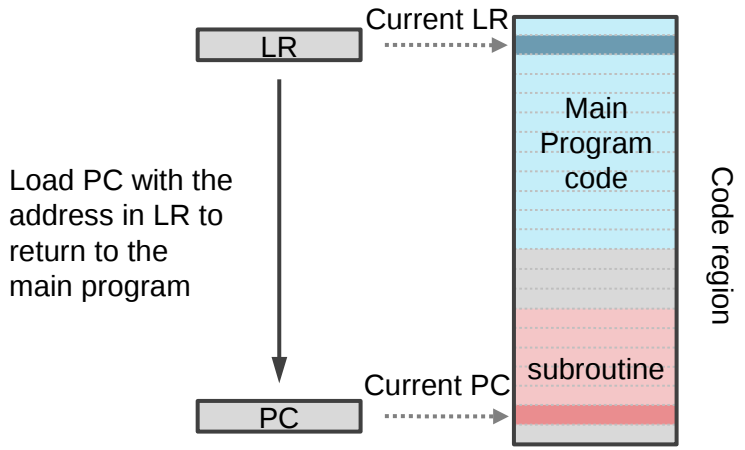


Cortex-M0 Registers

- R14: Link Register (LR)
  - The LR is used to store the return address of a subroutine or a function call;
  - The program counter (PC) will load the value from LR after a function is finished;



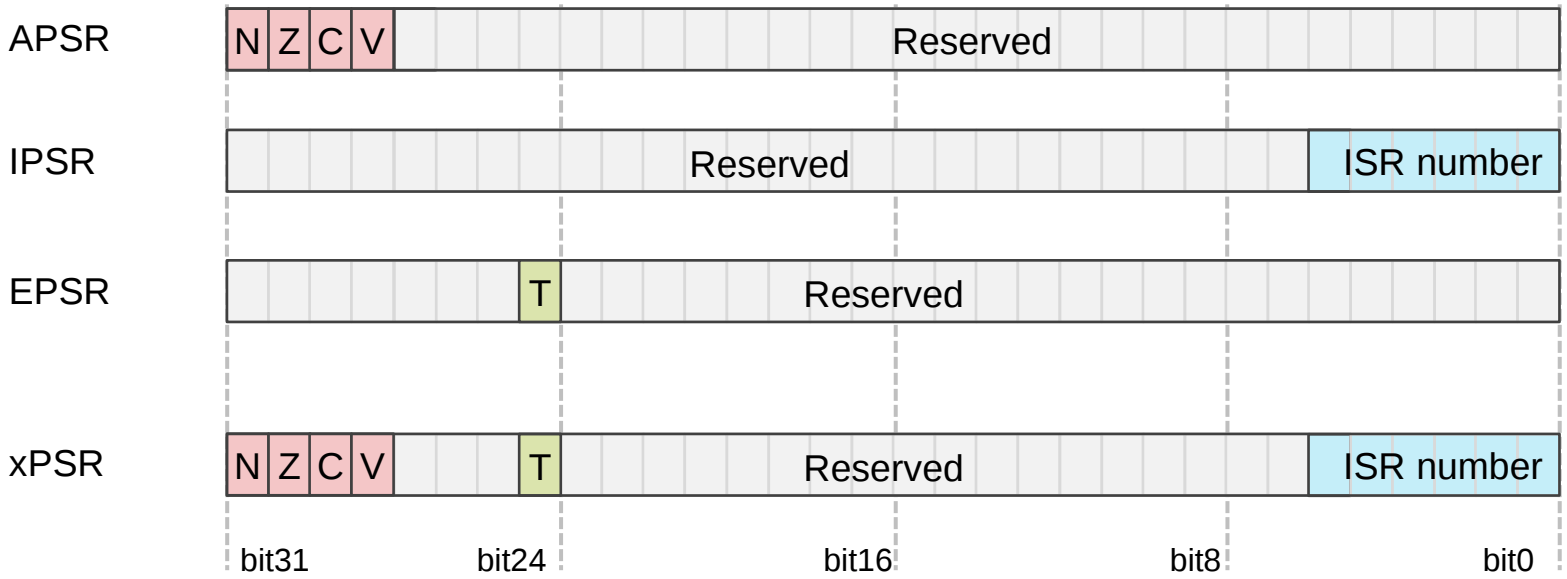
Call a subroutine



Return from a subroutine to the main program

Cortex-M0 Registers

- xPSR, combined Program Status Register
  - Provides the information about program execution and the ALU flags;
    - Application PSR (APSR)
    - Interrupt PSR (IPSR)
    - Execution PSR (EPSR)

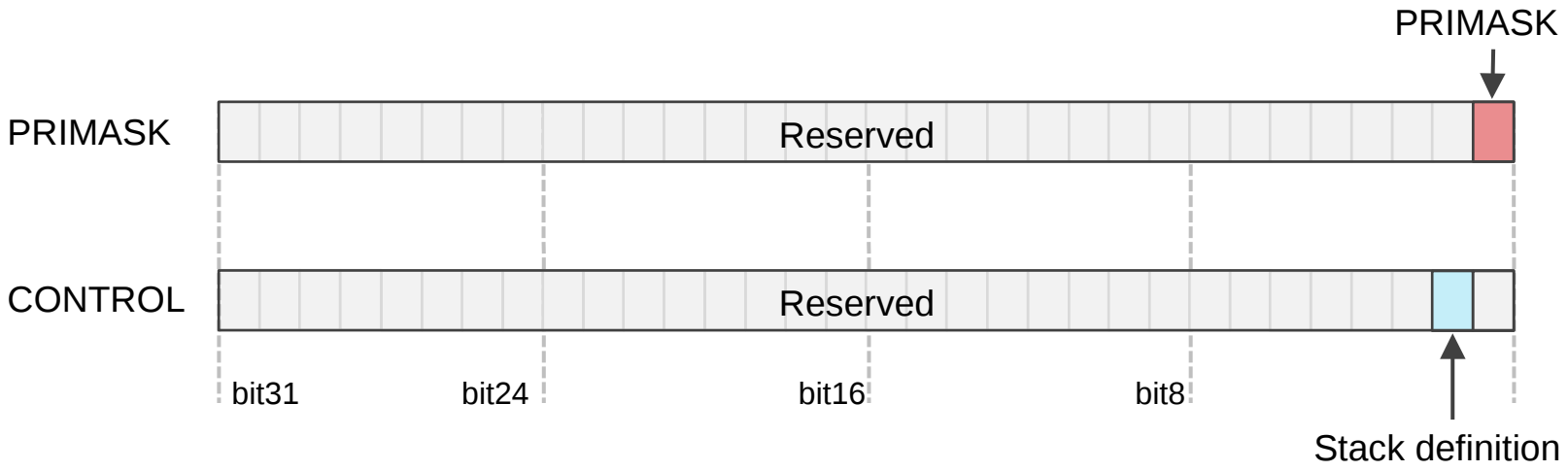


Cortex-M0 Registers

- APSR
  - N: negative flag
    - Set to one if the result from ALU is negative;
  - Z: zero flag
    - Set to one if the result from ALU is zero;
  - C: carry flag
    - Set to one if an unsigned overflow occurs;
  - V: overflow flag
    - Set to one if a signed overflow occurs;
- IPSR
  - ISR number
    - Current executing interrupt service routine number
- EPSR
  - T: Thumb state
    - Always one since Cortex-M0 only supports the Thumb state

Cortex-M0 Registers

- PRIMASK: Interrupt Mask Special Register
  - 1-bit PRIMASK
    - Set to one will block all the interrupts apart from nonmaskable interrupt (NMI) and the hard fault exception;
- CONTROL: special register
  - 1-bit stack definition
    - Set to one: use the process stack pointer (PSP);
    - Clear to zero: use the main stack pointer (MSP).



ARM Cortex-M0 Processor Memory Map

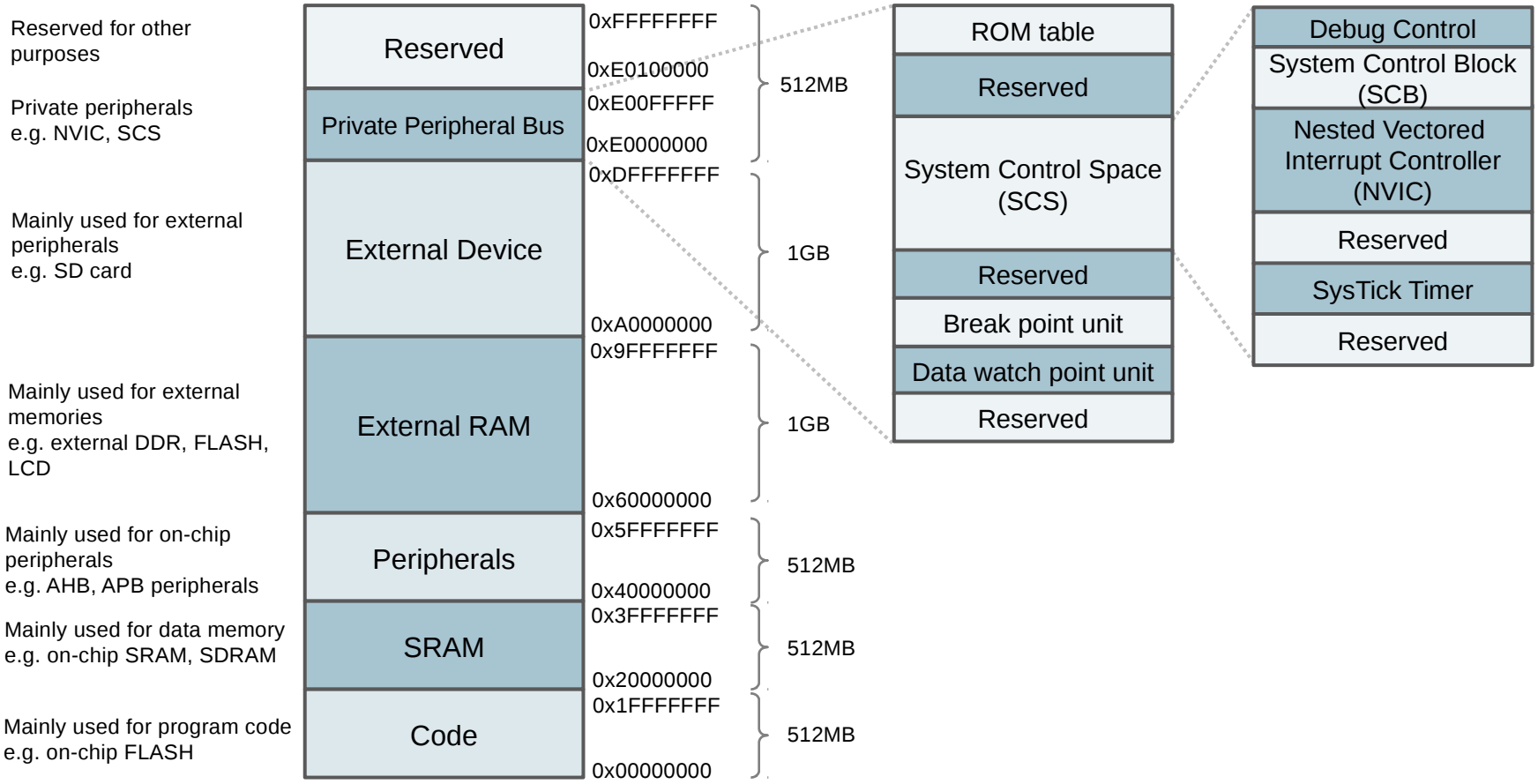
Cortex-M0 Memory Map

- The Cortex-M0 processor has 4 GB of memory address space

- The 4GB memory space is architecturally defined as a number of regions.
  - Each region is given for recommended usage;
  - Easy for software programmer to port between different devices.

- Nevertheless, despite of the default memory map, the actual usage of the memory map can also be flexibly defined by the user, except some fixed memory addresses, such as internal private peripheral bus.

Cortex-M0 Memory Map



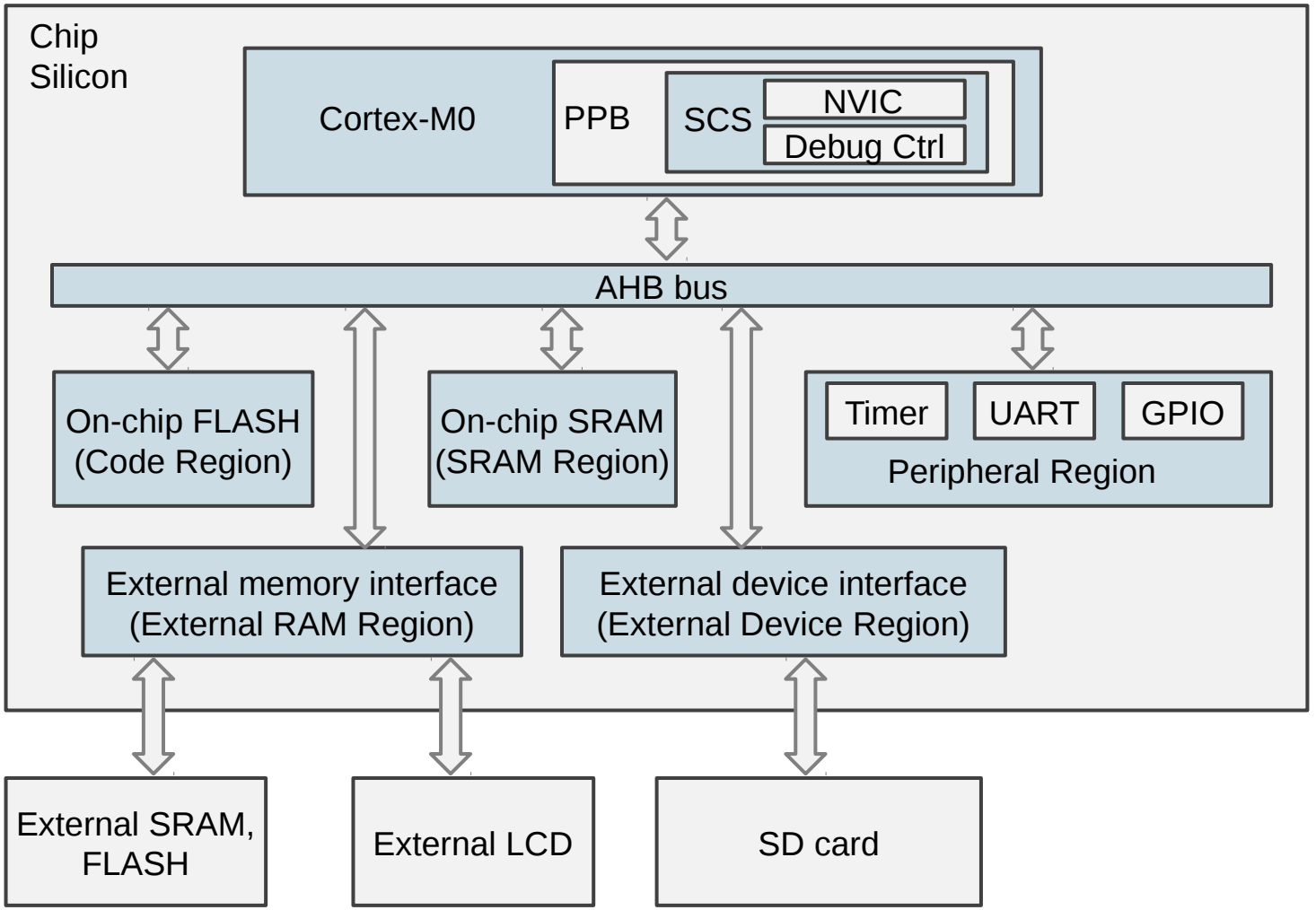
Cortex-M0 Memory Map

- Code Region
  - Primarily used to store program code;
  - Can also be used for data memory;
  - On-chip memory, such as on-chip FLASH.
- SRAM Region
  - Primarily used to store data, such as heaps and stacks;
  - Can also be used for program code;
  - On-chip memory; despite its name “SRAM”, the actual device could be SRAM, SDRAM or other types.
- Peripheral Region
  - Primarily used for peripherals, such as Advanced High-performance Bus (AHB) or Advanced Peripheral Bus (APB) peripherals;
  - On-chip peripherals.

Cortex-M0 Memory Map

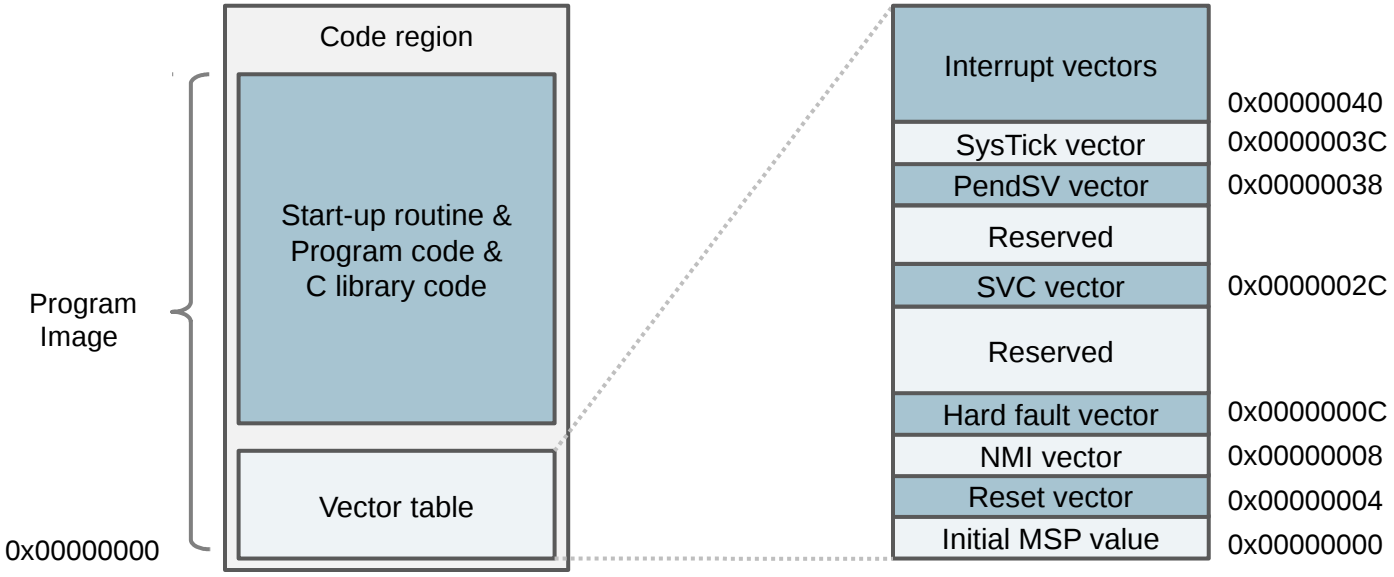
- External RAM Region
  - Primarily used to store large data blocks, or memory caches;
  - Off-chip memory, slower than on-chip SRAM region.
- External Device Region
  - Primarily used to map to external devices;
  - Off-chip devices, such as SD card.
- Internal Private Peripheral Bus (PPB)
  - Used inside the processor for processor's internal control;
  - Within PPB, a special range of memory is defined as System Control Space (SCS);
  - Nested Vectored Interrupt Controller (NVIC) is part of SCS.

Cortex-M0 Memory Map Example



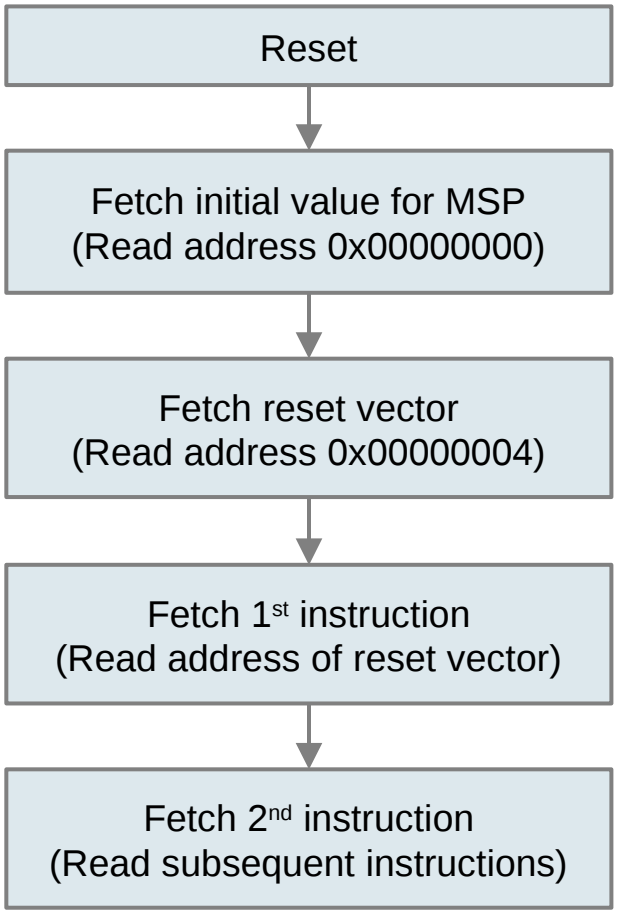
Cortex-M0 Program Image

- The program image in Cortex-M0 contains
  - Vector table -- includes the starting addresses of exceptions (vectors) and the value of the main stack point (MSP);
  - C start-up routine;
  - Program code – application code and data;
  - C library code – program codes for C library functions.



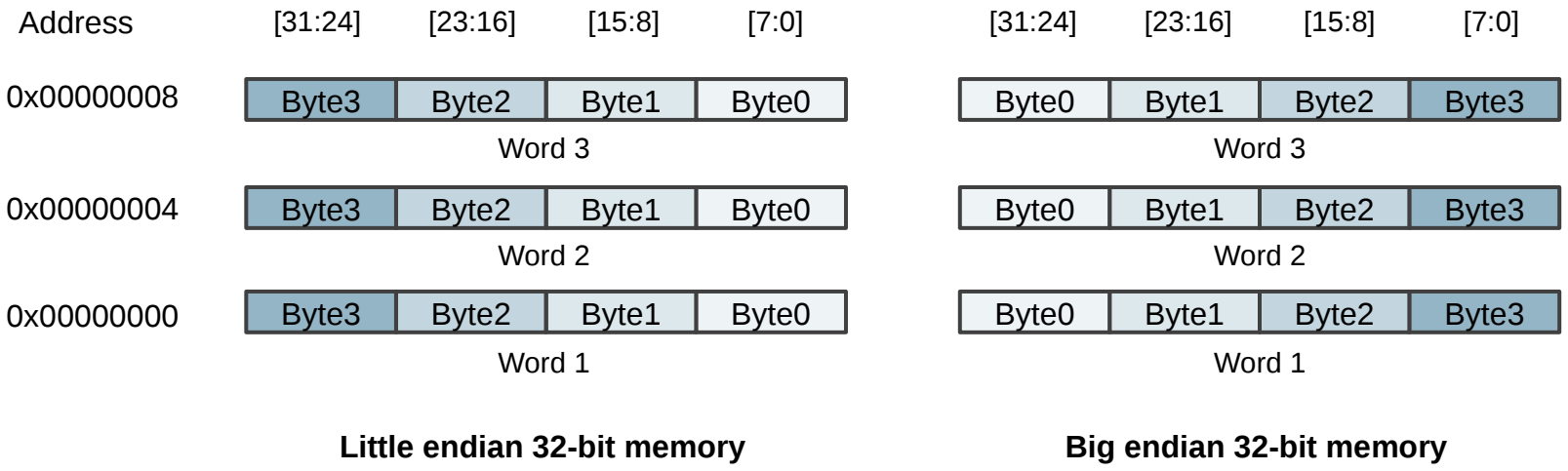
Cortex-M0 Program Image

- After reset:
  1. First reads the initial MSP value;
  2. Then reads the reset vector;
  3. Branches to the starting of the programme execution address (reset handler);
  4. Subsequently executes program instructions.



Cortex-M0 Endianness

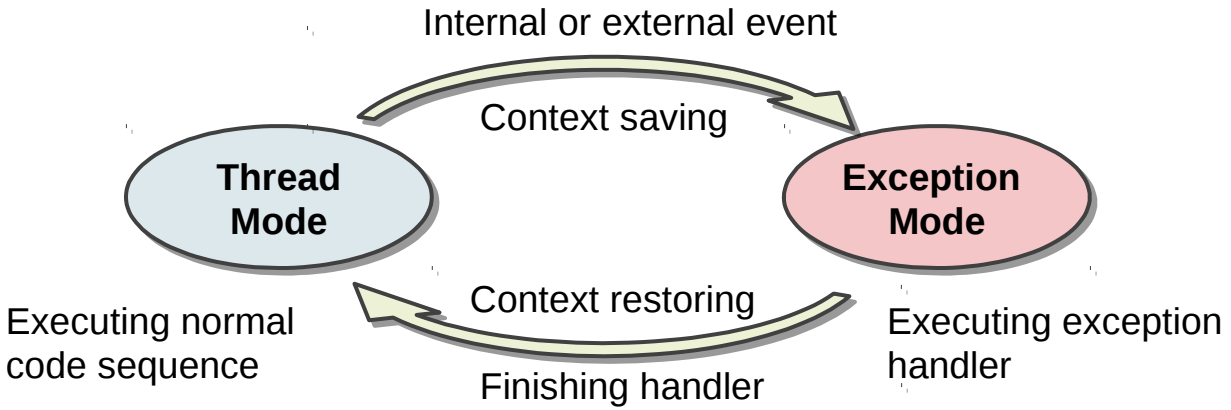
- Endian refers to the order of bytes stored in the memory
  - Little endian: lowest byte of a word-size data is stored in the bit 0 to bit 7
  - Big endian: lowest byte of a word-size data is stored in the bit 24 to bit 31
- Cortex-M0 supports both little endian and big endian
- However, Endianness only exists in the hardware level



ARM Cortex-M0 Processor Exceptions

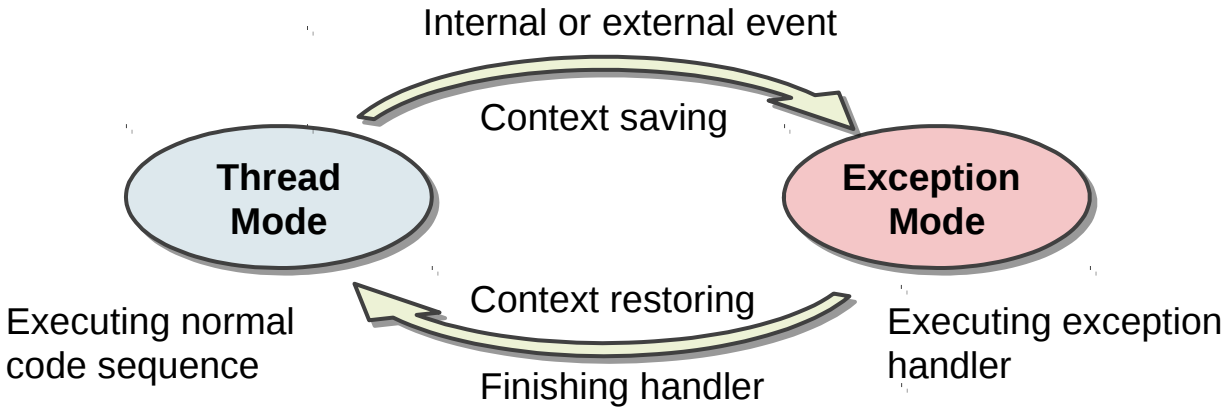
Cortex-M0 Exception Handling

- Exceptions are events that cause the program flow to exit the current program thread, and execute a piece of code associated with the event.
- Events can be either internal or external.
- The external event is also called interrupt request (IRQ).



Cortex-M0 Exception Handling

- Exception Handler
  - A piece of software code that is executed in the exception mode;
  - If the exception is caused by an IRQ, it can also be called as an interrupt handler, or interrupt service routine (ISR);
- Context Switching
  - Context saving: before entering the exception mode, the current program context, such as current registers values, are pushed onto the stack;
  - Context restoring: after finishing the handler, the previously stored context is restored by popping register values from the stack.



Cortex-M0 Exception Handling

- Exception Priority
  - The exceptions (or interrupts) are commonly divided into multiple levels of priorities;
  - A higher priority exception can be triggered and serviced during a lower priority exception;
  - Commonly known as a nested exception.
  - Exceptions can be disabled or enabled by software.

- Cortex-M0 Interrupt Controller
  - Supports up to 32 IRQ inputs and a non-maskable interrupt (NMI) inputs;
  - NMI is similar to IRQ but cannot be disabled and has the highest priority, useful for safety critical systems such as industrial control or automotive.

Vector Table for ARMv6-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

Vector Table in Assembly

- The interrupt vector can be defined in either C language or assembly language, for example in assembly:

```
  RESERVE8
                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
```