3.2.2. CortexM Microcontrollers

CortexM microcontrollers

- There are many CortexM microcontrollers, but they all use 32 bit microcontrollers.
- Many types: M0 (smallest, weaker), M0+, M1, M2 M4, M7, M23, M33 (most powerful).
- Three major subfamilies/architectures:
  ARMv6-M: cortex M0; cortex M0+, cortex M1
  ARMv7-M: Cortex M3
  ARMv7E-M: Cortex-M4, Cortex M7.
- CortexM series defines:

instruction set that you can issue

pieces of core processor functionalities that all CortexM microcontrollers have

CortexM Optional functionalities

- **Memory Protection Unit.**
  Used to protect firmware from applications.
  Lets the microcontroller operate in one of two modes: **user more** or **kernel mode**.

  - Memory protection unit can limit what memory the untrusted code (user code) can access, so to protect the trusted code (kernel code).

- **SysTick**, standard timer.

- Cortex-M supports all sort of extra features which are presented as peripherals, which appear as **blocks of memory** within the processor architecture.

An MCU: Atmel SAM4L series

- Cortex M4 core
- 128-512 KB flash memory (program), 32-64 KB RAM
- Operates at up to 48MHz at 1.68 to 3.6V
- 4 USART (UART/SPI) buses and 4 i2C buses - many buses to which you can attach many external chips and control them in parallel
- USB hardware support
- ADC (3-15 channels depending on model)
- 15 DMA (Direct memory access) channels for I/O processing offload

This means that there is a **peripheral dedicated hardware module (DMA controller)** with its own registers, control logic, etc. that the CPU can set up and that handles data transfer between memory and peripherals without involving the CPU.

✓ Supports multiple transfer type (M2M, M2P, P2M, P2P).

✓ Improves real time performance (CPU can handle other tasks while transfer happens in background).

- 1.5-3 μA sleep, 1.5 μs wakeup, 90μA/MHz active (4.3mA @ 48MHz)

A SoC: Nordic nRF51 SoC

- Cortex-M0 core
- Integrated BLE transceiver
- 128-256 flash memory, 16-32 KB RAM
- Operates at 16MHz at 1.68 to 3.6 V
- 1 UART, 1 SPI, one i2bus
- ADC
- 2.6 μA sleep, 4.2μs wakeup, 2.4-4.1mA active, 16mA TX, 13.4mA RX

A SoC: Nordic nRF51 SoC

- Cortex-M0 core
- Integrated BLE transceiver
- 128-256 flash memory, 16-32 KB RAM
- at 1.68 to 3.6 V
- 2bus
-
- 2.6 μA sleep, 4.2μs wakeup, 2.4-4.1mA active, 16mA TX, 13.4mA RX

Consumes more than SAM4L although the processor core is slower (this is because BLE and supporting circuits)

A SoC: Nordic nRF51 SoC

- Cortex-M0 core
- Integrated BLE transceiver
- 128-256 flash memory, 16-32 KB RAM
- Operates at 16MHz at 1.68 to 3.6 V
- 1 UART, 1 SPI, one i2bus
- ADC
- 2.6 μA sleep, 4.2μs wakeup, 2.4-4.1mA active, 16mA TX, 13.4mA RX

Consumption for transmitting and receiving (i.e., for using the radio) is very high compared to computation

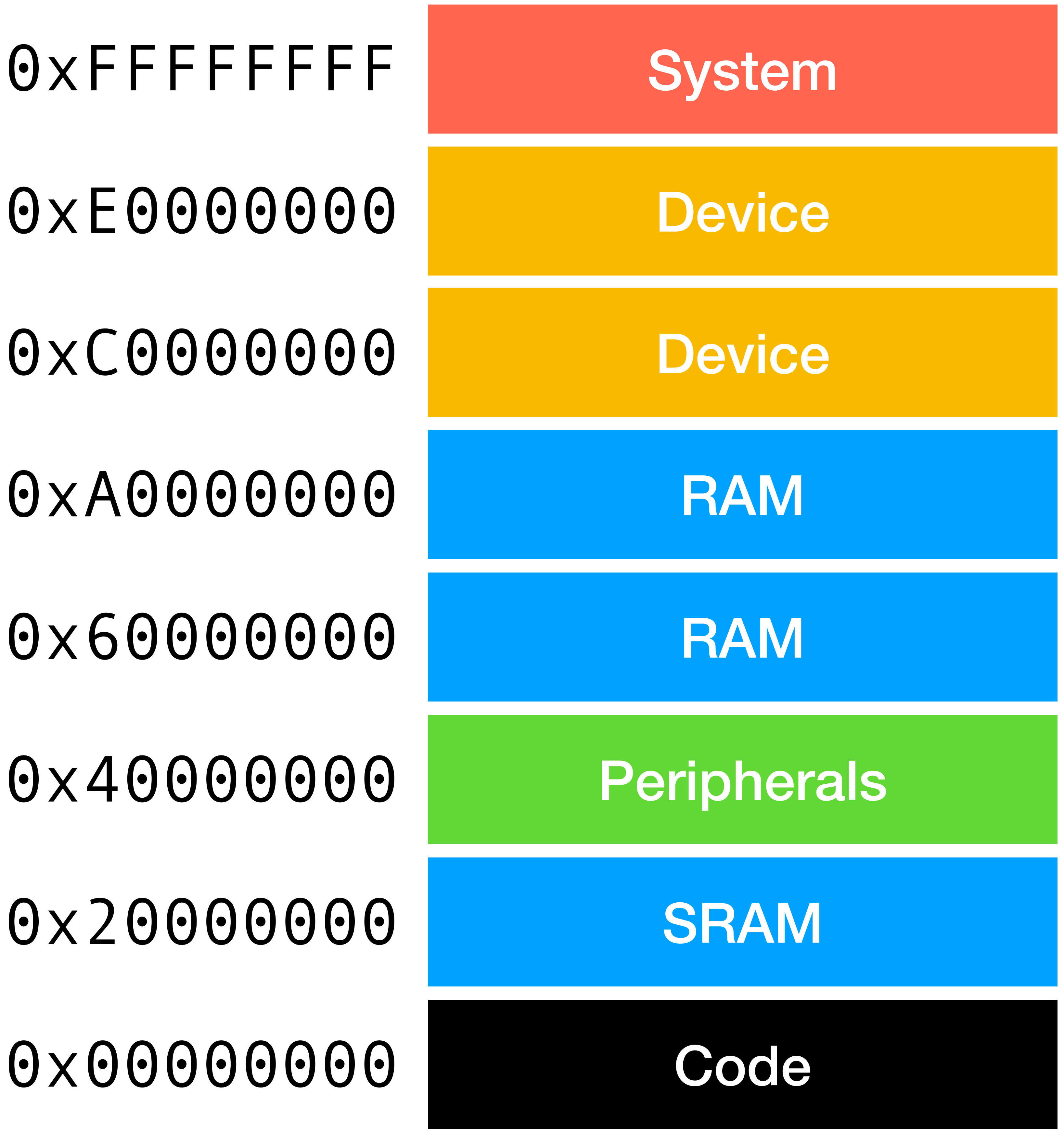
A SoC: Nordic nRF51 SoC

- Cortex-M0 core
- Integrated BLE transceiver
- 128-256 flash memory, 16-32 KB RAM
- Operates at 16MHz at 1.68 to 3.6 V
- 1 UART, 1 SPI, one i2bus
- ADC
- 2.6 μA sleep, 4.2μs wakeup, 2.4-4.1mA active, 16mA TX, 13.4mA RX

Consumption for transmitting and receiving (i.e., for using the radio) is very high compared to computation

→ Optimising for lifetime, power, energy means optimise for radio usage

CortexM memory Map (ARMv6-M, ARMv7-M)

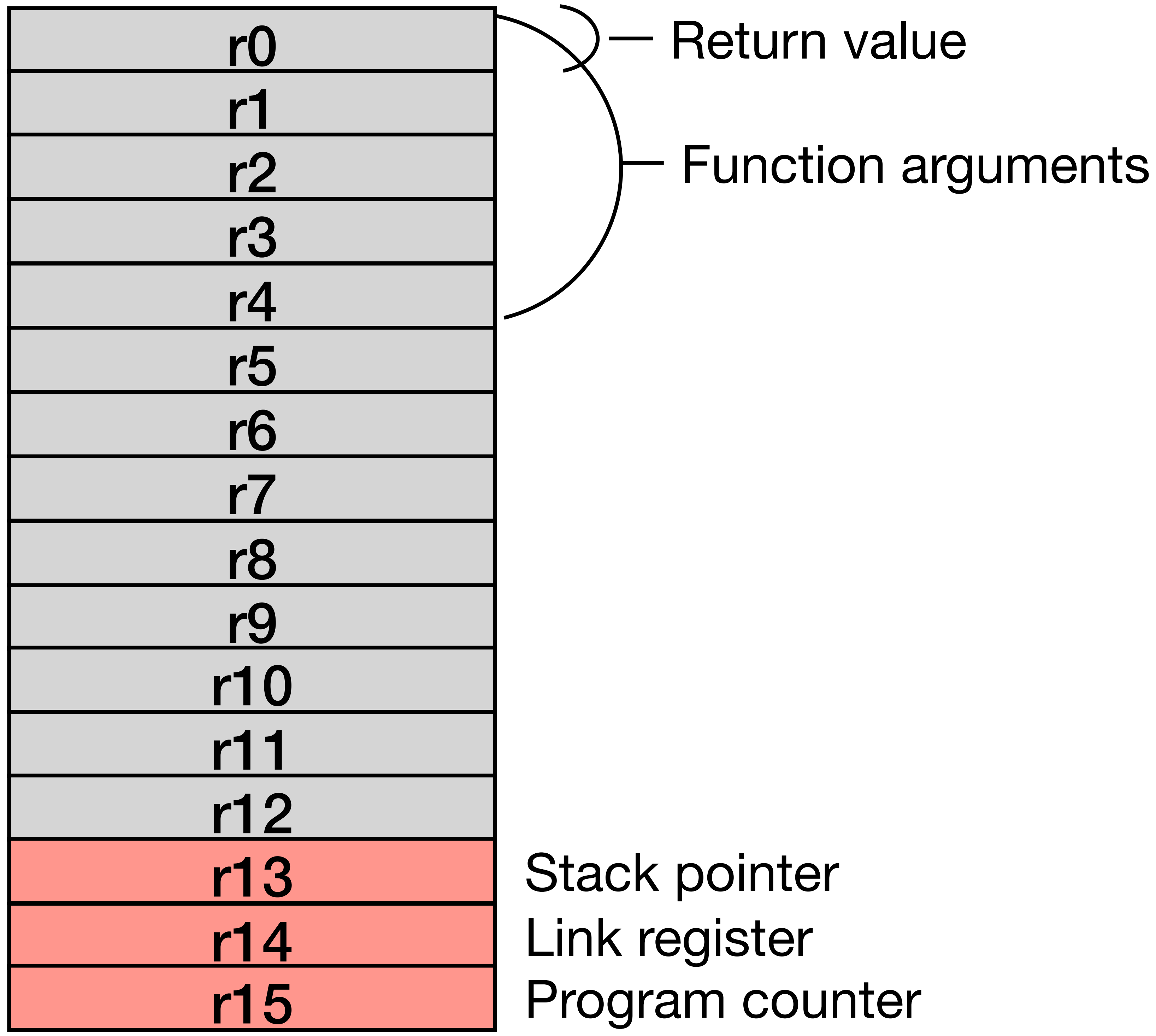


0xFFFFFFFF	System
0xE0000000	Device
0xC0000000	Device
0xA0000000	RAM
0x60000000	RAM
0x40000000	Peripherals
0x20000000	SRAM
0x00000000	Code

In CortexM microcontrollers, on chip features/components are modelled as memory-mapped peripherals.

Core Registers

- Let’s consider ARM architectures as an example.
- The CPU has 16 registers, r0, …, r15



r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13
r14
r15

Return value

Function arguments

Stack pointer

Link register

Program counter

The first registers are used for storing function arguments, and r0 is used to store the return value

REVIEW:

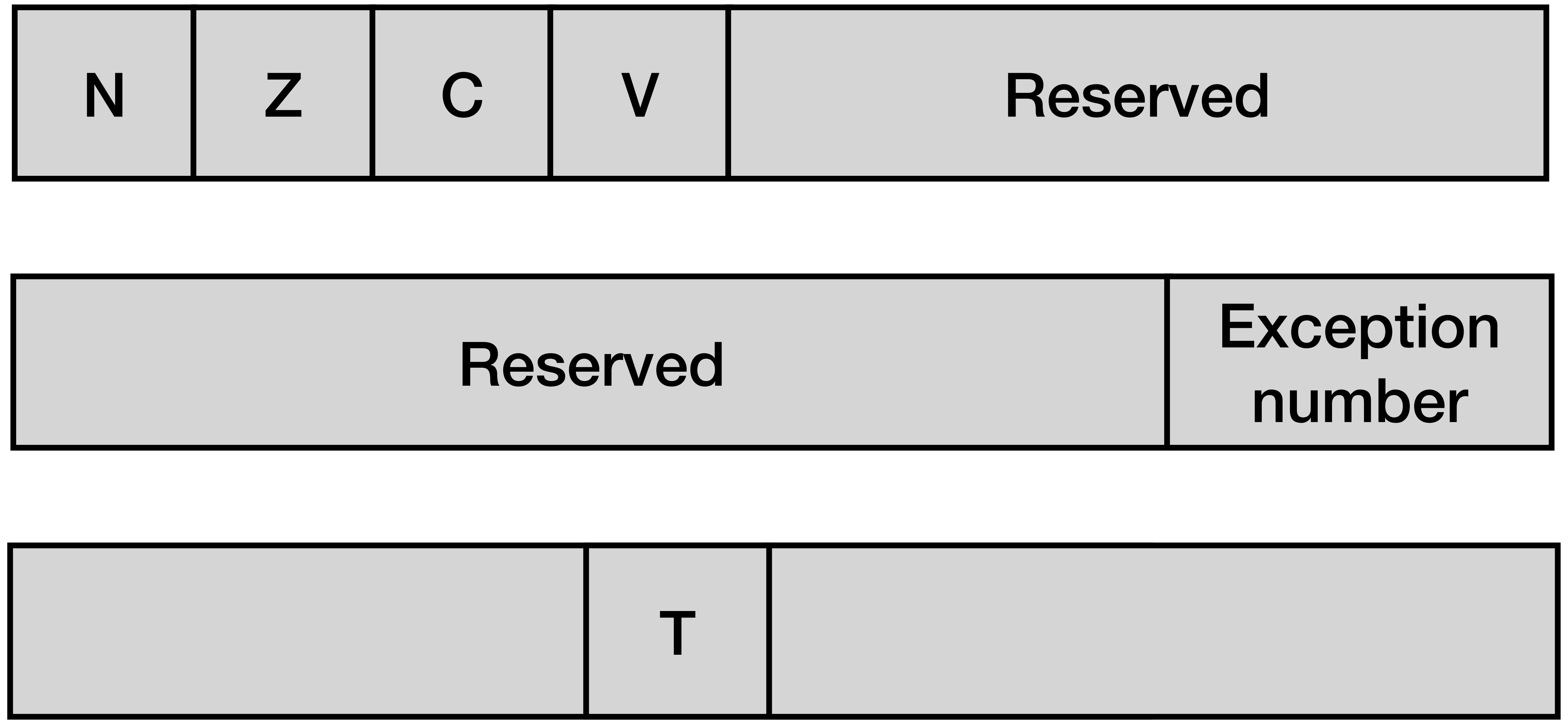
- **stack pointer: contains the location of the last item put onto the stack, i.e., the memory region that stores local variables, function parameters**
- **Link register: stores the return address for a function call.**
- **Program counter: contains the memory address of the next program instruction to be executed**

Running modes

- MCU can run in two modes: **user** and **kernel**.
- In user mode, applications run with limited privileges to prevent direct access to hardware, ensuring system stability.
- In kernel mode, the operating system has unrestricted access to all hardware resources and can perform critical tasks such as memory management and process control.
- A MCU typically switches from user mode to kernel mode during events that require higher privilege, for instance:
    - **system calls**, i.e., when a user application needs to request services from the operating systems (e.g., access to hardware).
    - **interrupts**, e.g., a peripheral sends an interrupt request
    - **exceptions** like faults or error
    - **context switching**, e.g., in case of multithreading, switching to kernel mode allows to manage scheduling and resource allocation

Control Registers

- In addition to the register used for computation, r0,…,r15, there are **control registers,** important for interrupt control, switching the addressing mode, paging control…



APSR (application program status register), containing information about the status of the processor

IPSR (interrupt program status register), containing the exception number of the current interrupt

EPSR (execution program status register), containing the thumb state bit - used for when instructions that take multiple cycles are interrupted, so the processor can resume those instructions

See https://developer.arm.com/documentation/dui1095/a/The-Cortex-M23-Processor/Programmers-model/Core-registers for more details

Interrupts (1)

- A hardware **interrupt trigger** is an event that generates an **interrupt request** (**IRQ**) via an electrical signal to the controller
- When an interrupt is received, the MCU:
    - halts the execution of currently running tasks,
    - starts running in kernel mode and performs a list of commands associated with the interrupt (called **interrupt handler function** or **interrupt service routine - ISR**),
    - resumes normal operations after the interrupt is handled.
- Some embedded systems are predominantly controlled by interrupts
  - handling them is of key importance

Interrupt Controller and states

Interrupt Controller and states

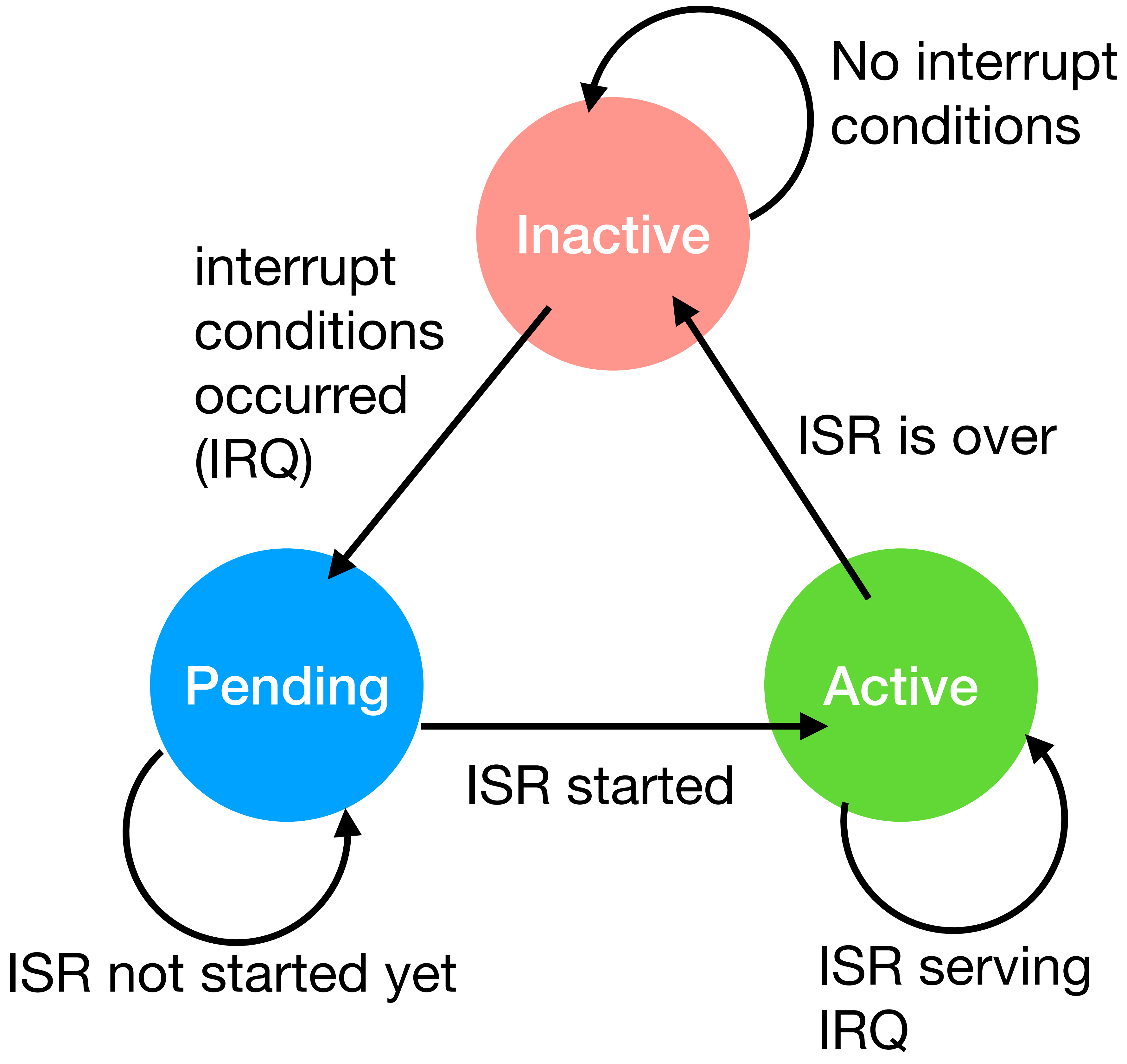
- The **interrupt controller** is a peripheral that helps the processor manage the interrupts.

Interrupt Controller and states

- The **interrupt controller** is a peripheral that helps the processor manage the interrupts.
- An **interrupt source** is any peripheral that can trigger an interrupt
  - ADC,
  - timer,
  - GPIO (General Purpose I/O),
  - UART.

Interrupt Controller and states

- The **interrupt controller** is a peripheral that helps the processor manage the interrupts.
- An **interrupt source** is any peripheral that can trigger an interrupt
  - ADC,
  - timer,
  - GPIO (General Purpose I/O),
  - UART.
- Interrupts have three states:
  - inactive, the conditions to generate the interrupt haven’t been met.
  - pending, the conditions have been met, but the ISR has not been called.
  - active, the ISR is serving the interrupt



Interrupt Vector Table (IVT)

- Interrupt service routines have addresses called **interrupt vectors** which are stored in the **interrupt vector table** (**IVT**).
- The IVT matches each ISR with IRQs coming from different interrupt sources.
  - IVTs are usually stored in the flash memory and are different depending on the vendor or the microcontroller.
- Interrupts usually come with **priorities** to determine which interrupt to serve first if more than one interrupt is pending.

Types of Interrupts

- Interrupts can be of two types:

- **Hardware interrupts** occur when the interrupt request (IRQ) comes from an external device as an input to the microcontroller.
  - Hardware interrupts can happen **asynchronously** at any given point during the execution of the program.
  - example: A temperature sensor has been programmed such that when the temperature becomes greater than 30 degrees celsius, it should inform the CPU.

- **Software interrupts** are called upon by the microcontroller when it executes special instructions or certain conditions are met while executing the code.
  - exceptions and traps
  - example: divisions by zero