SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

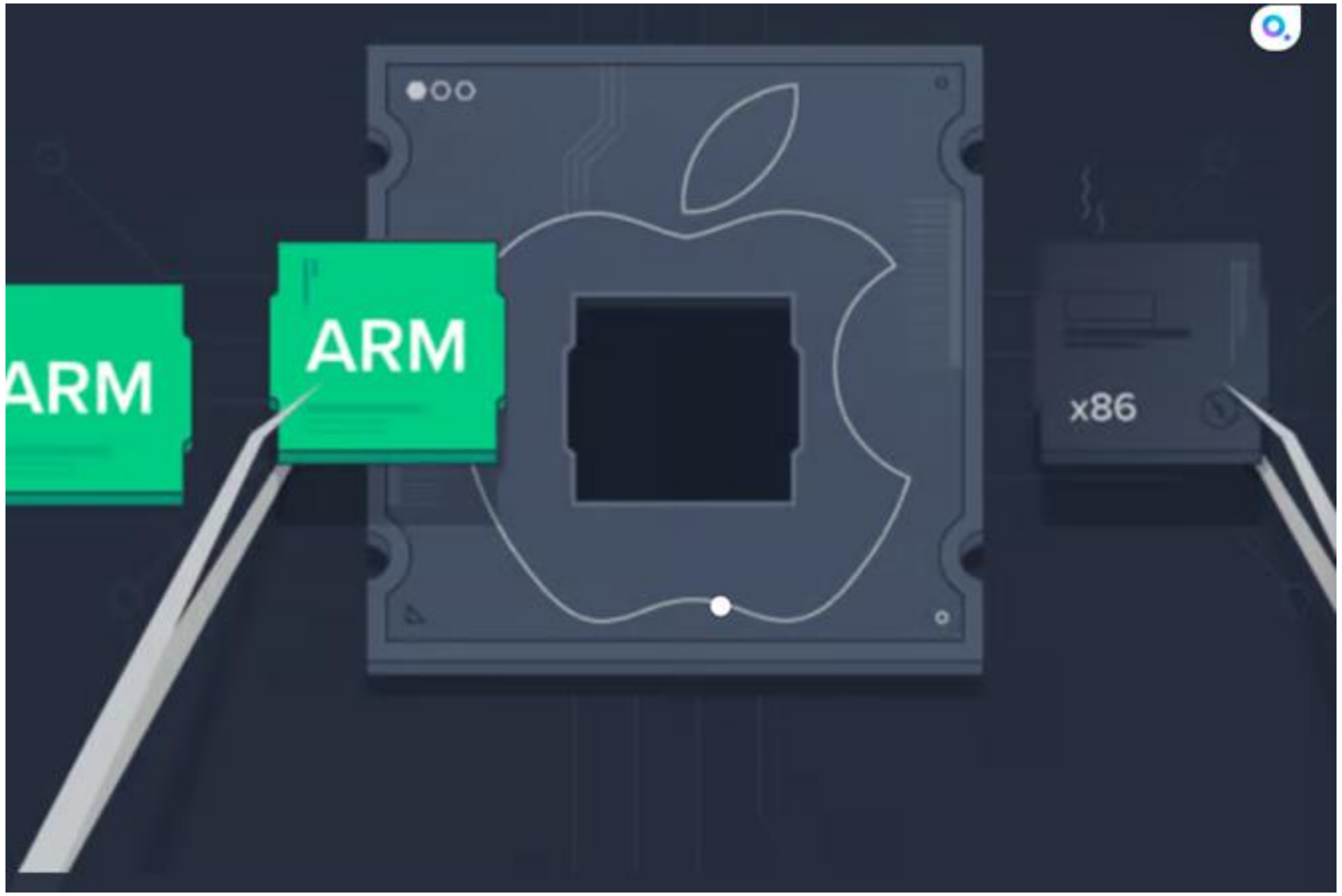
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ARM Cortex M0

Dr.G.Arthy
Assistant Professor
Department of EEE
SNS College of Engineering

ARM Partnership Model

Intel replaced with ARM



ARM Cortex

ARM Cortex processors come in various variants, each designed to cater to specific requirements in terms of performance, power efficiency, and application domains.

ARM Cortex Variants

Cortex-A series: Designed for applications requiring high performance and running operating systems like Linux. These CPUs are used in smartphones, tablets, and other computing devices.

Cortex-R series: Focused on real-time processing and used in applications where deterministic behavior and reliability are critical, like automotive systems, industrial control, and hard disk controllers.

Cortex-M series: Optimized for microcontroller applications, offering a balance between performance and power efficiency. These are widely used in embedded systems, IoT devices, and other applications requiring low power consumption.

Features

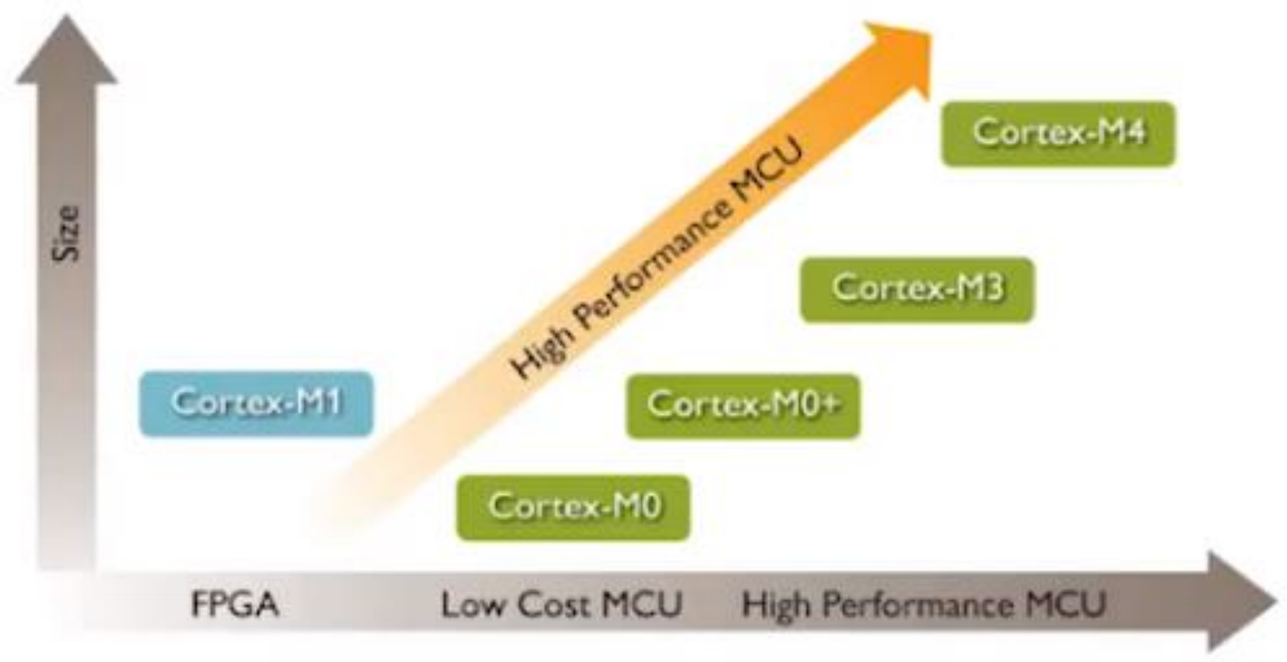
The Arm Cortex-M0 processor is one of the smallest ARM processors .

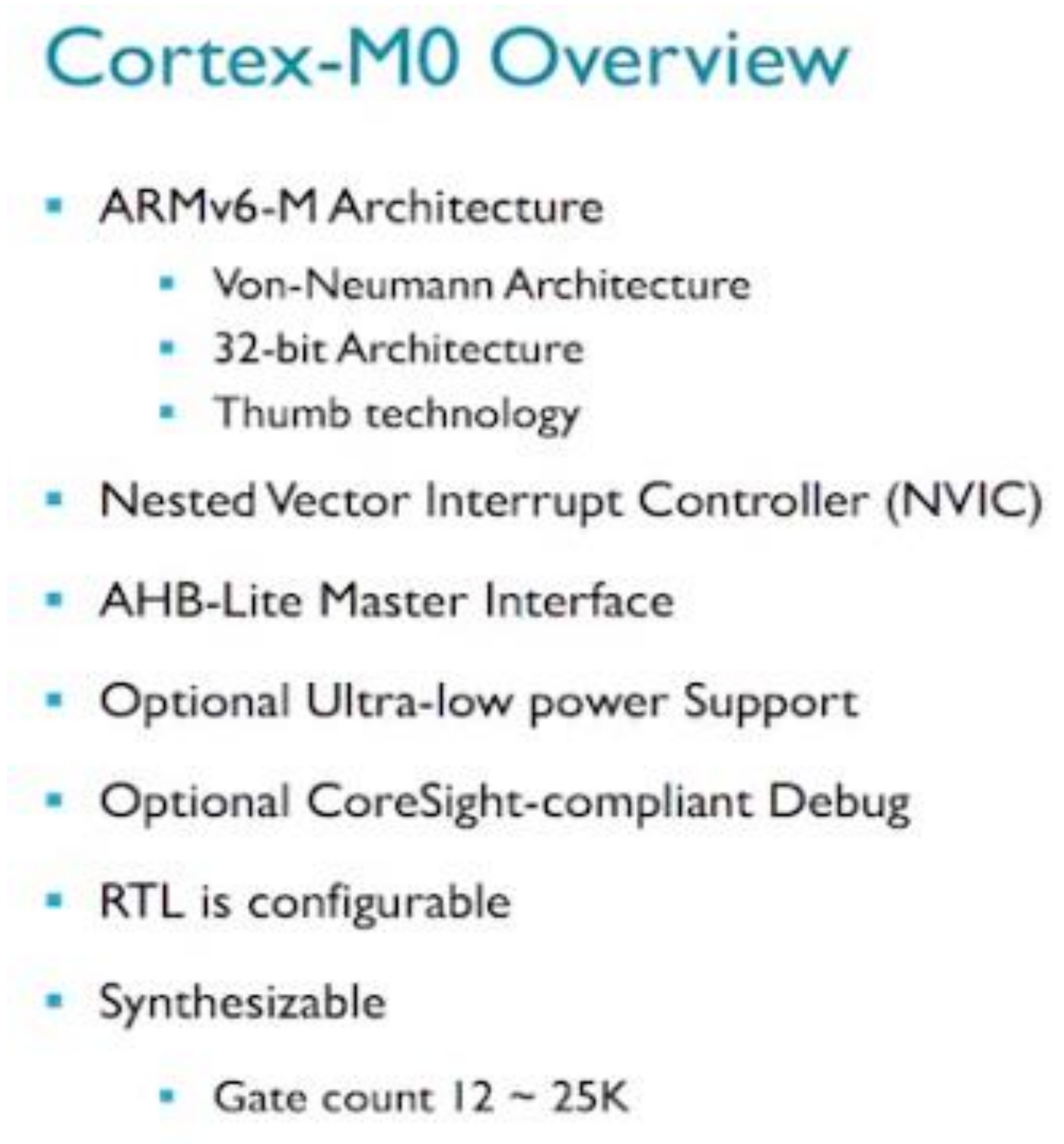
The Cortex-M0 has an exceptionally small silicon area, low power and minimal code footprint, enabling developers to achieve 32-bit performance at an 8-bit price point, bypassing the step to 16-bit devices. The ultra-low gate count of the processor enables its deployment in analog and mixed signal devices.

Equipment Adopting ARM Cores

Performance Comparison

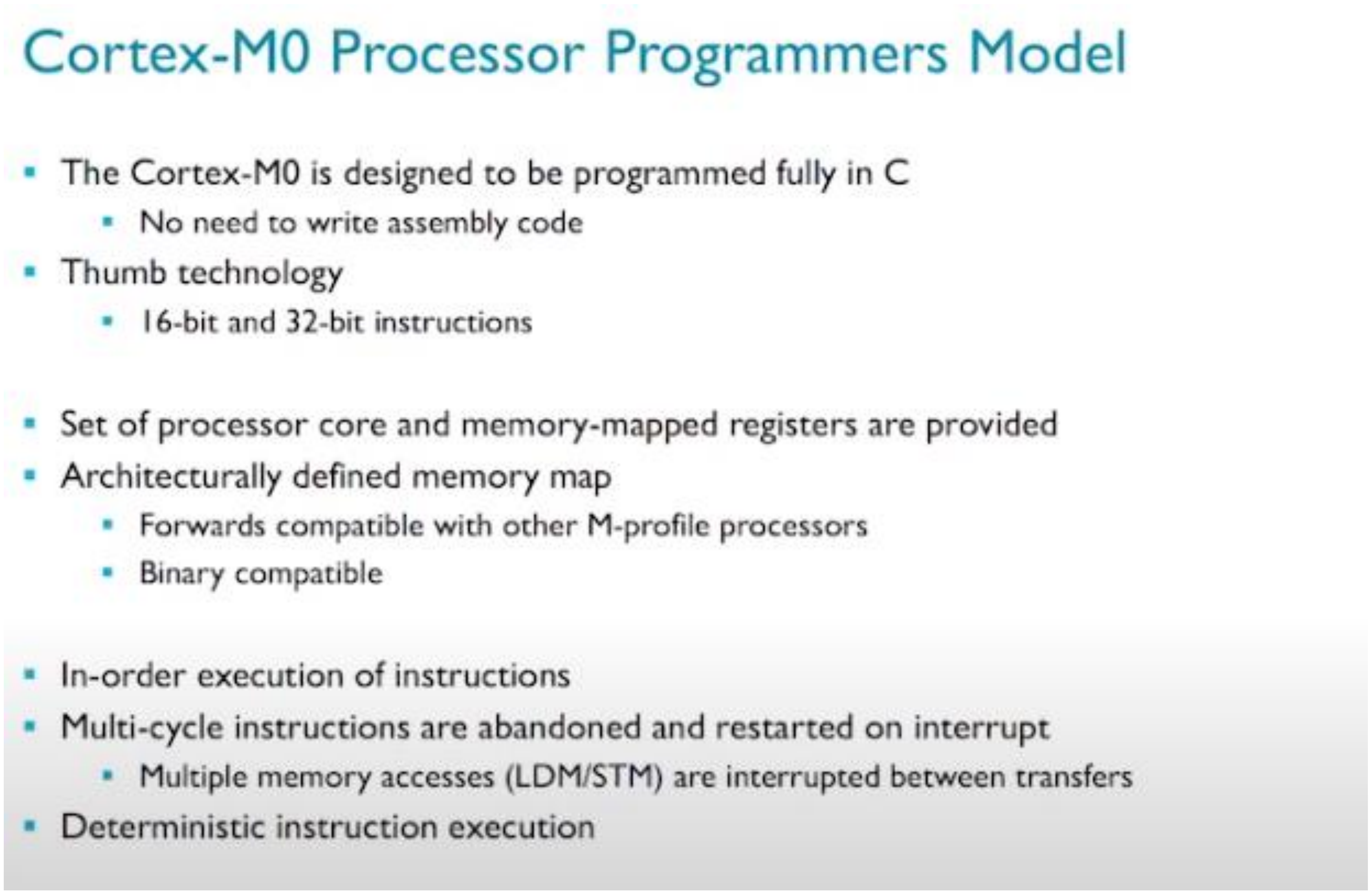
Cortex M Class Summary





Cortex-M0 Overview

- ARMv6-M Architecture
  - Von-Neumann Architecture
  - 32-bit Architecture
  - Thumb technology
- Nested Vector Interrupt Controller (NVIC)
- AHB-Lite Master Interface
- Optional Ultra-low power Support
- Optional CoreSight-compliant Debug
- RTL is configurable
- Synthesizable
  - Gate count 12 ~ 25K



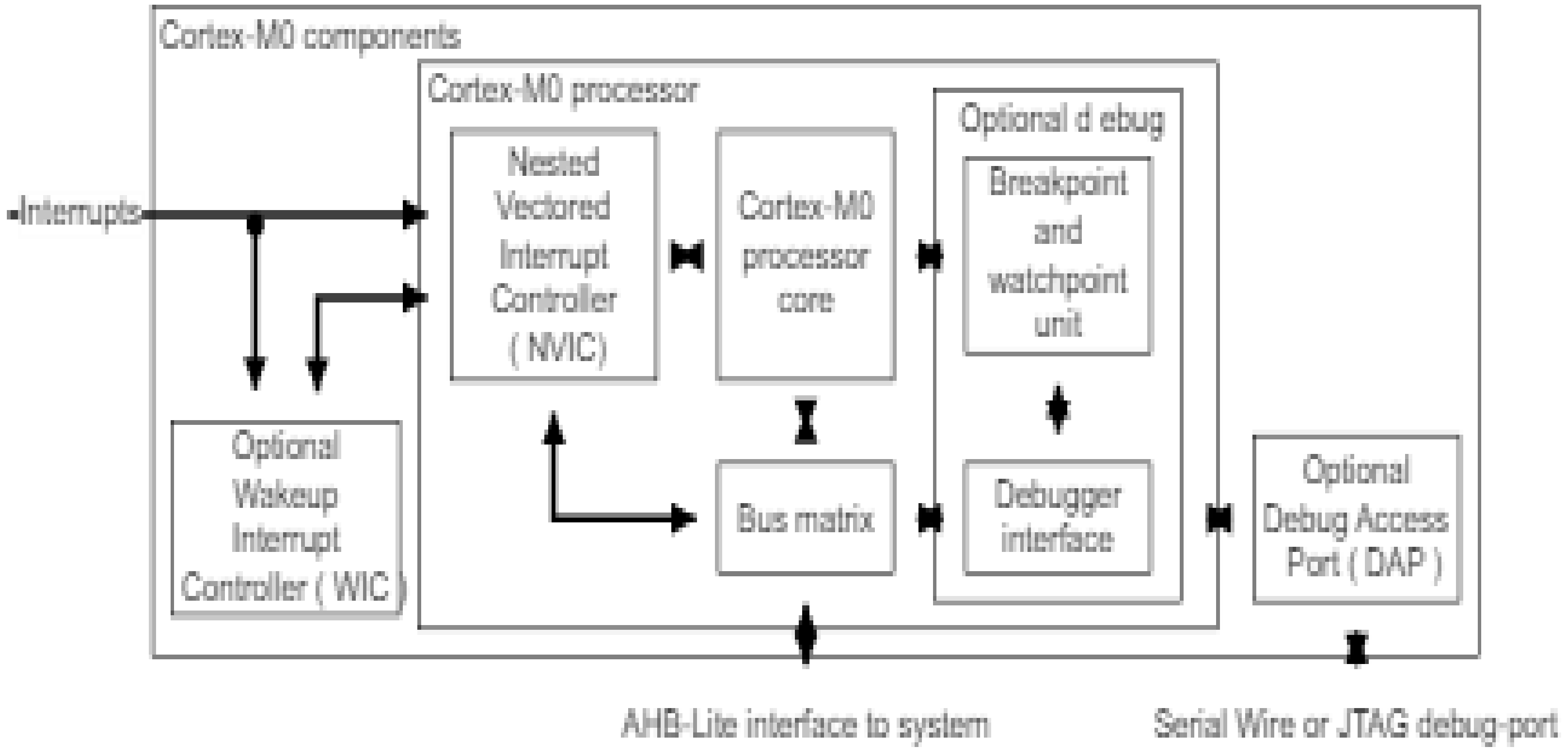
Cortex-M0 Processor Programmers Model

- The Cortex-M0 is designed to be programmed fully in C
  - No need to write assembly code
- Thumb technology
  - 16-bit and 32-bit instructions

- Set of processor core and memory-mapped registers are provided
- Architecturally defined memory map
  - Forwards compatible with other M-profile processors
  - Binary compatible

- In-order execution of instructions
- Multi-cycle instructions are abandoned and restarted on interrupt
  - Multiple memory accesses (LDM/STM) are interrupted between transfers
- Deterministic instruction execution

ARM Cortex M0 Overview



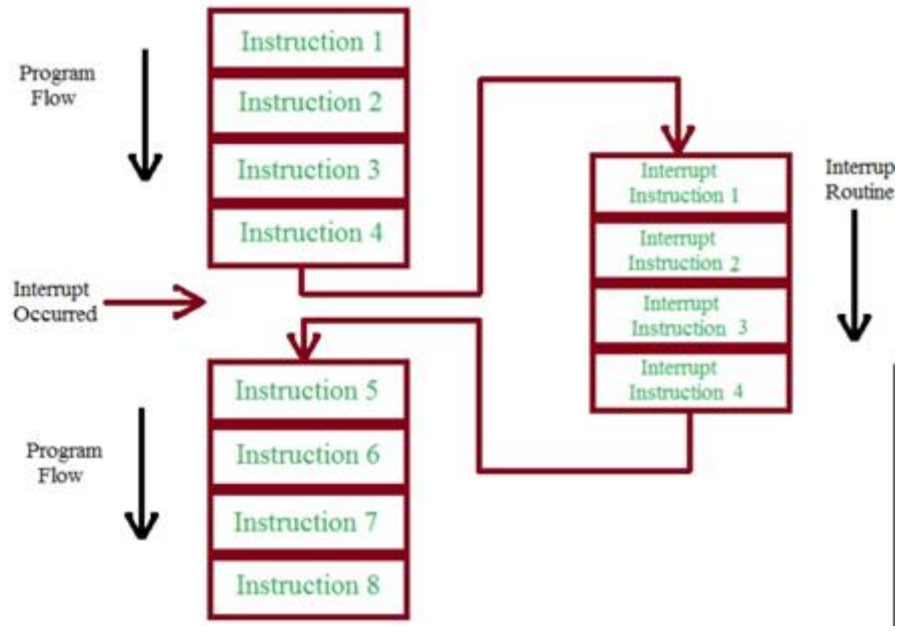
Cortex-M0 components
Cortex-M0 processor
Interrupts
Nested Vectored Interrupt Controller (NVIC)
Cortex-M0 processor core
Optional d ebug
Breakpoint and watchpoint unit
Debugger interface
Bus matrix
Optional Wakeup Interrupt Controller (WIC)
Optional Debug Access Port (DAP)
AHB-Lite interface to system
Serial Wire or JTAG debug-port

Features

It consists of :

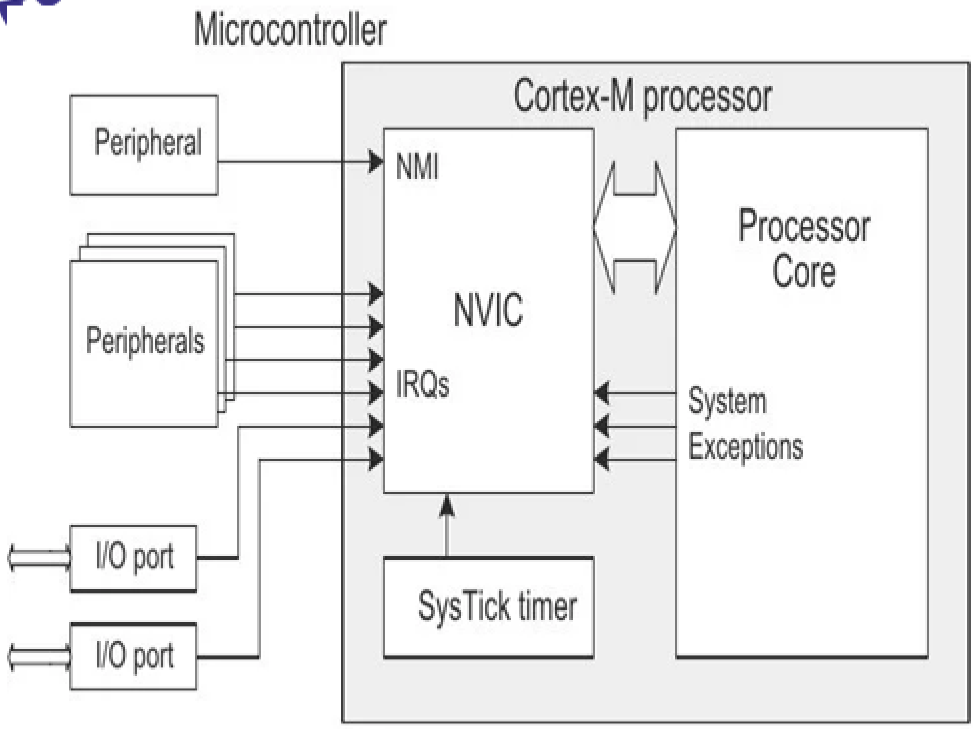
- Cortex-M core
- Memory (static ram and flash)
- A set of device specific peripherals
- A DMA controller.
- AHB Lite Master Interface
- Bus Matrix
- An interrupt controller (NVIC)
- A debugger interface

How Interrupts are processed?



- An interrupt is generated by any hardware or software source.
- The processor suspends the currently running task as the response of the interrupt.
- After that, the processor then services the interrupt service routine (also known as interrupt handler) to service the interrupt that occurred.
- The processor then returns to the suspended task and resumes its normal program execution.

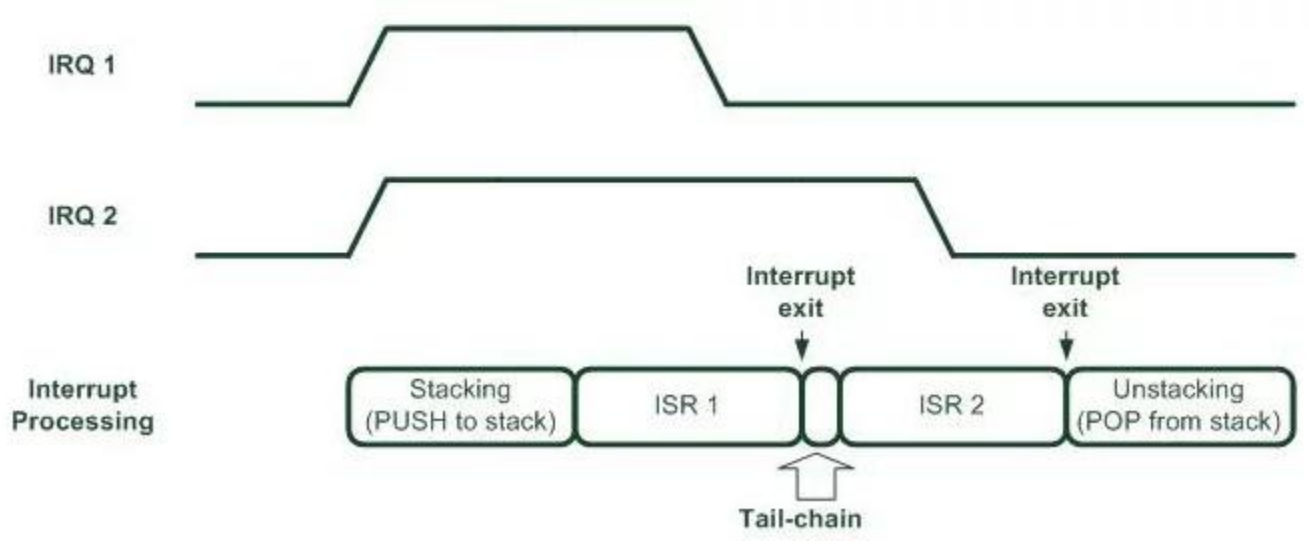
NVIC



A nested vectored interrupt controller is used to manage the interrupts from multiple interrupt sources. NVIC is closely integrated with the processor core to achieve low-latency interrupt processing and efficient processing of late-arriving interrupts.

Arm cortex M controllers are using this NVIC.

“Nested” in NVIC



Wakeup Interrupt Controller (WIC)

- The Wakeup Interrupt Controller (WIC) is a peripheral that can detect an interrupt and wake the processor from deep sleep mode.
- The WIC is enabled only when the system is in deep sleep mode.
- The WIC is not programmable, and does not have any registers or user interface. It operates entirely from hardware signals.

Breakpoints and watchpoints

- Breakpoints and watchpoints are features of debuggers that allow you to pause the execution of your code at specific locations or conditions.

- Breakpoints are set on lines of code, and watchpoints are set on variables or expressions.

Debug Access Port(DAP)

DAP is a module that contains a set of registers and logic required to provide microcontroller debugging via a debug interface.

DAP consists of two main components: Debug Port (DP) and Access Port (AP)

AHB-Lite Interface

The main system bus uses the AHB-Lite protocol.(Advance High Performance Bus)

This is a version of the AHB system bus aimed at single-master system designs.

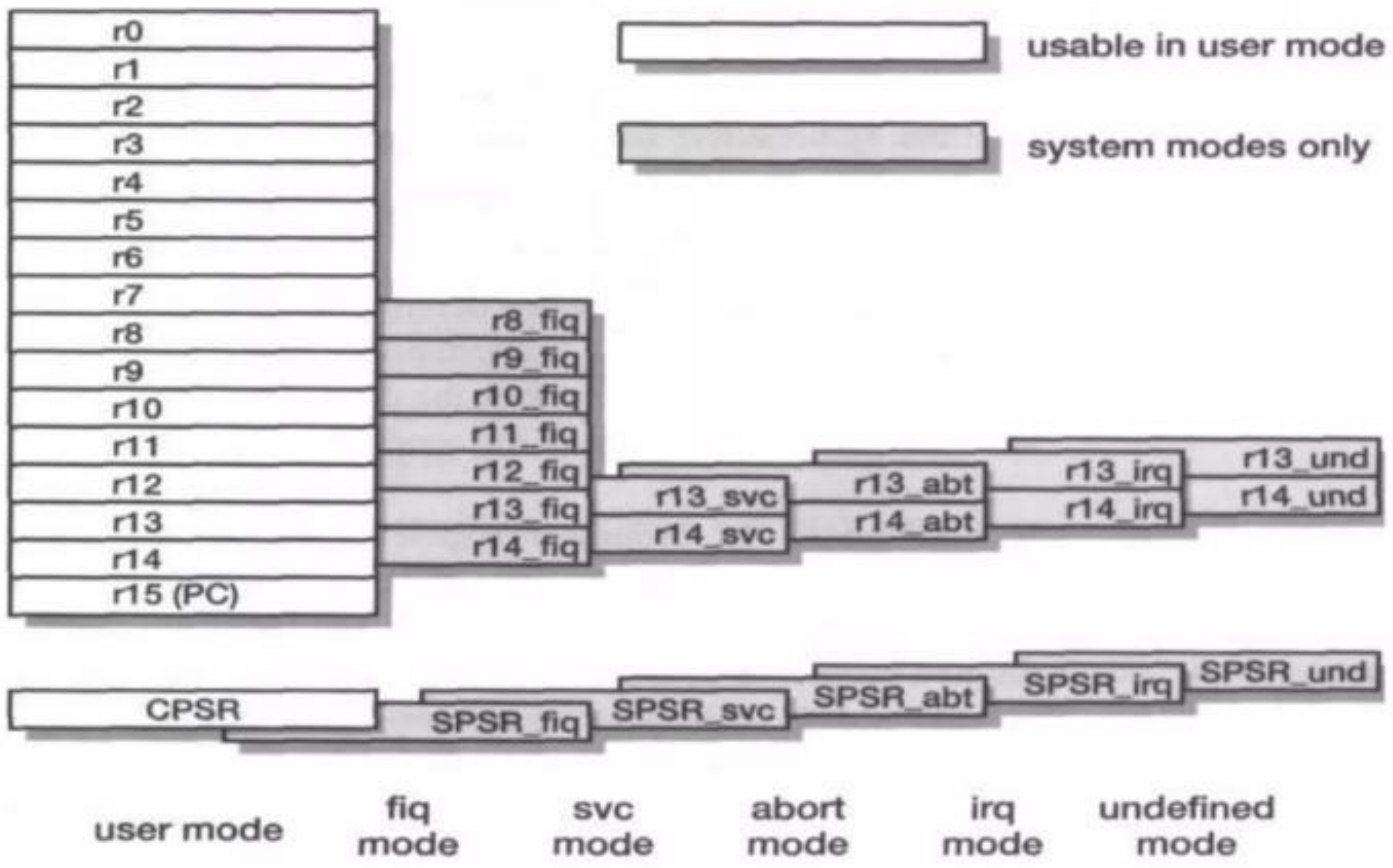
The ARM core is the only master permitted. The system bus allows the processor to access resources on the baseboard and on other modules.

ARM PROGRAMMER'S MODEL

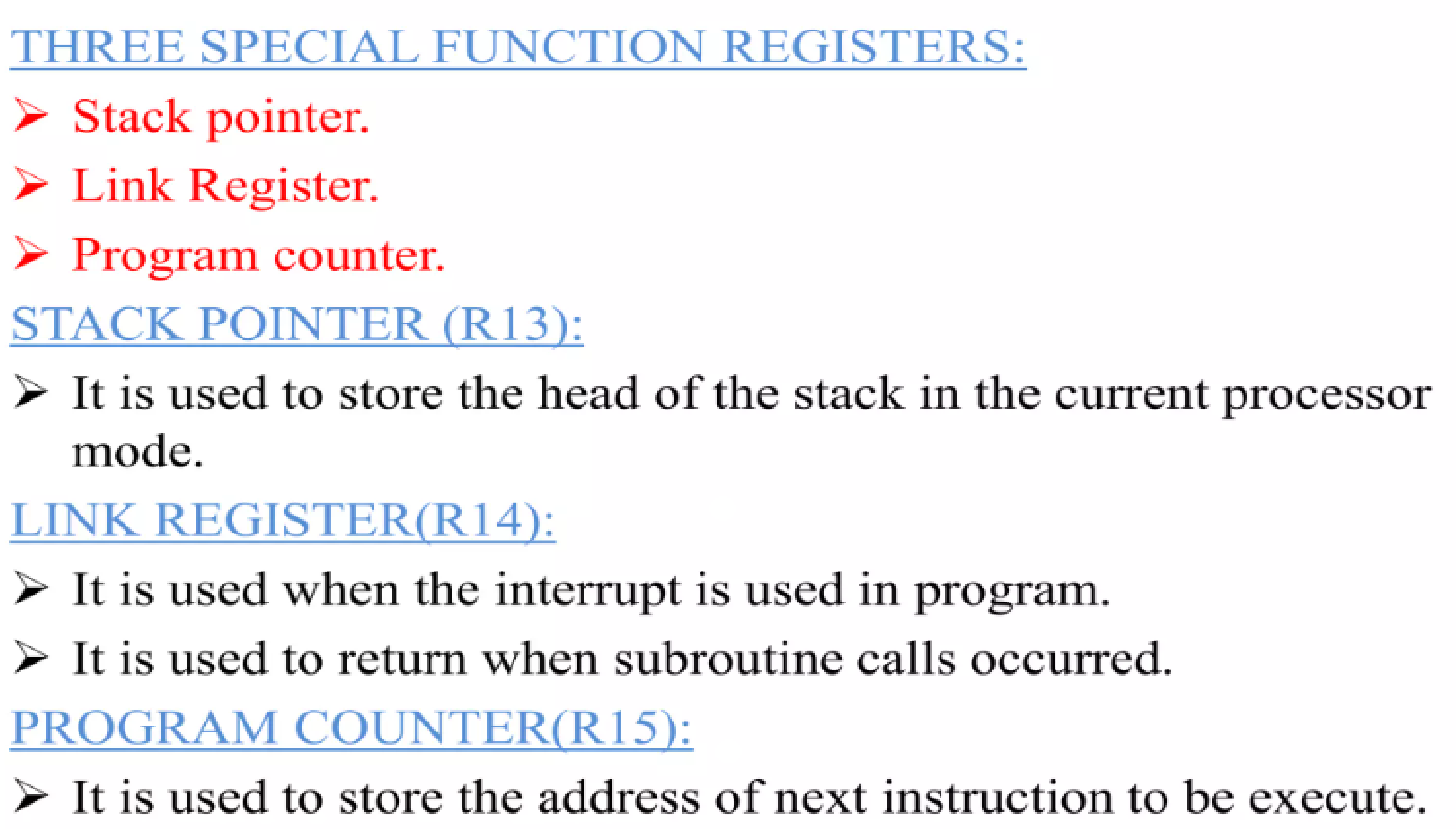
ARM Programmer’s Model

- ARM is a flexible programmer’s designed architecture with different applications.
- Design is simple, optimum and economic.
- A processor's instruction set defines the operations that the programmer can use to change the state of the system incorporating the processor.
- This state usually comprises the values of the data items in the processor's visible registers and the system's memory.
- Each instruction can be viewed as performing a defined transformation from the state before the instruction is executed to the state after it has completed.

ARM Registers



ARM Registers



THREE SPECIAL FUNCTION REGISTERS:

- ➢ Stack pointer.
- ➢ Link Register.
- ➢ Program counter.

STACK POINTER (R13):

- ➢ It is used to store the head of the stack in the current processor mode.

LINK REGISTER(R14):

- ➢ It is used when the interrupt is used in program.
- ➢ It is used to return when subroutine calls occurred.

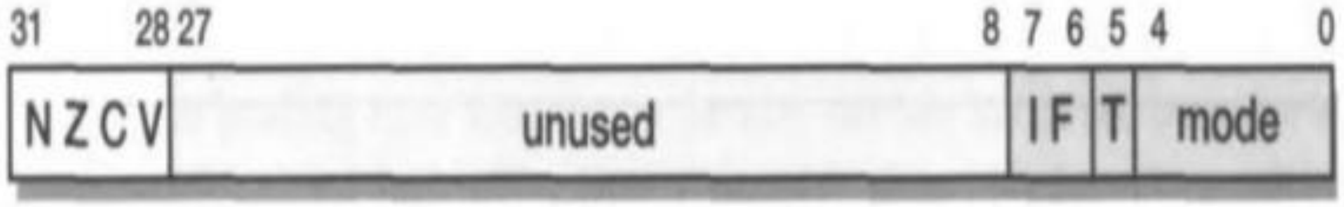
PROGRAM COUNTER(R15):

- ➢ It is used to store the address of next instruction to be execute.

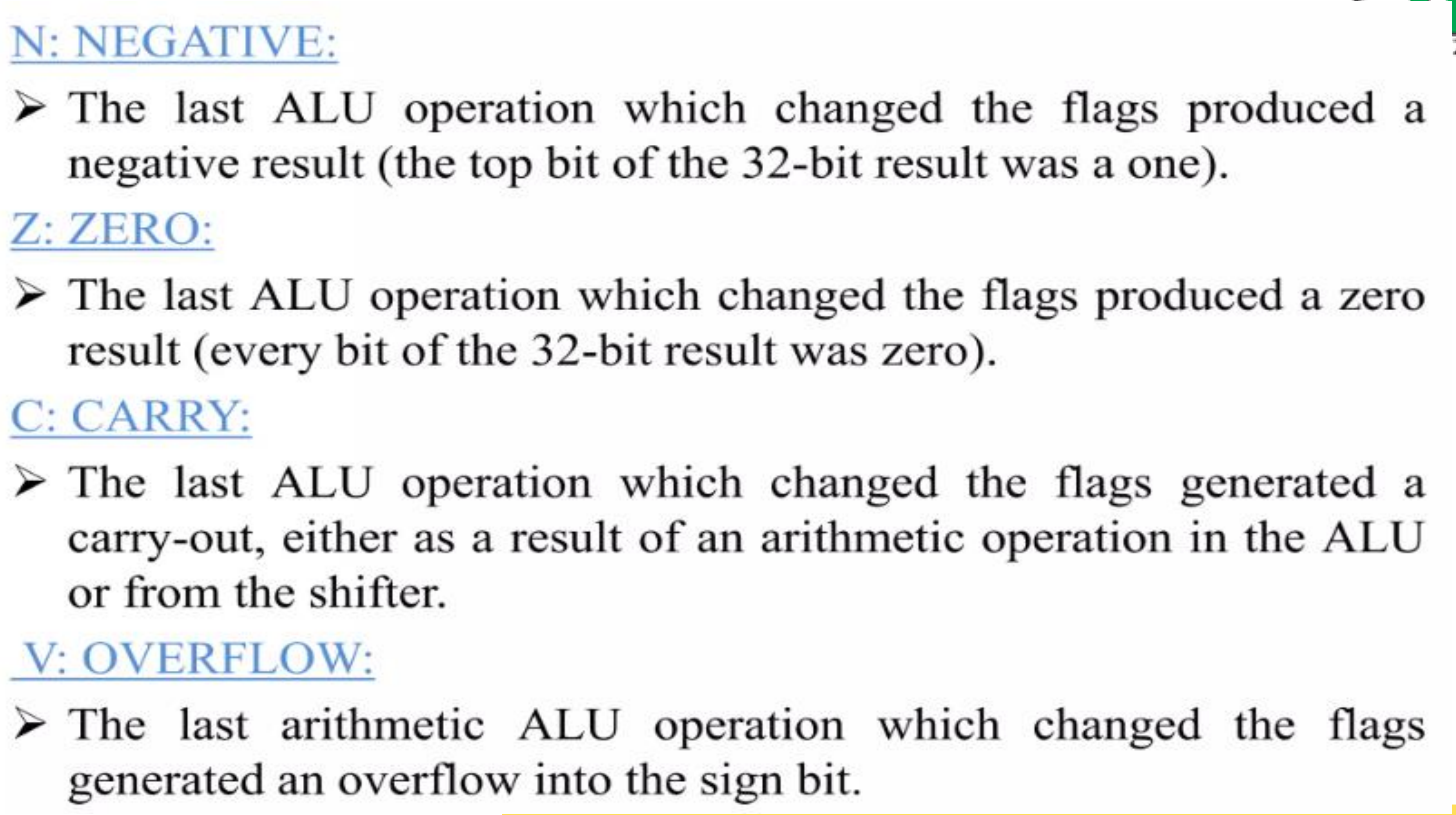
CPSR

THE CURRENT PROGRAM STATUS REGISTER (CPSR):

- CPSR is used to store the current status of ALU after execution of operations.
- The CPSR is used in user-level programs to store the condition code bits.
- These bits are used, for example, to record the result of a comparison operation and tc control whether or not a conditional branch is taken.



ARM Registers



N: NEGATIVE:

- The last ALU operation which changed the flags produced a negative result (the top bit of the 32-bit result was a one).

Z: ZERO:

- The last ALU operation which changed the flags produced a zero result (every bit of the 32-bit result was zero).

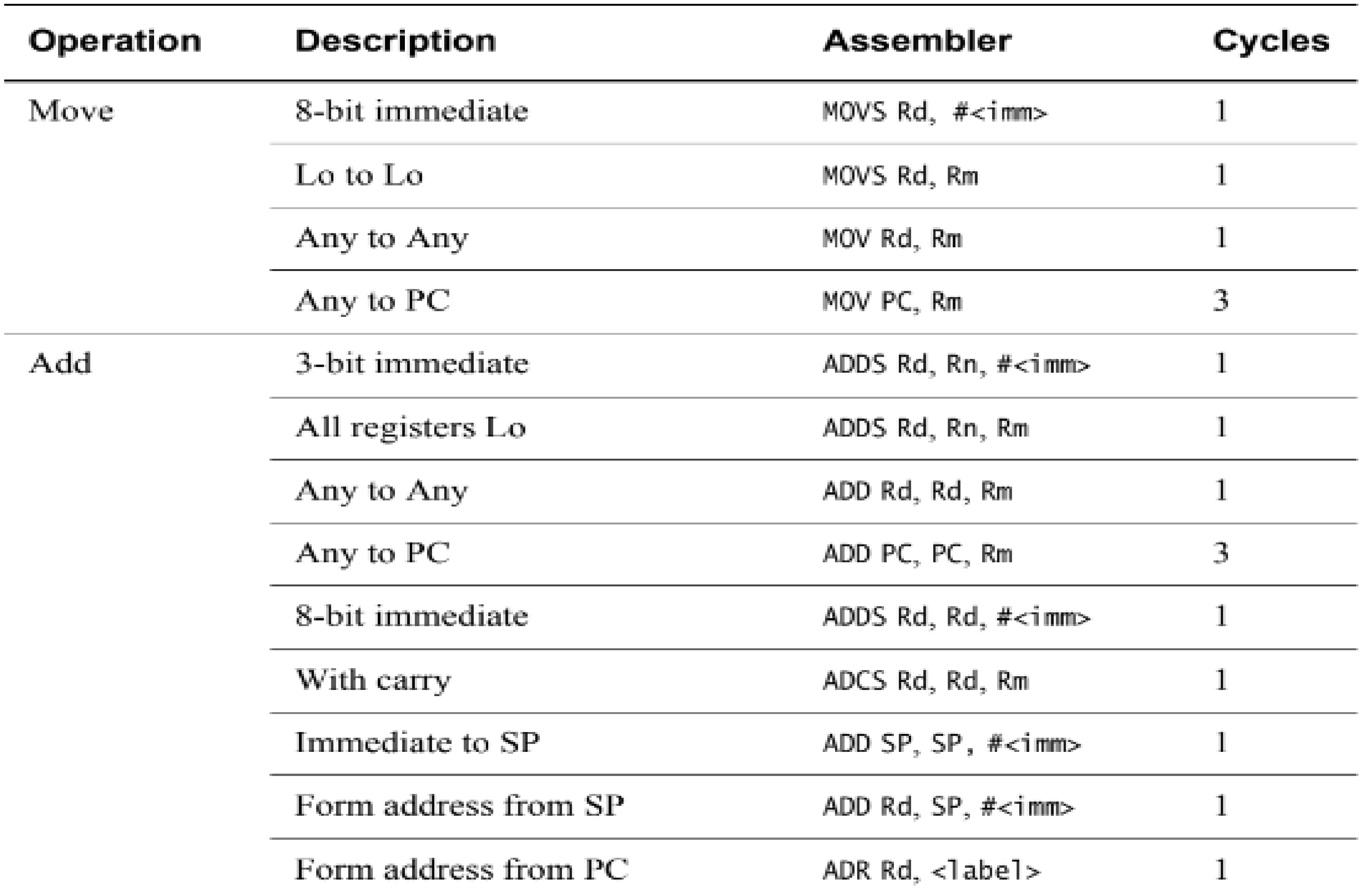
C: CARRY:

- The last ALU operation which changed the flags generated a carry-out, either as a result of an arithmetic operation in the ALU or from the shifter.

V: OVERFLOW:

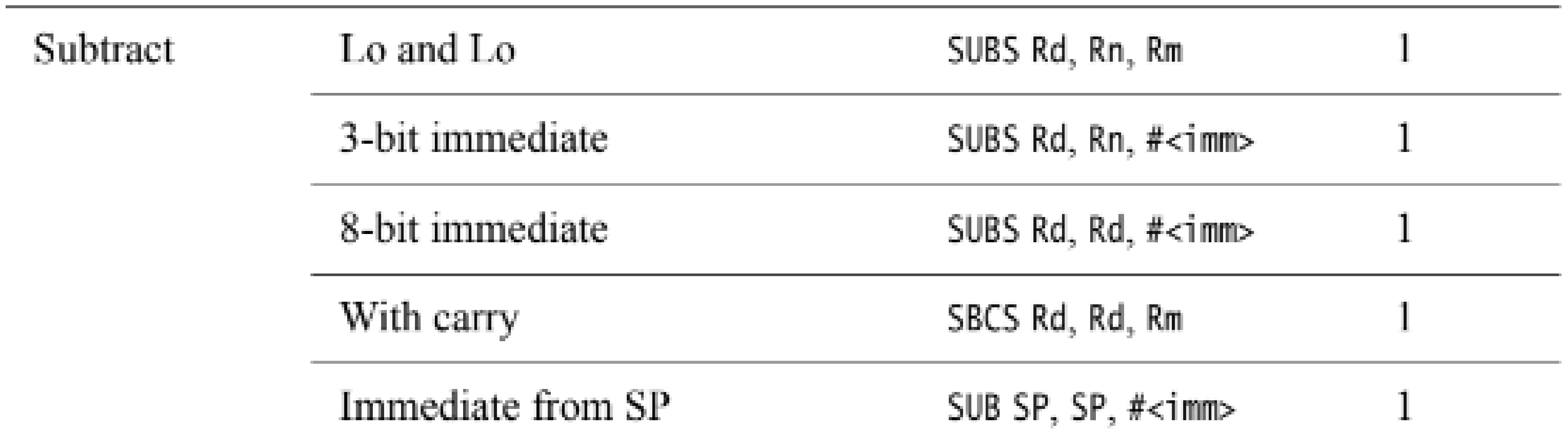
- The last arithmetic ALU operation which changed the flags generated an overflow into the sign bit.

Instruction Set



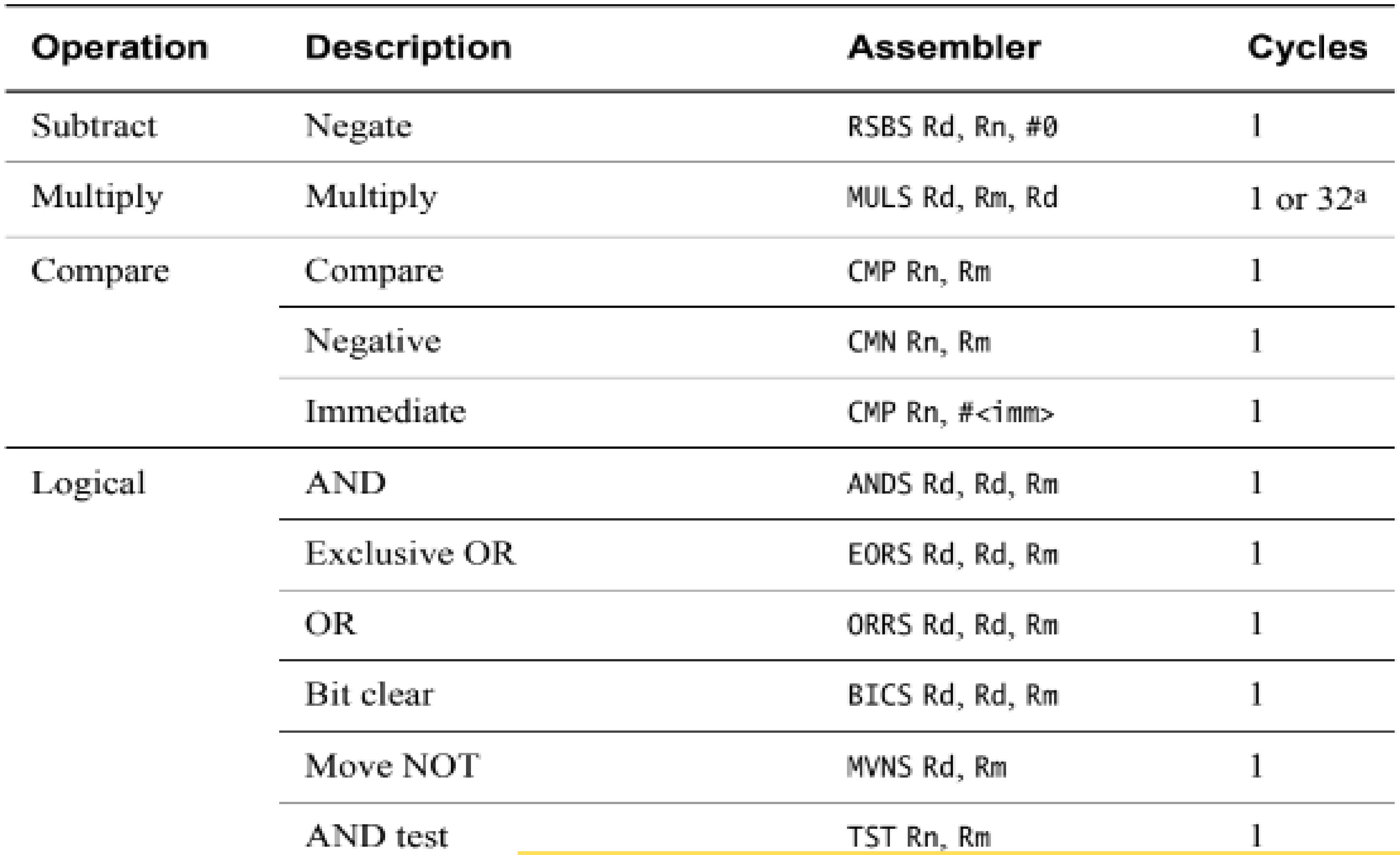
Operation	Description	Assembler	Cycles
Move	8-bit immediate	`MOVS Rd, #<imm>`	1
	Lo to Lo	`MOVS Rd, Rm`	1
	Any to Any	`MOV Rd, Rm`	1
	Any to PC	`MOV PC, Rm`	3
Add	3-bit immediate	`ADDS Rd, Rn, #<imm>`	1
	All registers Lo	`ADDS Rd, Rn, Rm`	1
	Any to Any	`ADD Rd, Rd, Rm`	1
	Any to PC	`ADD PC, PC, Rm`	3
	8-bit immediate	`ADDS Rd, Rd, #<imm>`	1
	With carry	`ADCS Rd, Rd, Rm`	1
	Immediate to SP	`ADD SP, SP, #<imm>`	1
	Form address from SP	`ADD Rd, SP, #<imm>`	1
	Form address from PC	`ADR Rd, <label>`	1

Instruction Set



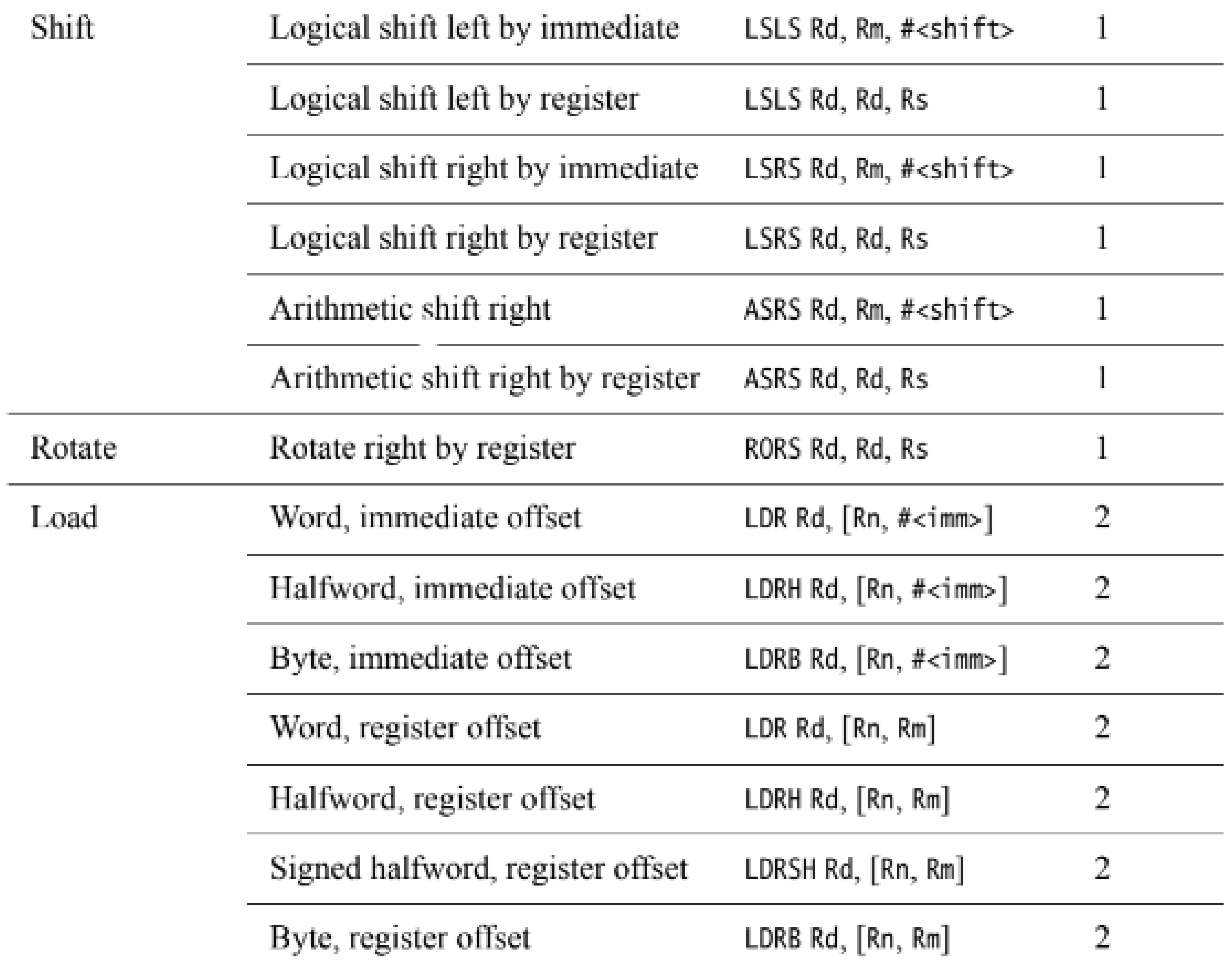
Subtract	Lo and Lo	SUBS Rd, Rn, Rm	1
	3-bit immediate	SUBS Rd, Rn, #<imm>	1
	8-bit immediate	SUBS Rd, Rd, #<imm>	1
	With carry	SBCS Rd, Rd, Rm	1
	Immediate from SP	SUB SP, SP, #<imm>	1

Instruction Set



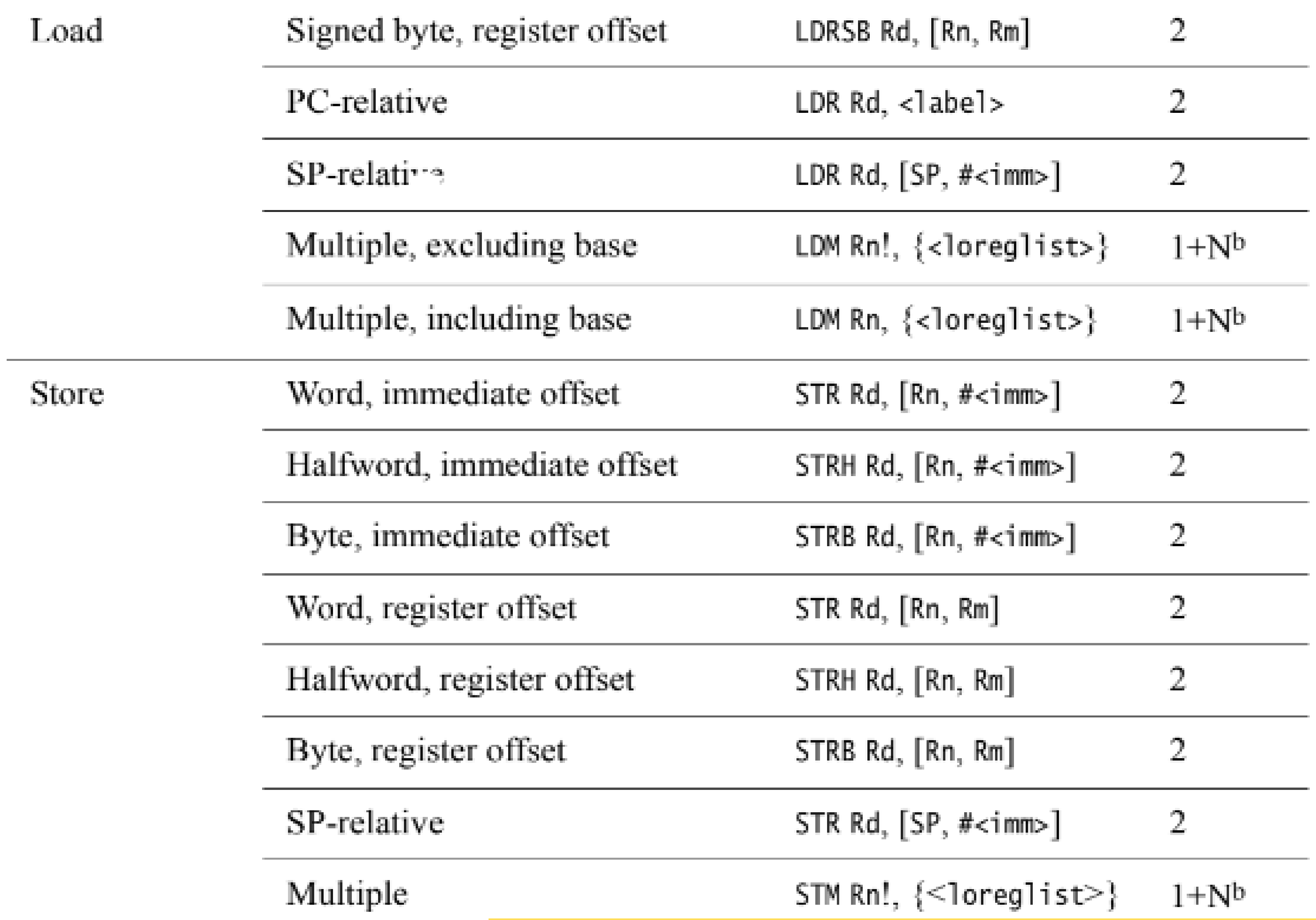
Operation	Description	Assembler	Cycles
Subtract	Negate	RSBS Rd, Rn, #0	1
Multiply	Multiply	MULS Rd, Rm, Rd	1 or 32[a]
Compare	Compare	CMP Rn, Rm	1
	Negative	CMN Rn, Rm	1
	Immediate	CMP Rn, #<imm>	1
Logical	AND	ANDS Rd, Rd, Rm	1
	Exclusive OR	EORS Rd, Rd, Rm	1
	OR	ORRS Rd, Rd, Rm	1
	Bit clear	BICS Rd, Rd, Rm	1
	Move NOT	MVNS Rd, Rm	1
	AND test	TST Rn, Rm	1

Instruction Set



Shift	Logical shift left by immediate	LSLS Rd, Rm, #<shift>	1
	Logical shift left by register	LSLS Rd, Rd, Rs	1
	Logical shift right by immediate	LSRS Rd, Rm, #<shift>	1
	Logical shift right by register	LSRS Rd, Rd, Rs	1
	Arithmetic shift right	ASRS Rd, Rm, #<shift>	1
	Arithmetic shift right by register	ASRS Rd, Rd, Rs	1
Rotate	Rotate right by register	RORS Rd, Rd, Rs	1
Load	Word, immediate offset	LDR Rd, [Rn, #<imm>]	2
	Halfword, immediate offset	LDRH Rd, [Rn, #<imm>]	2
	Byte, immediate offset	LDRB Rd, [Rn, #<imm>]	2
	Word, register offset	LDR Rd, [Rn, Rm]	2
	Halfword, register offset	LDRH Rd, [Rn, Rm]	2
	Signed halfword, register offset	LDRSH Rd, [Rn, Rm]	2
	Byte, register offset	LDRB Rd, [Rn, Rm]	2

Instruction Set

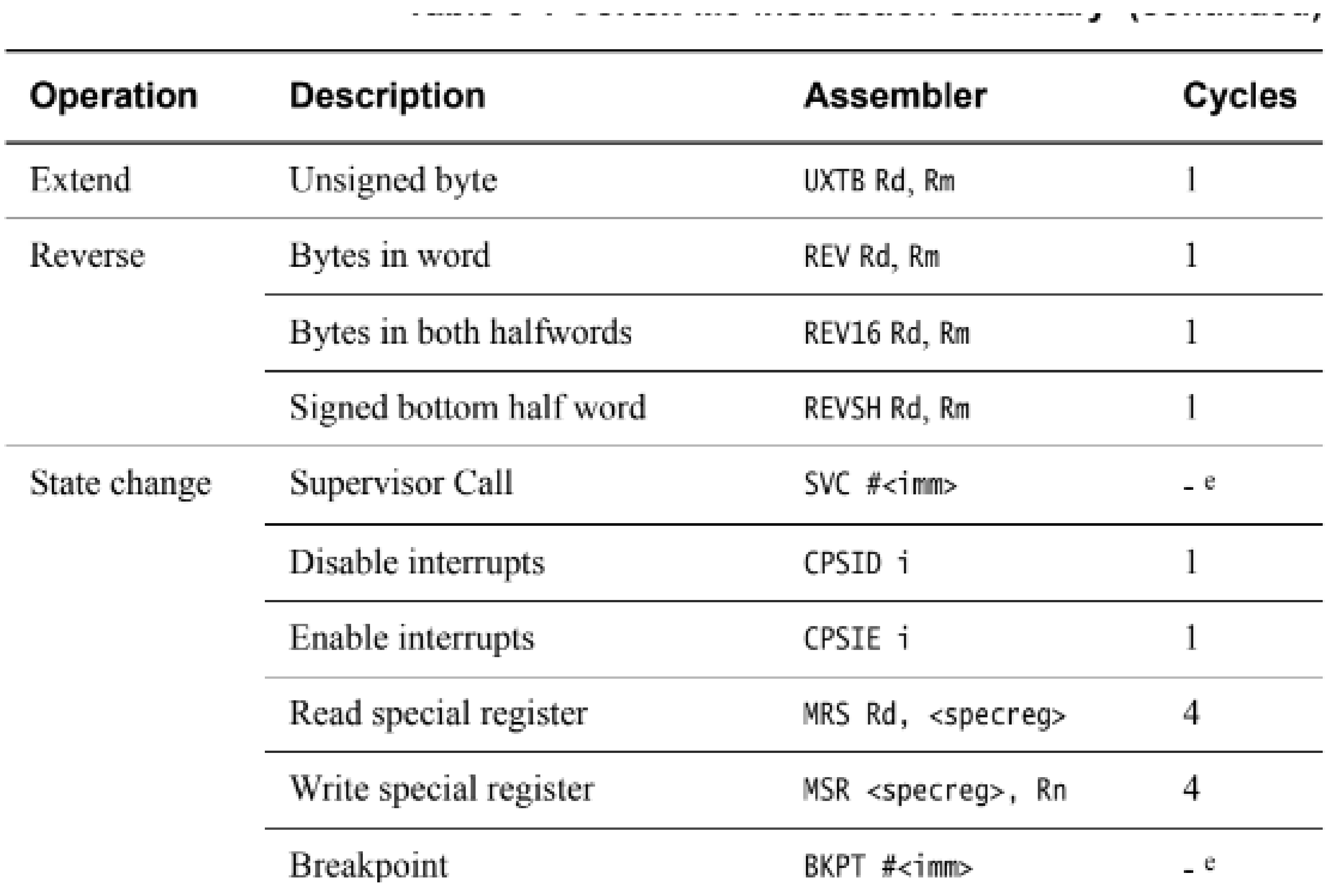


Load	Signed byte, register offset	LDRSB Rd, [Rn, Rm]	2
	PC-relative	LDR Rd, <label>	2
	SP-relative	LDR Rd, [SP, #<imm>]	2
	Multiple, excluding base	LDM Rn!, {<loreglist>}	1+N[b]
	Multiple, including base	LDM Rn, {<loreglist>}	1+N[b]
Store	Word, immediate offset	STR Rd, [Rn, #<imm>]	2
	Halfword, immediate offset	STRH Rd, [Rn, #<imm>]	2
	Byte, immediate offset	STRB Rd, [Rn, #<imm>]	2
	Word, register offset	STR Rd, [Rn, Rm]	2
	Halfword, register offset	STRH Rd, [Rn, Rm]	2
	Byte, register offset	STRB Rd, [Rn, Rm]	2
	SP-relative	STR Rd, [SP, #<imm>]	2
	Multiple	STM Rn!, {<loreglist>}	1+N[b]

Instruction Set

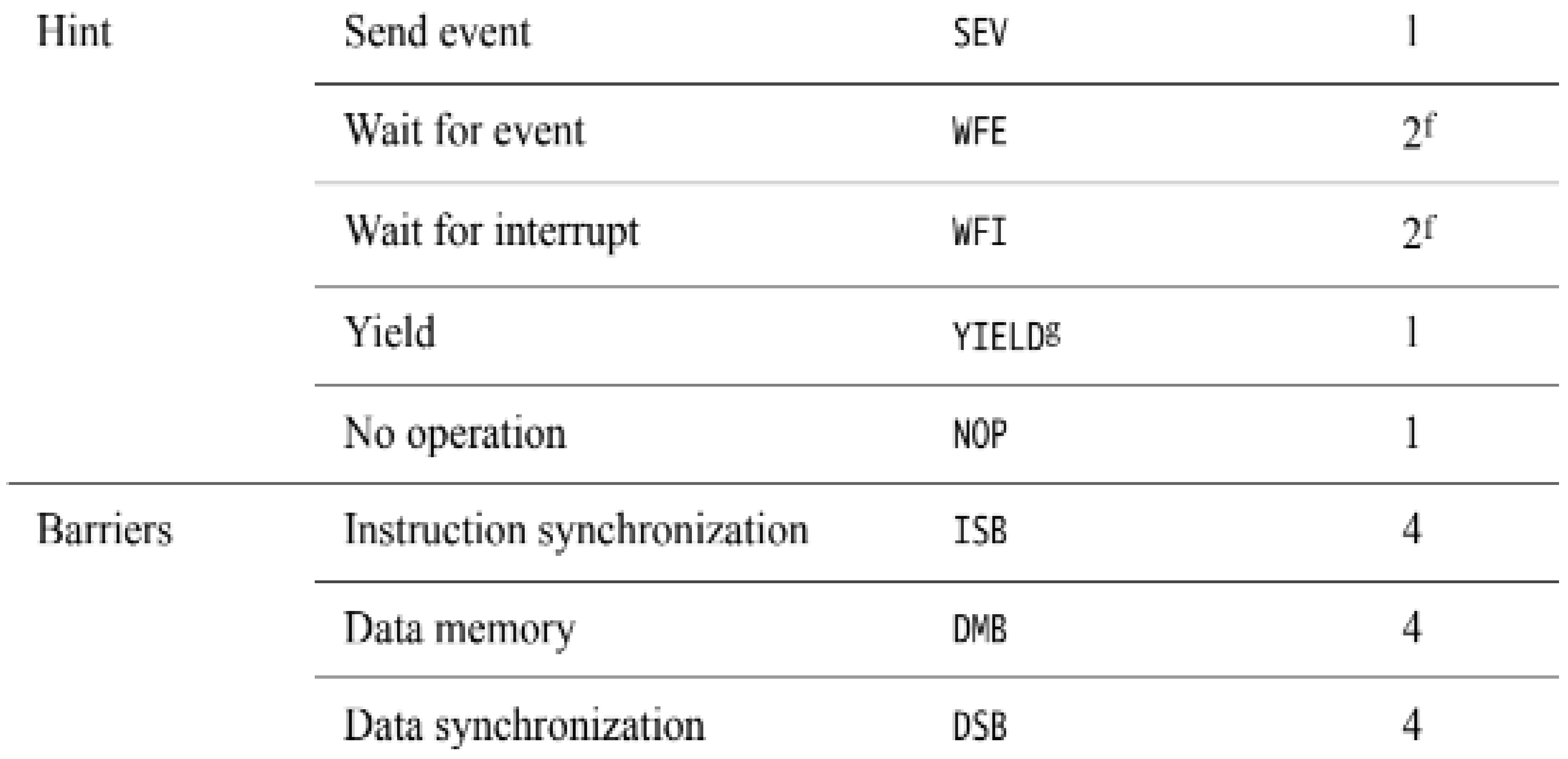
Push	Push	`PUSH {<loreglist>}`	1+N[b]
	Push with link register	`PUSH {<loreglist>, LR}`	1+N[b]
Pop	Pop	`POP {<loreglist>}`	1+N[b]
	Pop and return	`POP {<loreglist>, PC}`	4+N[c]
Branch	Conditional	`B<cc> <label>`	1 or 3[d]
	Unconditional	`B <label>`	3
	With link	`BL <label>`	4
	With exchange	`BX Rm`	3
	With link and exchange	`BLX Rm`	3
Extend	Signed halfword to word	`SXTH Rd, Rm`	1
	Signed byte to word	`SXTB Rd, Rm`	1
	Unsigned halfword	`UXTH Rd, Rm`	1

Instruction Set



Operation	Description	Assembler	Cycles
Extend	Unsigned byte	UXTB Rd, Rm	1
Reverse	Bytes in word	REV Rd, Rm	1
	Bytes in both halfwords	REV16 Rd, Rm	1
	Signed bottom half word	REVSH Rd, Rm	1
State change	Supervisor Call	SVC #<imm>	- [e]
	Disable interrupts	CPSID i	1
	Enable interrupts	CPSIE i	1
	Read special register	MRS Rd, <specreg>	4
	Write special register	MSR <specreg>, Rn	4
	Breakpoint	BKPT #<imm>	- [e]

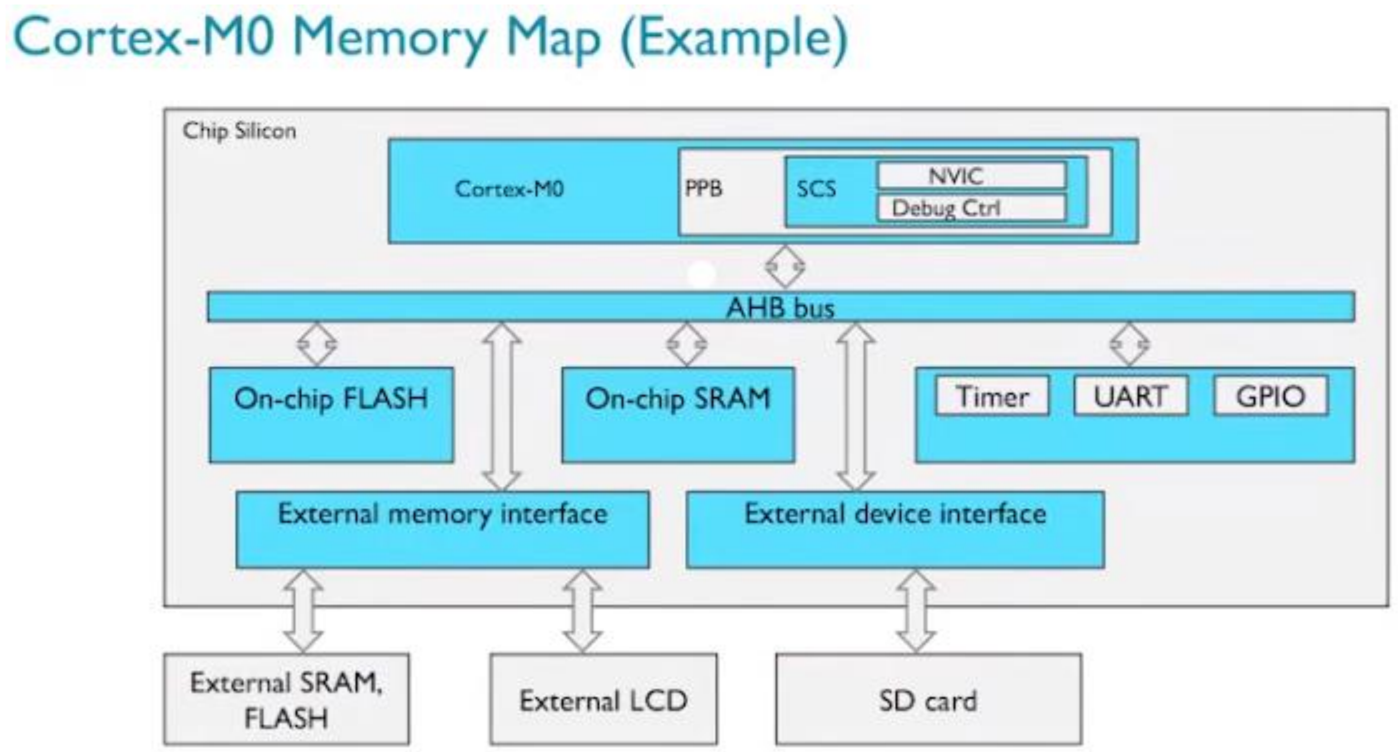
Instruction Set



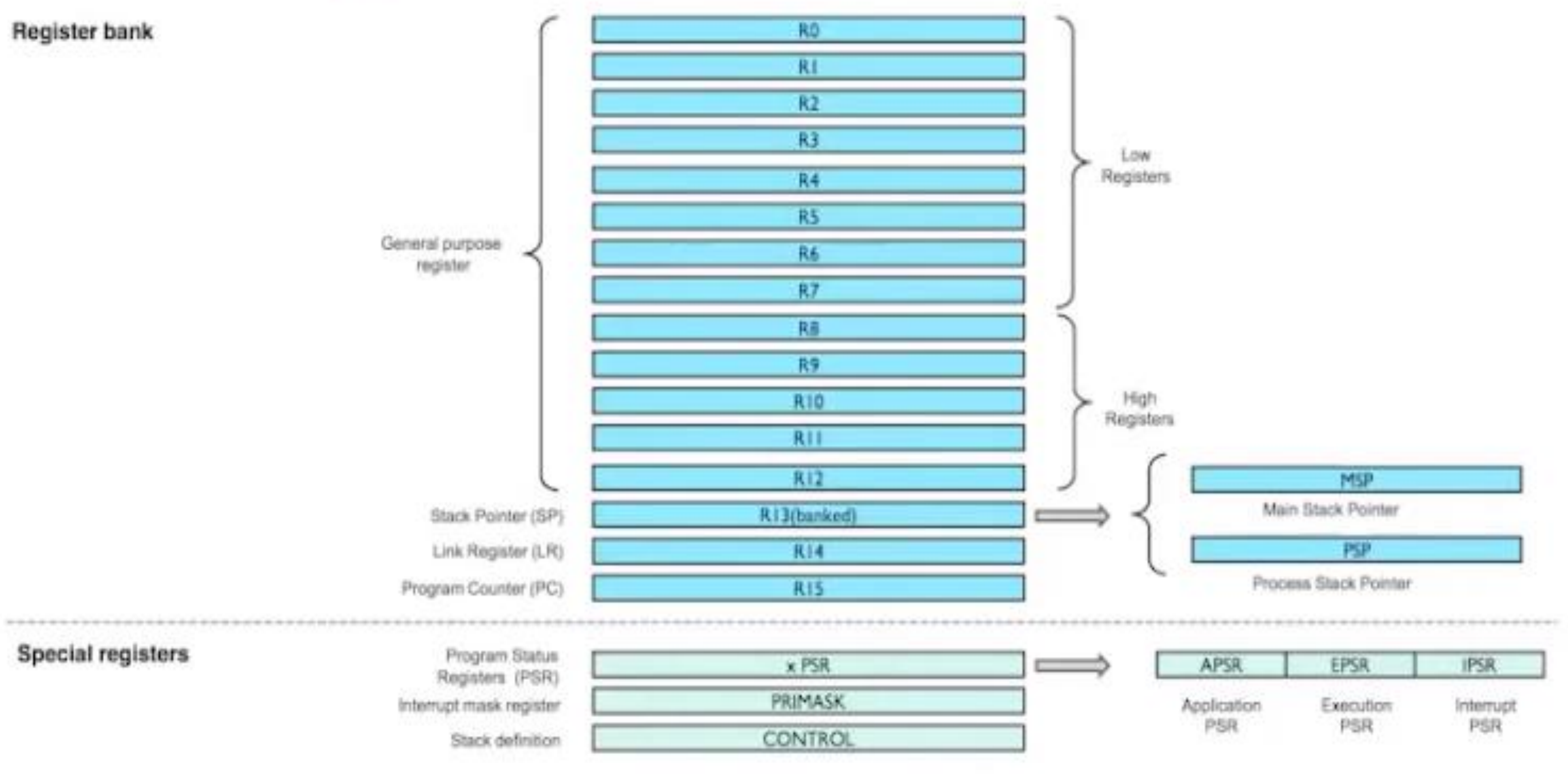
Hint	Send event	SEV	1
	Wait for event	WFE	2[f]
	Wait for interrupt	WFI	2[f]
	Yield	YIELD[g]	1
	No operation	NOP	1
Barriers	Instruction synchronization	ISB	4
	Data memory	DMB	4
	Data synchronization	DSB	4

Memory Systems

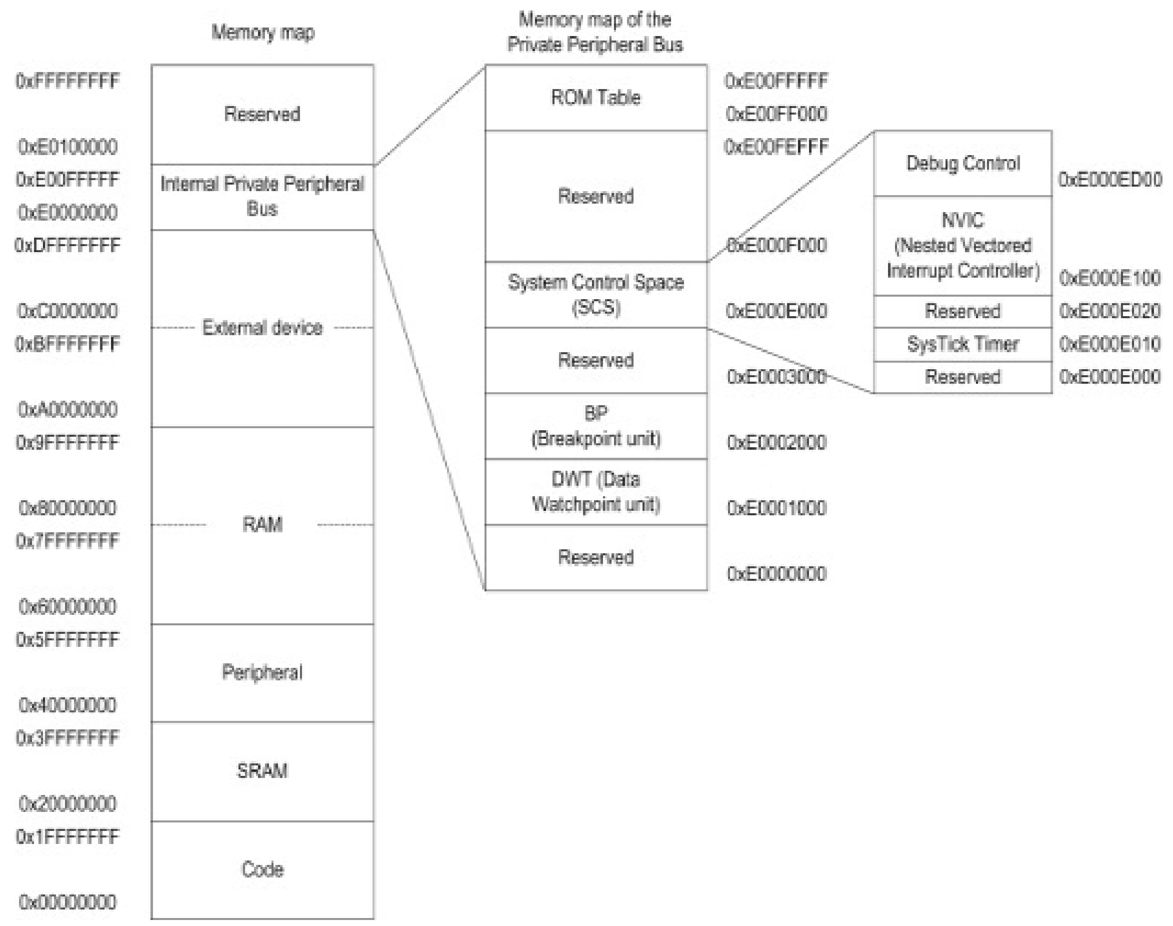
Memory System Overview



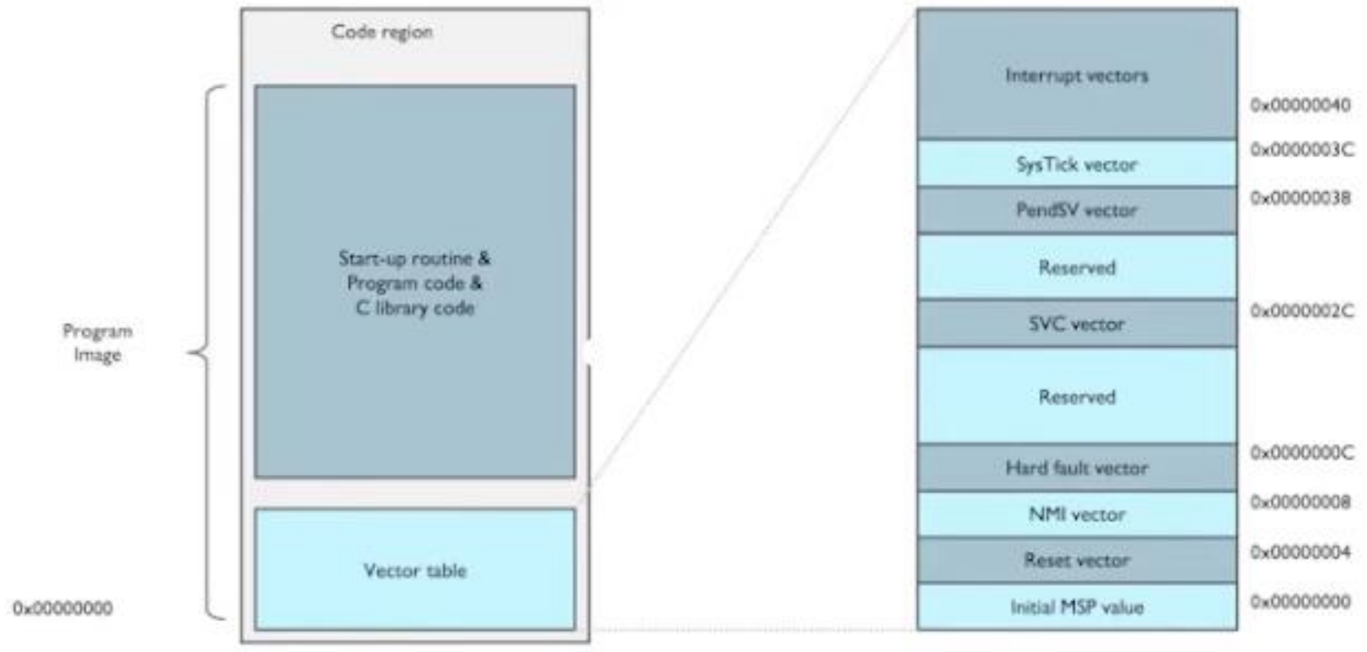
Cortex-M0 Registers



MEMORY OVERVIEW



Cortex-M0 Program Image



SVC- Super visitor call
User Can trigger exceptions. Exceptions run in privileged mode.
This is used to work on exceptions in Unprivileged mode.

Memory space

- The 4 GB memory space of the Cortex®-M0 processors is architecturally divided into a number of regions

Code Region (0x00000000–0x1FFFFFFF)

- The size of the code region is 512 MB.
- Used to store program code
- Contains the initial exception vector table at address 0x00000000

SRAM Region (0x20000000–0x3FFFFFFF)

- The SRAM region is located in the next 512 MB of the memory map.
- Primarily used to store data, including stack.
- Also used to store program codes.
- For example, in some cases you might want to copy program codes from slow external memory to the SRAM and execute it from there.

Peripheral Region (0x40000000–0x5FFFFFFF)

- The Peripheral region also has the size of 512 MB.
- It is primarily used for peripherals, and can also be used for data storage.
- Program execution is not allowed in the Peripheral region.
- The peripherals connected to this memory region can either be AHB-Lite peripheral(Master Slave) or APB peripherals (via a bus bridge).

RAM Region (0x60000000–0x9FFFFFFF)

- The RAM region consists of two 512 MB blocks, which results in total of 1 GB space.
- Both 512 MB memory blocks are primarily used to stored data, and in most cases the RAM region can be used as a 1 GB continuous memory space.
- The RAM region can also be used for program code execution.
- The only differences between the two halves of the RAM region are the memory attributes, which might cause differences in caching behavior if a system level cache (level-2 cache) is used.

Device Region (0xA0000000–0xDFFFFFFF)

- The external device region consists of two 512 MB memory blocks, which results in a total of 1 GB space.
- Both 512 MB memory blocks are primarily used for peripherals and I/O usages.
- The device region does not allow program execution, but it can be used for general data storage.
- Similar to the RAM region, the two halves of the device region have different memory attributes.

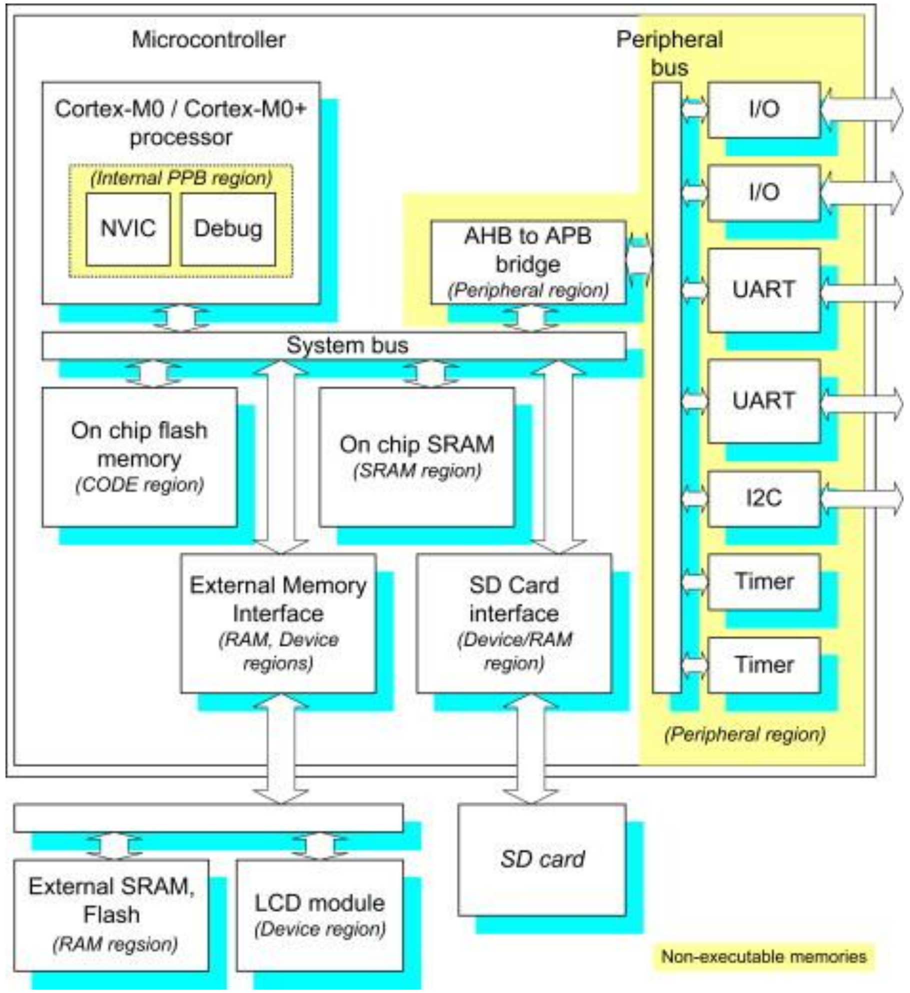
Internal Private Peripheral Bus (0xE0000000–0xE00FFFFF)

- The internal Private Peripheral Bus (PPB) memory space is allocated for peripherals inside the processor, such as the interrupt controller Vectored Interrupt Controller (NVIC), as well as the debug components.
- The internal PPB memory space is 1 MB in size, and program execution is not allowed in this memory range.
- Within the PPB memory range, a special range of memory is defined as the System Control Space (SCS).
- The SCS address is from 0xE000E000 to 0xE000EFFF. It contains the interrupt control registers, system control registers, debug control registers, etc. The NVIC registers are part of the SCS memory space. The SCS also contains an optional timer called the SysTick.

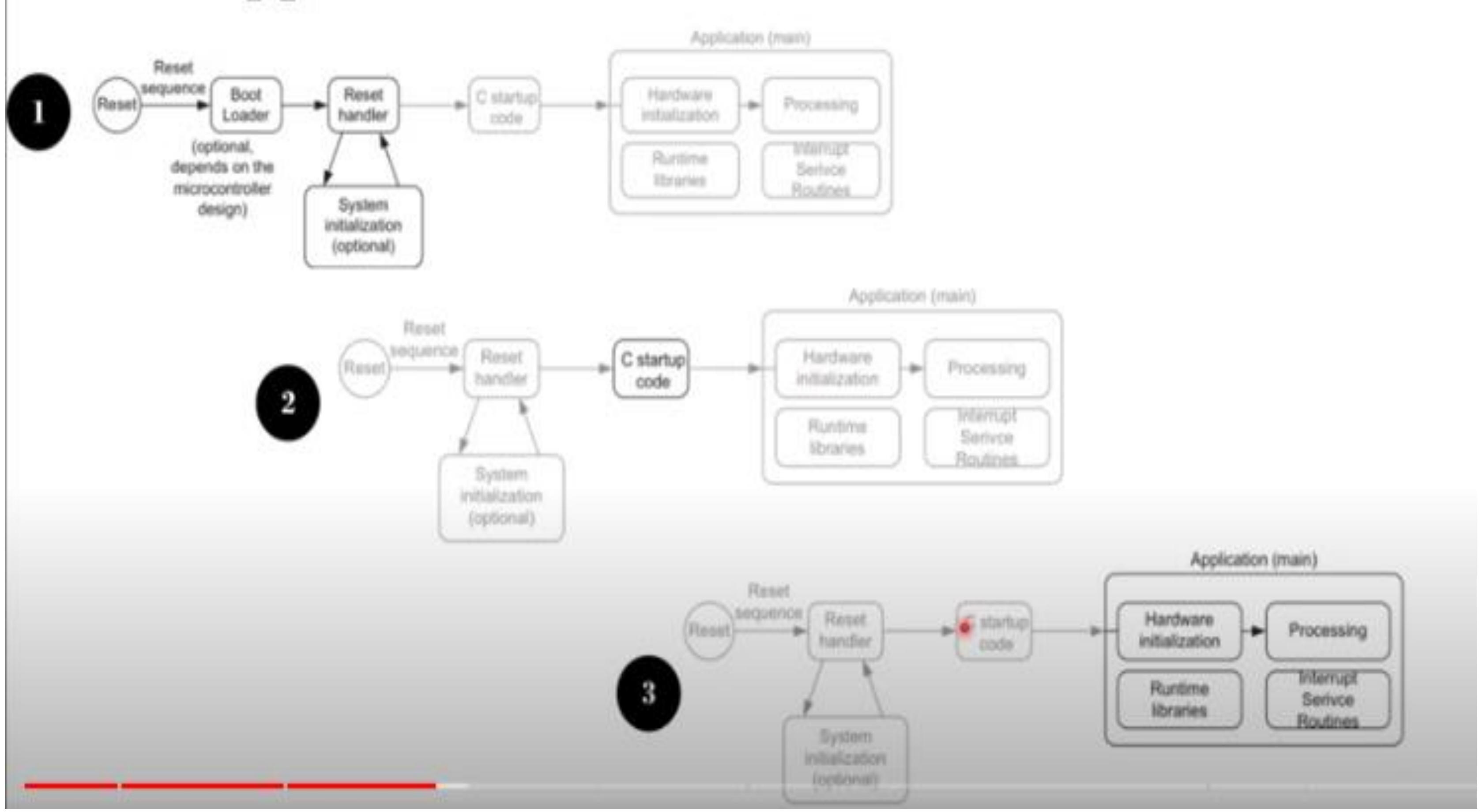
Reserved Memory Space (0xE0100000–0xFFFFFFFF)

- The last section of the memory map is a 512 MB reserved memory space.
- This may be used in some microcontrollers for microcontroller vendor specific usages.

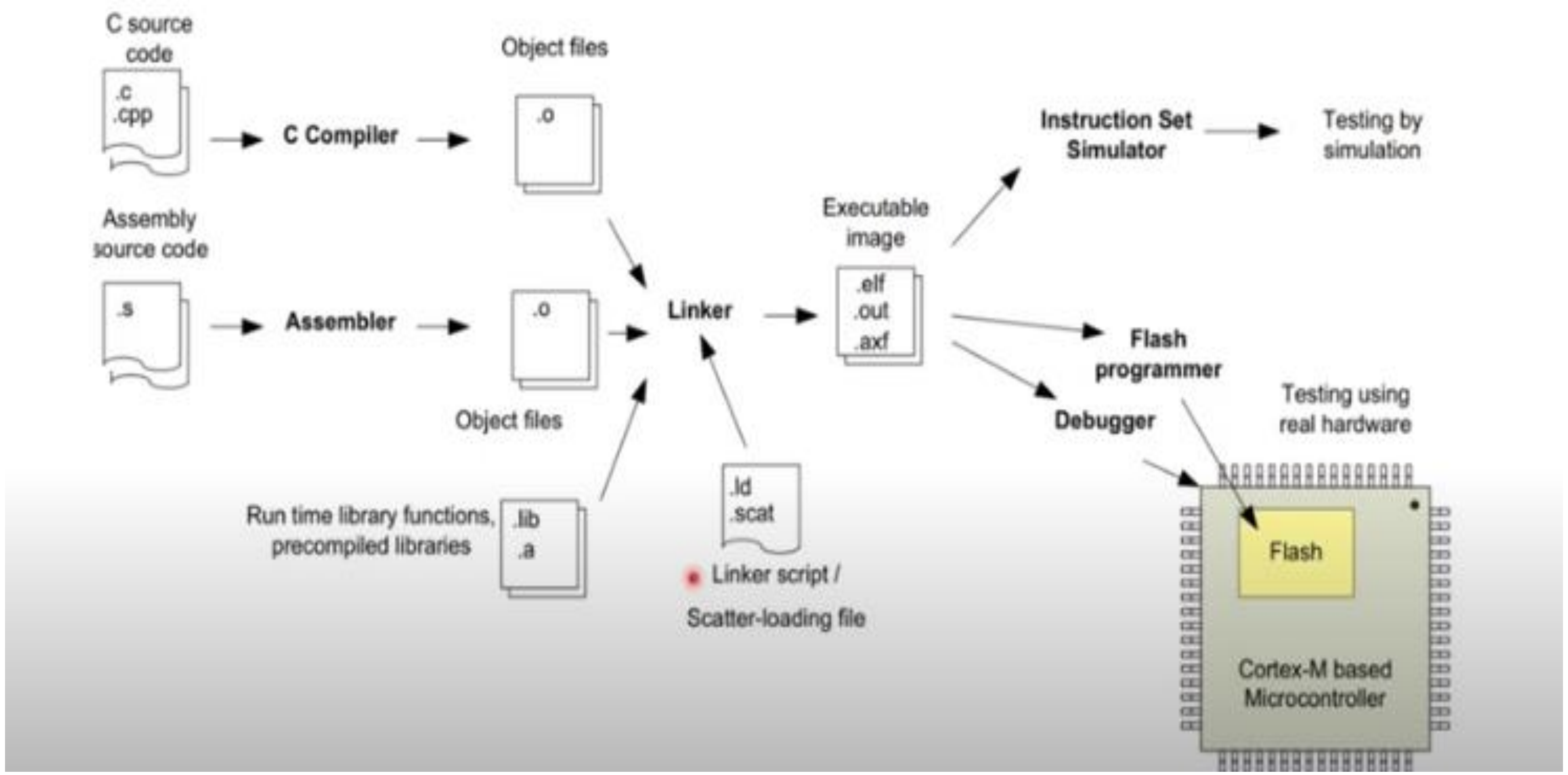
CORTEX M0 Overview



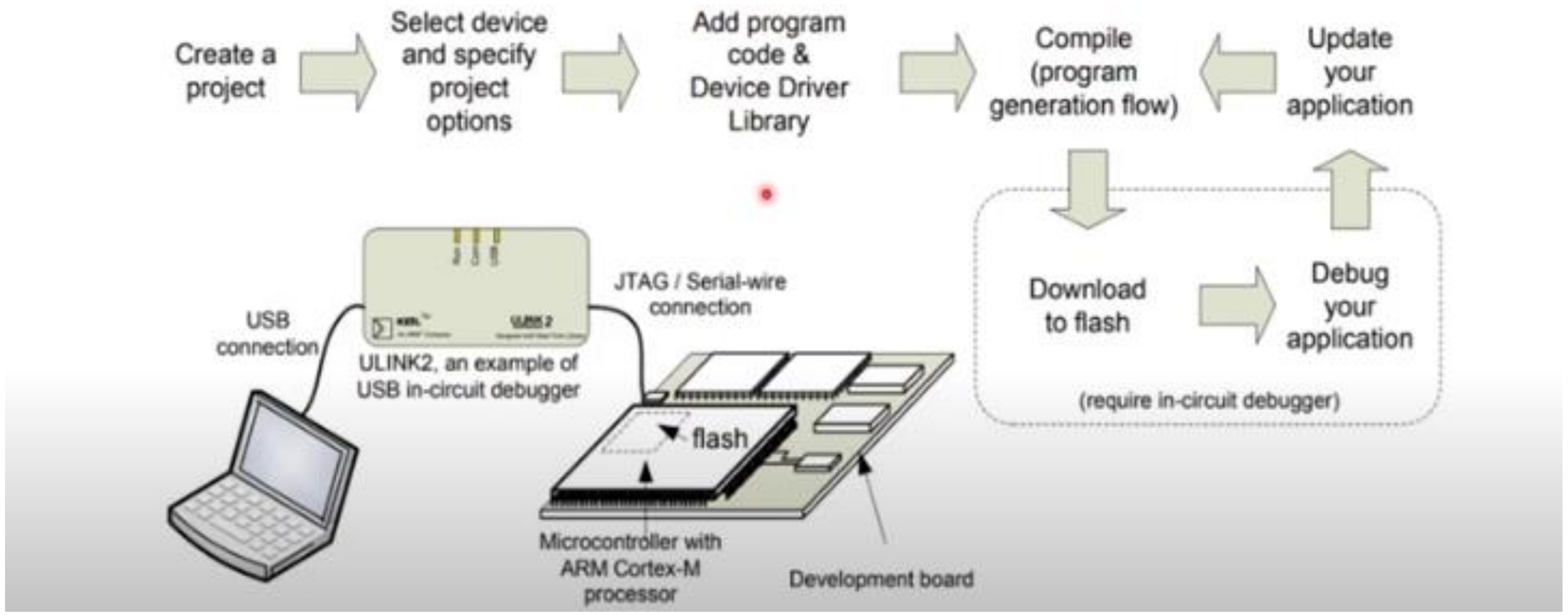
What happens when a microcontroller starts?



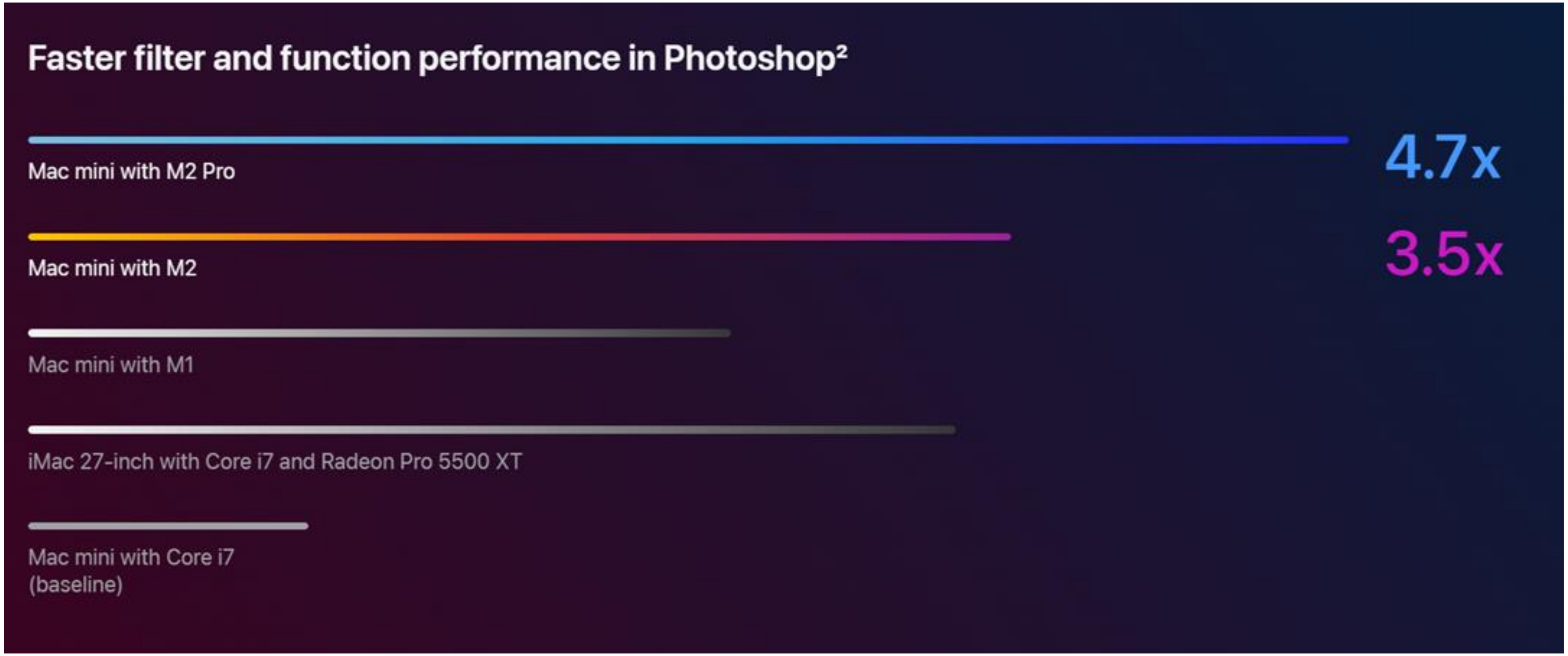
Typical Program Generation Flow



Typical Development Flow


Create a project
Select device and specify project options
Add program code & Device Driver Library
Compile (program generation flow)
Update your application
Download to flash
Debug your application
(require in-circuit debugger)
USB connection
ULINK2, an example of USB in-circuit debugger
JTAG / Serial-wire connection
flash
Microcontroller with ARM Cortex-M processor
Development board

Performance between M1,M2 and Intel



ASSESSMENT

1. In which year Apple moved from Intel to ARM?

a) 2020 b) 2015 c)2021 d)2005

2. What is meant by RISC architecture?

Reference

https://microcontrollerslab.com/nested-vectored-interrupt-controller-nvic-arm-cortex-m/

https://www.sciencedirect.com/topics/engineering/memory-space

Thank you