

arm

Arm and Arm Processors

Learning Outcomes

At the end of this module, you will be able to:

- Identify the characteristics of Arm processor classes and their corresponding instruction set architecture.
- Describe the properties of the Armv7-A architecture, including the AAPCS, operating modes, registers, memory model, and Virtualization Extensions.
- Outline the properties of the Arm Cortex-A9 Processor.
- Identify what Arm Neon technology is and its usage.

Arm Products

- Processors
 - Cortex-A, R, M, SecurCore
- System IP
 - CoreLink, CoreSight, AMBA Design Tools
- Multimedia
 - Mali graphics, video, display
- Physical IP
 - Artisan Logic IP, Interface IP, Memory IP, DesignStart
- Tools
 - Software tools (Development Studio, Keil MDK), debug adapters, models, boards
- Support
 - Training, documentation, Arm Connected Community

Arm Processor Families

- Cortex-A series (advanced application)
 - High-performance processors for open OSs
 - Applications include smartphones, digital TV, server solutions, and home gateways.
- Cortex-R series (real-time)
 - Exceptional performance for real-time applications
 - Applications include automotive braking systems and 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.

Arm Processor Families

- SecurCore series
 - High-security applications such as smartcards and e-government
- Neoverse
 - High performance efficiency for cloud, infrastructure, and AI/ML-accelerated applications
- Classic processors
 - Include Arm7, Arm9, and Arm11 families

Arm Cortex-A Series Family

- Cortex-A series: Cortex-A5, A7, A8, A9, A12, A15, A17, A53, A57
- High-performance application processors
 - Run rich OSs, multicore technology, 32-bit and 64-bit supports
- Applications
 - Mobile computing Netbook, tablet, eReader
 - Mobile handset Smartphones, feature phones, wearables
 - Digital home Set-top box, digital TV, Blu-Ray player, gaming consoles
 - Automotive Infotainment, navigation
 - Enterprise industrial printers, routers, wireless base-stations, VOIP phones and equipment
 - Wireless infrastructure

Arm Cortex-M Series Family

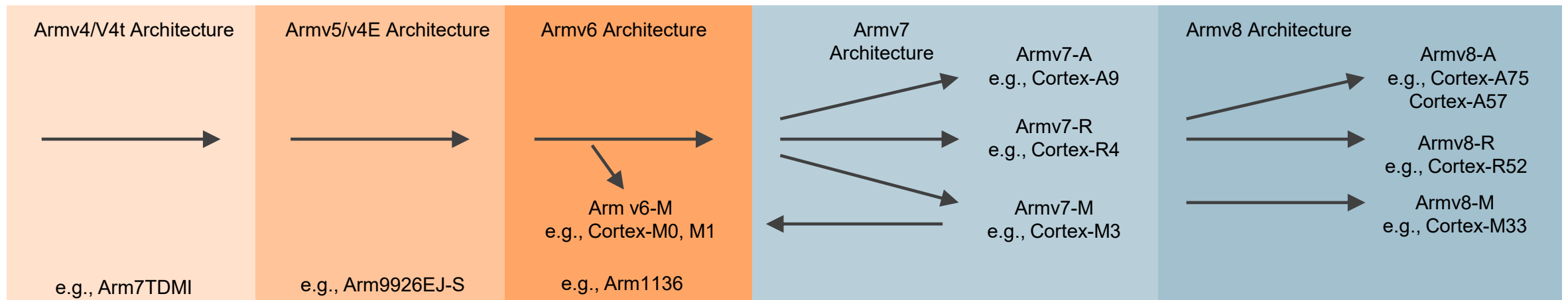
- Cortex-M series: Cortex-M0, M0+, M1, M3, M4, M7
- Low-power processor for embedded microcontrollers
 - Energy-efficiency
 - Smaller code
 - Ease of use
- Applications
 - Internet of Things, connectivity, smart metering, human interface devices, automotive and industrial control systems, domestic household appliances, consumer products, and medical instrumentation

Sample Arm Processors

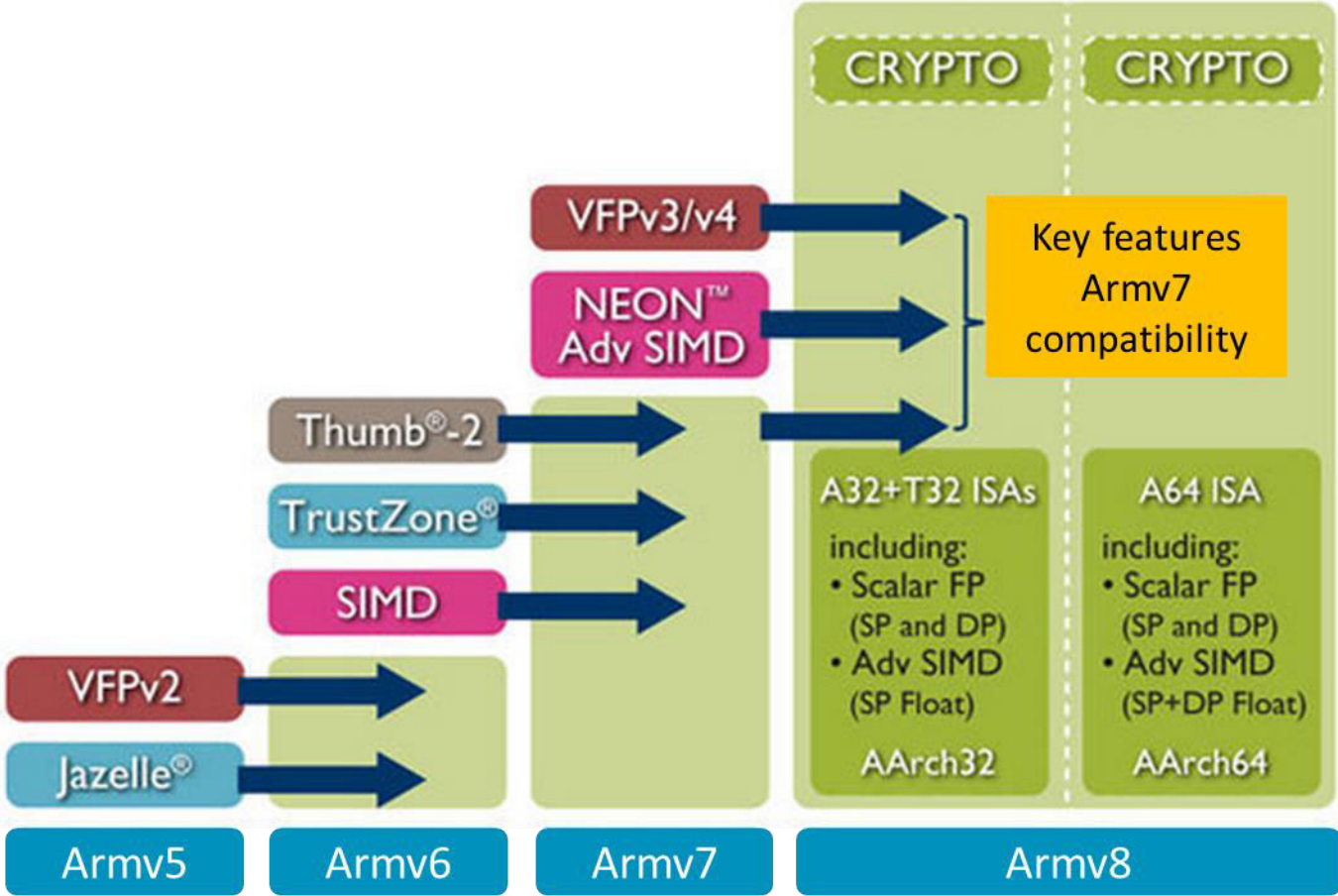
Processor Class	ARM Processor	Architecture	Performance DMIPS/MHz	ARM instructions	Thumb-2 instructions	Jazelle-DBX JAVA by tcode execution	Jazelle-RCT Dynamic compiler support	TrustZone security	E' DSP extensions	Media SIMD extensions	NEON SIMD extensions	Floating point	Caches	Memory Management Unit (MMU)	Memory Protection Unit (MPU)	Hardware Cache coherency	Target OS	Trace support
Arm7 Arm9 Arm11	ARM7TDMI/ARM7TDMI-S	ARMv4-T	0.95	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	Real Time	✓
	ARM946E-S	ARMv5-E	1.23	✓	✗	✗	✗	✗	✓	✗	✗	Optional	✓	✗	✓	✗	Real Time	✓
	ARM926EJ-S	ARMv5-EJ	1.06	✓	✗	✓	✗	✗	✓	✗	✗	Optional	✓	✓	✗	✗	Platform	✓
	ARM1136J-S	ARMv6	1.18	✓	✗	✓	✗	✗	✓	✓	✗	Optional	✓	✓	✗	✗	Platform	✓
	ARM1156T2-S	ARMv6-T2	1.45	✓	✓	✗	✗	✗	✓	✓	✗	Optional	✓	✗	✓	✗	Real Time	✓
	ARM1176JZ-S	ARMv6-Z	1.26	✓	✗	✓	✗	✓	✓	✓	✗	Optional	✓	✓	✗	✗	Platform	✓
	ARM11 MPCore	ARMv6	1.25	✓	✗	✓	✗	✗	✓	✓	✗	Optional	✓	✓	✗	✓	Platform/SMP	✓
M	Cortex-M0+	ARMv6-M	0.90	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	Real Time	✗
	Cortex-M0	ARMv6-M	0.90	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	Real Time	✗
	Cortex-M1	ARMv6-M	0.79	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	Real Time	✗
	Cortex-M3	ARMv7-M	1.25	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	Optional	✗	Real Time	Instruction only
	Cortex-M4	ARMv7-ME	1.25	✓	✓	✗	✗	✗	✓	✓	✗	Optional	✓	✗	Optional	✗	Real Time	✓
R	Cortex-A5 MPCore	ARMv7+MP	1.58	✓	✓	✓	✓	✓	✓	✓	Optional	Optional	✓	✓	✗	✓+ACP	Platform/SMP	✓
	Cortex-R4	ARMv7	1.66	✓	✓	✗	✗	✗	✓	✓	✗	Optional	✓	✗	Optional	✗	Real Time	✓
	Cortex-R5	ARMv7	1.66	✓	✓	✗	✗	✗	✓	✓	✗	Optional	✓	✗	Optional	✗	Real Time	✓
	Cortex-R7	ARMv7	2.53	✓	✓	✗	✗	✗	✓	✓	✗	Optional	✓	✗	Optional	✗	Real Time	✓
	Cortex-A7	ARMv7+MP	1.90	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓+ACP	Platform/SMP	PTM
A	Cortex-A8	ARMv7	2.07	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	Platform	✓
	Cortex-A9 MPCore	ARMv7+MP	2.50	✓	✓	✓	✓	✓	✓	✓	Optional	Optional	✓	✓	✗	✓+ACP	Platform/SMP	PTM
	Cortex-A15 MPCore	ARMv7+MP	2.50	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓+ACP	Platform/SMP	PTM
	Cortex-A53	ARMv8	2.3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓+ACP	Platform/SMP	PTM
	Cortex-A57	ARMv8	>4.0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓+ACP	Platform/SMP	PTM

Arm Processors v 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 the processor's Technical Reference Manual



Arm Architectures

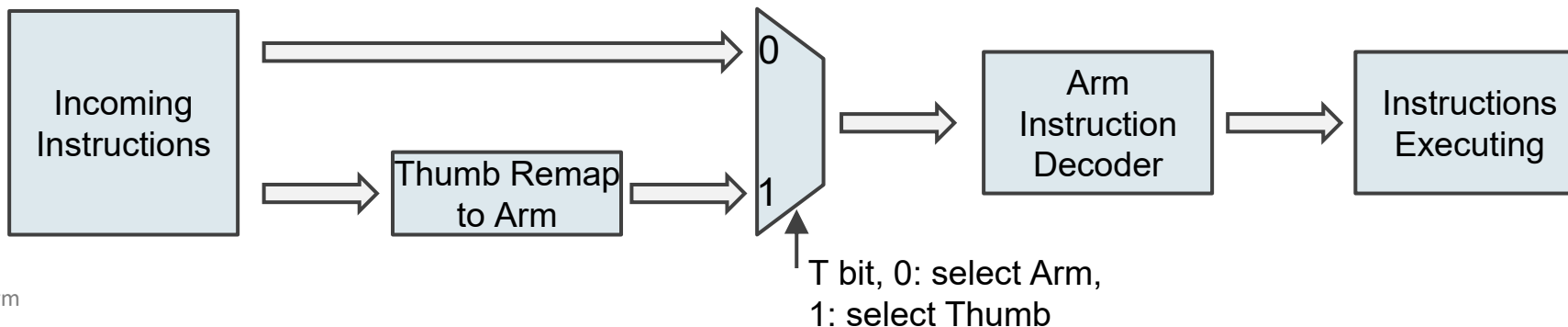


Which Architecture Is My Processor?

Processor core	Architecture
Arm7TDMI family	v4T
Arm9TDMI family	v4T
Arm9E family	v5TE, v5TEJ
Arm10E family	v5TE, v5TEJ
Arm11 family	v6
• Arm1136J(F)-S	v6
• Arm1156T2(F)-S	v6T2
• Arm1176JZ(F)-S	v6Z
• Arm11 MPCore	v6k
Cortex family	
• Arm Cortex-A57	v8-A (64-bit, highest performance)
• Arm Cortex-A53	v8-A (64-bit)
• Arm Cortex-A15	v7-A (with security and virtualization extensions)
• Arm Cortex-A9	v7-A (with security extensions)
• Arm Cortex-A8	v7-A (with security extensions)
• Arm Cortex-A7	v7-A (with security and virtualization extensions)
• Arm Cortex-A5	v7-A (with security extensions)
• Arm Cortex-R5	v7-R
• Arm Cortex-R4	v7-R
• Arm Cortex-M4	v7-M
• Arm Cortex-M3	v7-M
• Arm Cortex-M1	v6-M (16-bit Thumb, except for system instructions)
• Arm Cortex-M0	v6-M (16-bit Thumb, except for system instructions)

Arm and Thumb Instruction Sets

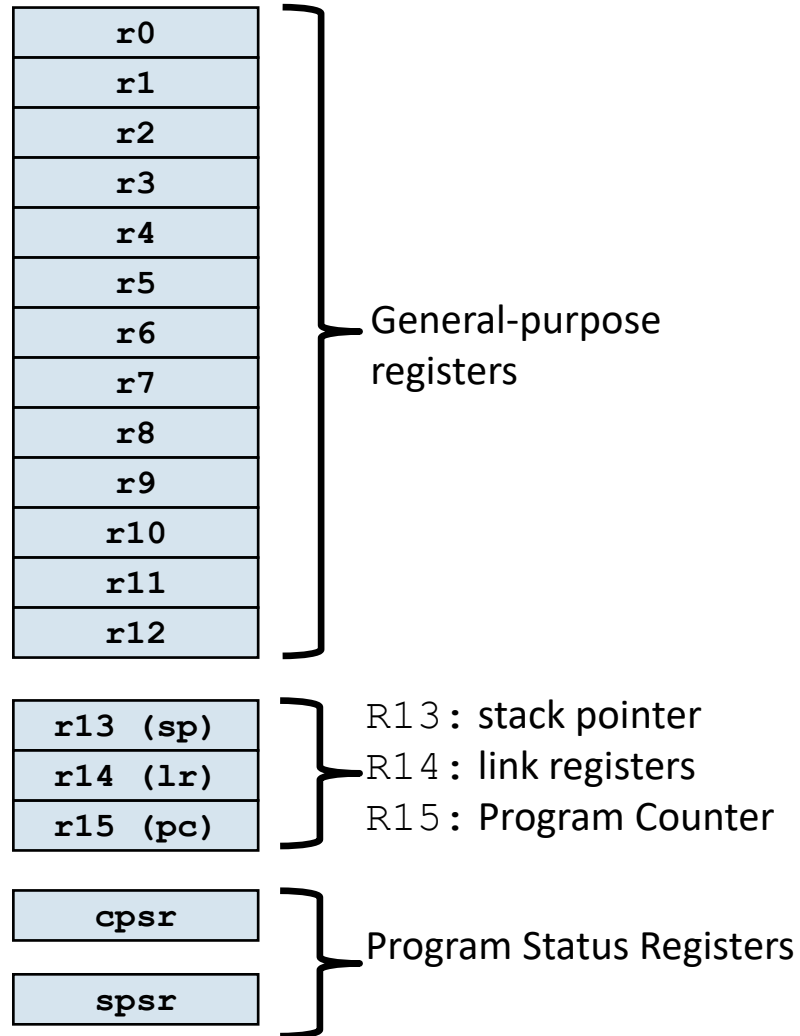
- Early Arm processors
 - 32-bit instruction set, called the Arm instruction set
 - Powerful and good performance
 - Larger program memory compared to 8-bit and 16-bit processors
 - Larger power consumption
- Thumb-1 instruction set
 - 16-bit instruction set, first used in the Arm7TDMI processor in 1995
 - Provides a subset of the Arm instructions, giving better code density compared to 32-bit RISC architecture
 - Code size is reduced by ~30%, but performance is also reduced by ~20%.
 - Can be used together with Arm instructions using a multiplexer



Arm and Thumb Instruction Sets

- Thumb-2 instruction set
 - Consists of both 32-bit Thumb and original 16-bit Thumb-1 instruction sets
 - Compared to the 32-bit Arm instruction set, code size is reduced by ~26%, while maintaining similar performance
- Thumb Execution Environment (ThumbEE) instruction set
 - Based on Thumb
 - With some changes and additions to make it a better target for dynamically generated code, i.e., code compiled on the device either shortly before or during execution
- Armv7-A architecture
 - Based on Thumb-2 and ThumbEE

The Arm Register Set



- Sixteen general-purpose registers
- Some of the registers have special significance.
 - R15: Program Counter (`pc`)
 - R14: Link register (`lr`)
 - R13: Stack pointer (`sp`)
- There are also two status registers.
 - Current Program Status Register (CPSR)
 - Saved Program Status Register (SPSR)
 - Only present in exception modes
 - Only accessible by some instructions

Assembler Syntax

- Data processing instructions

- **<operation><condition> Rd, Rm, <op2>**

- ADDEQ r4, r5, r6
- SUB r5, r7, #4
- MOV r4, #7

- Memory access instructions

- **<operation><size> Rd, [<address>]**

- LDR r0, [r6, #4]
- STRB r4, [r7], #8
-

- **<operation><addressing mode> <Rn>!, <registers list>**

- LDMIA r0, {r1, r2, r7}
- STMFD sp!, {r4-r11, lr}

- Program flow instructions

- **<branch> <label>**

- BL foo
- B bar

AAPCS

Arguments into function
Result(s) from function
otherwise corruptible
(Additional parameters
passed on stack)

r0	(a1)
r1	(a2)
r2	(a3)
r3	(a4)

Register variables
Must be preserved

r4	(v1)
r5	(v2)
r6	(v3)
r7	(v4)
r8	(v5)
r9	(v6/SB)
r10	(v7)
r11	(v8)

Scratch register
(corruptible)

r12

Stack Pointer
Link Register
Program Counter

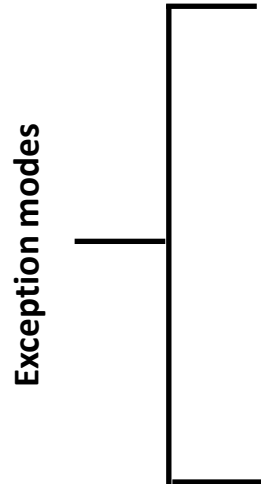
r13	(sp)
r14	(lr)
r15	(pc)

- The compiler has a set of rules known as a Procedure Call Standard that determines how to pass parameters to a function (see AAPCS).
- CPSR flags may be corrupted by function call.
- Assembler code that links with compiled code must follow the AAPCS at external interfaces.
- The AAPCS is part of the ABI for the Arm architecture.
- r9 is used as the static base if the RWPI option selected.

- sp should always be 8-byte (2 words) aligned.
- r14 can be used as a temporary register once value stacked.

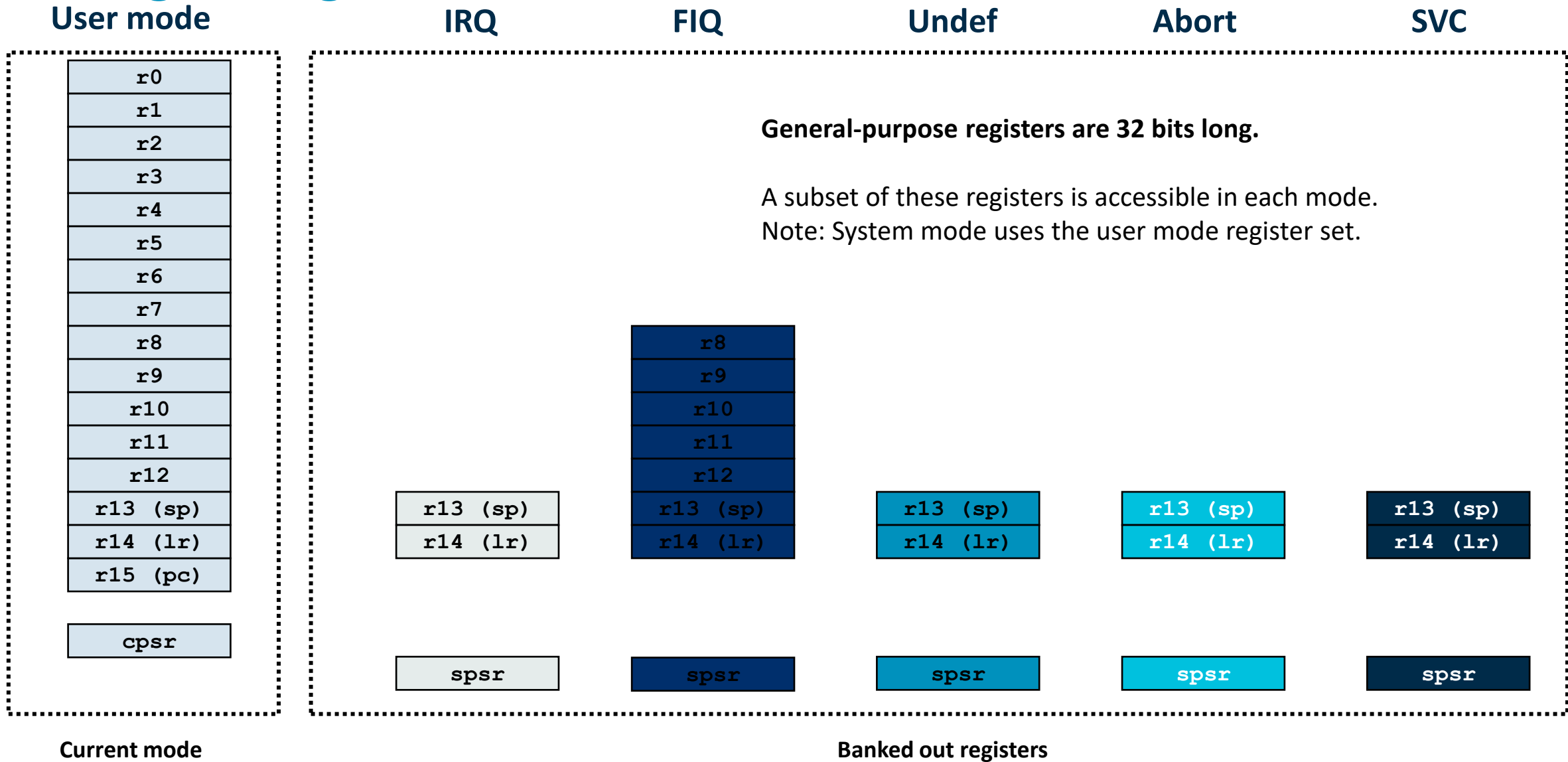
Processor Modes

- The Arm processor has seven basic operating modes:
 - Each mode has access to its own stack space and a different subset of registers.
 - Some operations can only be carried out in a privileged mode.



Mode	Description	Privilege level	
Supervisor	Entered on reset and when a supervisor call instruction (SVC) is executed	PL1	Privileged modes
FIQ	Entered when a high-priority (fast) interrupt is raised	PL1	
IRQ	Entered when a normal-priority interrupt is raised	PL1	
Abort	Used to handle memory access violations	PL1	
Undef	Used to handle undefined instructions	PL1	
System	Privileged mode using the same registers as user mode	PL1	
User	Mode under which most applications/OS tasks run	PL0	Unprivileged mode

Banking of Registers

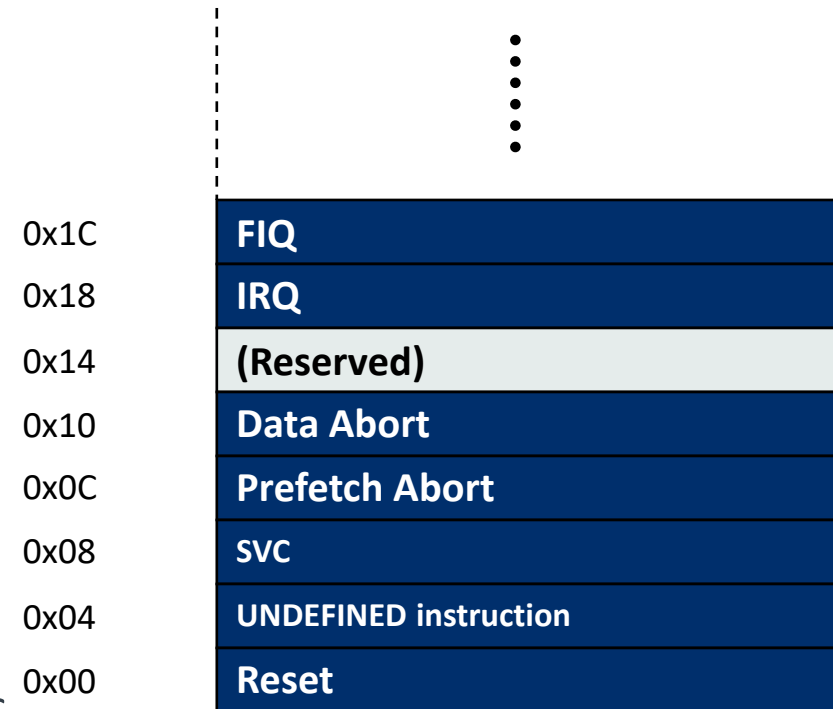


Exceptions

- When an exception occurs:
 - It causes entry into a processor mode that executes software at PL1 or PL2.
 - It causes the execution of a software handler for the exception.
- Exception includes:
 - Resets
 - Interrupts
 - Memory system aborts
 - Undefined instructions
 - SVCs, secure monitor calls (SMCs), and hypervisor calls (HVCs)
- Processor execution is forced to the exception vector (an address) corresponding to that type of exception.
- Vector table:
 - A set of eight consecutive vectors
 - World aligned memory addresses starting at an exception base address

Vector Table

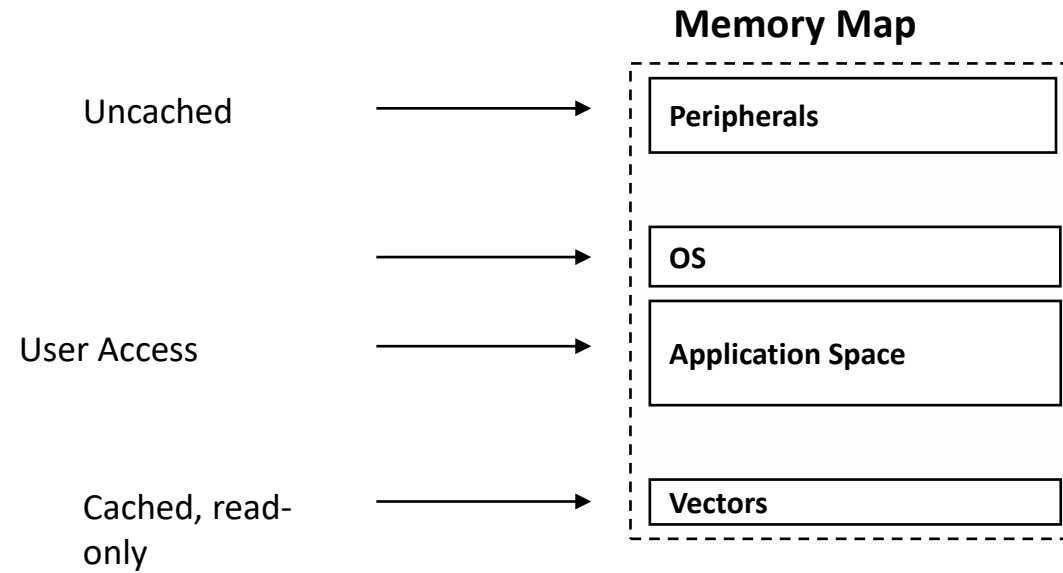
- A vector table has one entry per exception type.
- Table entries contain instructions, not addresses.
 - 1 × Arm instruction
 - 2 × 16-bit Thumb instructions
 - 1 × 32-bit Thumb instruction
 - Arm/Thumb controlled by `SCTLR.TE` bit
- The vector table address is configurable.
 - `0x0` or `0xFFFF0000`
 - `SCTLR.V` bit / `VINITHI` signal
 - The security extensions add support for other addresses.
 - Vector base address registers



Vector Table

At reset, the vector table can be at `0x0` or `0xFFFF0000`.

Memory Model



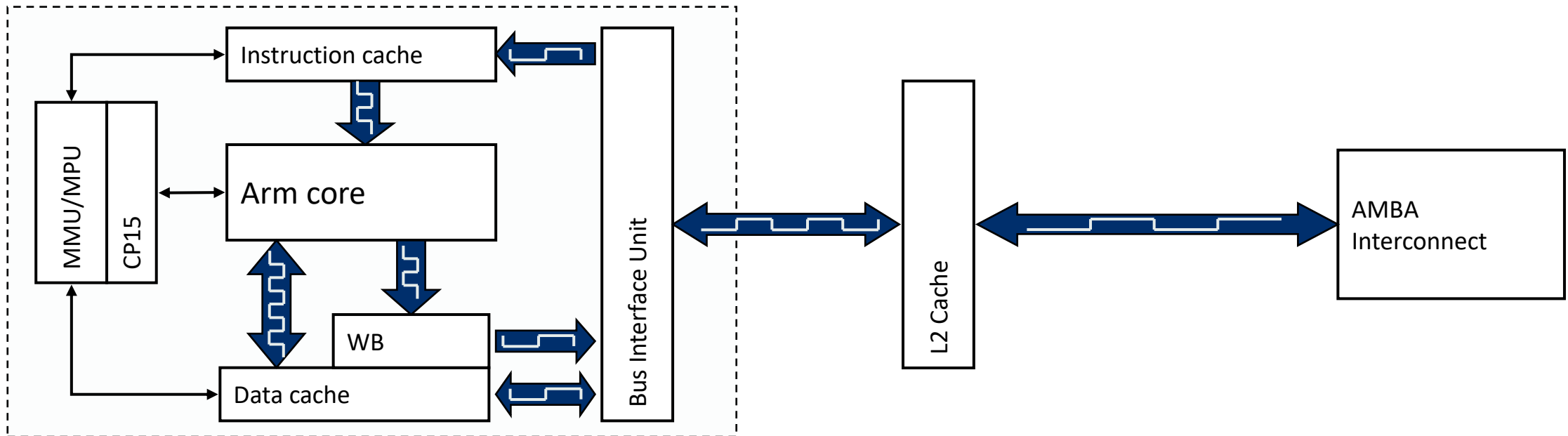
- A system includes different memories and peripherals.
 - The processor needs to be told how it should access different devices.
- For each address region:
 - Access permissions
 - Read/write permissions for user/privileged modes
 - Memory types
 - Caching/buffering and access ordering rules for memory accesses

Memory Types

- In Armv6/Armv7, address locations must be described in terms of a type.
- The type tells the processor how it can access that location:
 - Memory access ordering rules
 - Caching and buffering behavior
 - Speculation
- There are three mutually exclusive memory type attributes:
 - Normal: Data and instructions
 - Device: Devices/peripherals
 - Strongly ordered: Device/peripherals, or data used by legacy code
- Normal and device memory allow additional attributes for specifying the cache policy and whether the region is shared.
 - For example, normal memory can be cached or non-cached.

Example: Cached Arm Macrocell

- For memory management, an Arm core can include either an MMU or MPU.
- Memory Management Unit (MMU)
 - Implements Virtual Memory System Architecture (VMSA)
- Memory Protection Unit (MPU)
 - Implements physical memory system architecture (PMSA)



Data Alignment

- Armv6/v7 data alignment:
 - Data accesses can be unaligned.
 - Only a subset of load/store instructions support unaligned accesses.
 - Unaligned accesses are only allowed to addresses marked as normal.
 - The load/store unit will access memory with aligned memory accesses and make the data available to the CPU.
- Instructions are aligned as follows:
 - Arm instructions are word aligned.
 - Thumb and ThumbEE instructions are halfword-aligned.
 - Java bytecodes are byte-aligned.
- Arm processors are little-endian.
 - But can be configured to access big-endian memory systems

Endianness

- Endianness determines how contents of registers relate to the contents of memory.
 - Arm registers are word (4 bytes) width.
 - Arm addresses memory as a sequence of bytes.
- Arm processors are little-endian.
 - But can be configured to access big-endian memory systems.

Little-endian memory system

- Least significant byte is at lowest address.

Big-endian memory system

- Most significant byte is at lowest address.

- Arm supports three models of endianness.
 - LElittle-endian
 - BE-32 word-invariant big-endian (dropped in architecture v7)
 - BE-8 byte-invariant big-endian (introduced in architecture v6)

PMU

- Armv6 & Armv7-A/R processors include a Performance Monitoring Unit (PMU).
- The PMU provides a non-intrusive method of collecting execution information from the core.
 - Enabling the PMU does not change the timing of the core.
- PMU accessed through
 - CP15 (mandatory)
 - A memory-mapped interface (optional)
 - An external debug interface (optional)
- The PMU provides:
 - Cycle counter: counts execution cycles (optional 1/64 divider)
 - Programmable event counters
 - The number of counters and available events vary between cores.
 - The PMU can be configured to generate interrupts if a counter overflows.
 - Interrupt signals are an output from the core.
 - Need to be connected to the system's interrupt controller.

Coprocessors

- On earlier Arm processors, additional coprocessors could be added to expand the Arm instruction set.
- Newer processors do not allow user-defined coprocessors:
 - Usually better for system designers to use memory-mapped peripherals
 - Easier to implement, since coprocessors have to be tied in to the core pipeline
- Arm uses coprocessors for internal functions so as not to enforce a particular memory map:
 - System control coprocessor: cp15
 - Used for processor configuration: System ID, caches, MMU, TCMs, etc.
 - Debug coprocessor: cp14
 - Can be used to access debug control registers
 - VFP and Neon: cp10 and cp11

Architecture Extensions

- Architecture extensions to meet the changing needs of applications in new markets
- Security
 - The TrustZone
 - Additional operating mode, Monitor (Mon) mode, with associated banked registers and an additional secure operating state
- 40-bit physical addressing (LPAE)
 - Extension to the VMSEA7 virtual memory architecture
 - Enables the generation of 40-bit physical addresses from 32-bit virtual addresses
- Virtualization
 - Extra mode: Hypervisor mode, with associated banked registers
 - New Hyp exception to trap software accesses to hardware and configuration registers
- Advanced SIMD and floating-point: Both floating point (VFP) support and Advanced SIMD (Neon)
 - Can be implemented together, in which case they share a common register bank and some common instructions

TrustZone

- Processor provides two worlds: secure and normal.
 - Each world has its own vector table and page tables.
- “Monitor” mode acts as a gatekeeper for moving between worlds.
- Two physical address spaces, controlled by NS attribute
 - Secure (S) and Non-secure (NS)
 - S:0x8000 treated as different physical location from NS:0x8000
- Debug for Secure world code and data can be restricted.



Virtual Memory System Architecture (VMSA)

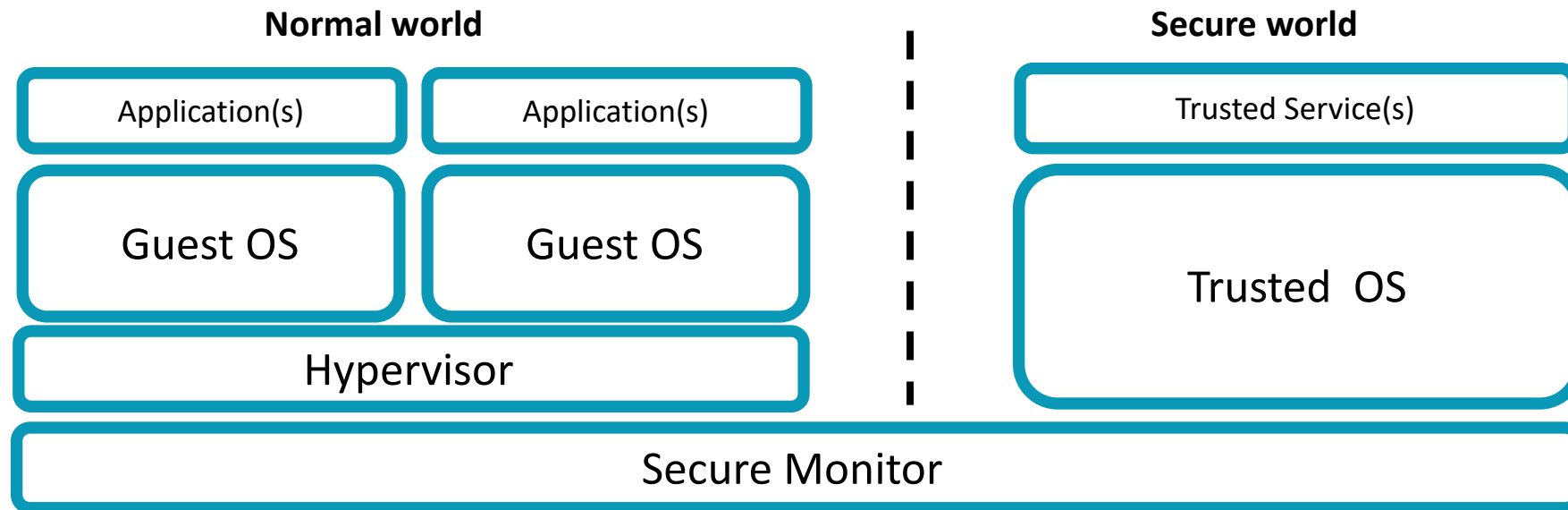
- Provides virtual address to physical address translation system
 - Up to 40 bits fine grain translation
- Arm Memory Management Unit (MMU) implements VMSA
 - Translation tables
 - Descriptor

Large Physical Address Extensions

- Long-descriptor format for page tables added
 - 32-bit virtual address mapped onto 40-bit physical address space
 - New translation table format using 64-bit translation table descriptors
 - 1TB of memory space accessible
- 32-bit short-descriptor format still supported
 - Configurable in the translation table base control register EAE bit (bit 31)
 - Can use 16MB memory supersections to map onto 40-bit address space

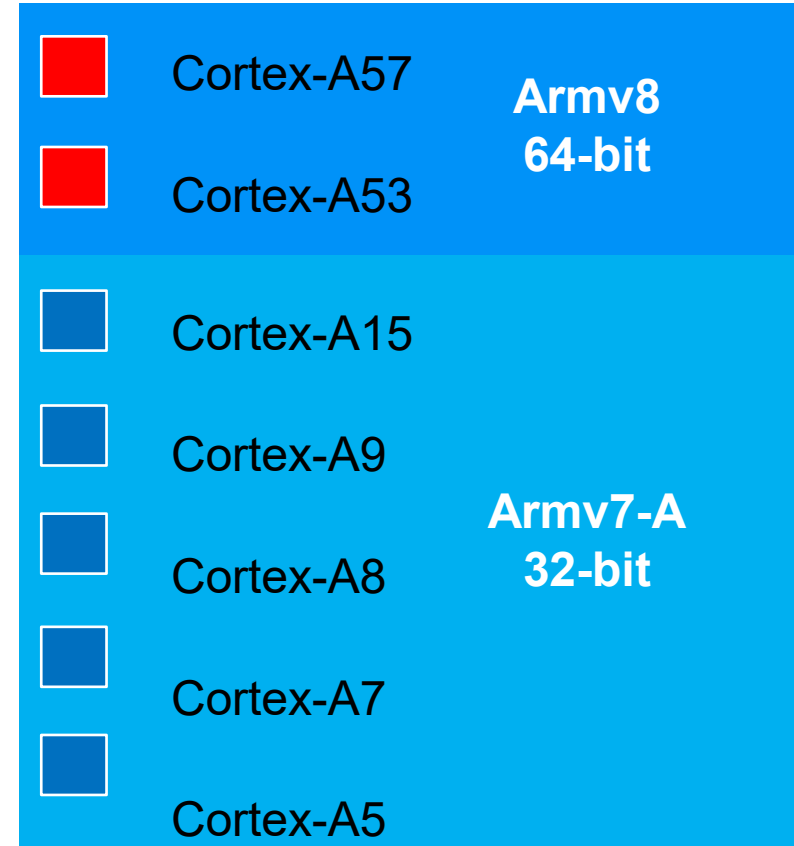
Virtualization

- Support for running multiple guest OSs in the normal world
- Hypervisor mode to control switching between guest OSs
- Two-stage address translation: OS and hypervisor levels
- Hypervisor mode can trap exceptions and choose which guest to direct them to.



Arm Cortex-A Series Processors

- Armv8 architecture: 64-bit
 - Cortex-A57
 - Cortex-A53
- Armv7 architecture: 32-bit
 - Cortex-A15
 - Cortex-A9
 - Cortex-A8
 - Cortex-A7
 - Cortex-A5



Arm Cortex-A Series Overview

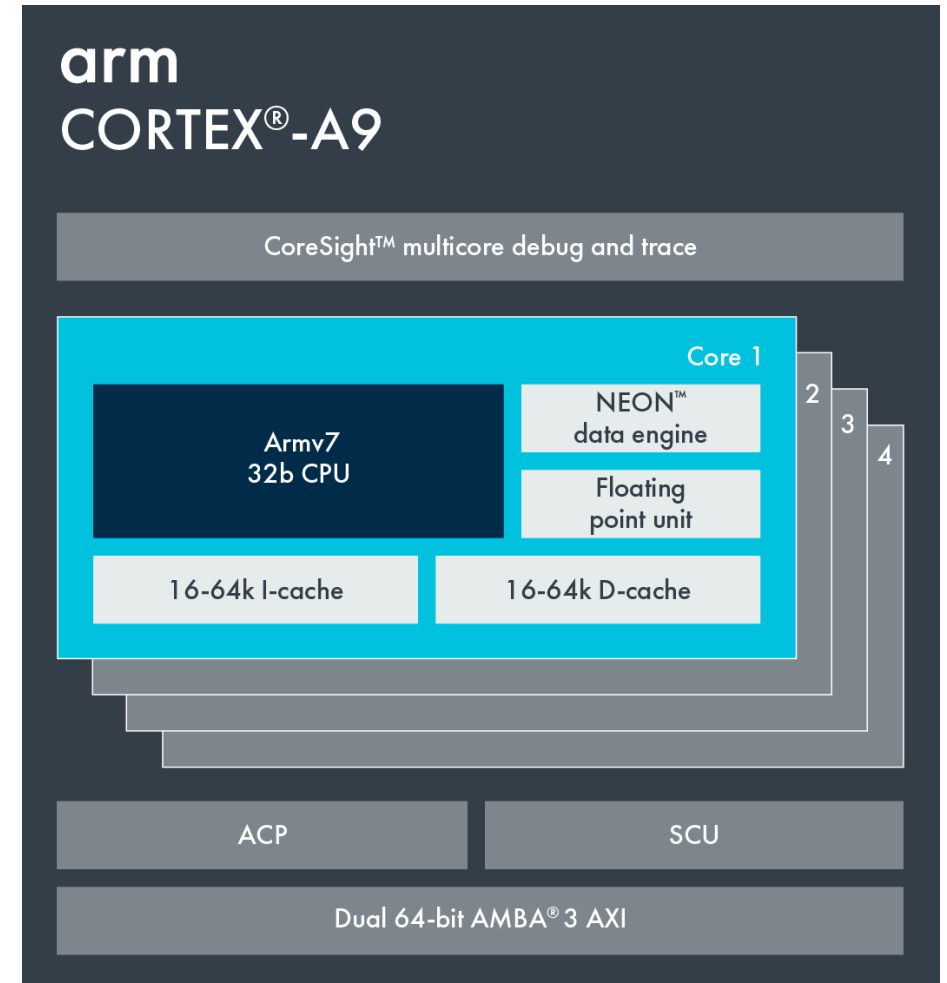
- High-performance
 - Used in applications that have high-compute requirements
 - Run rich OSs and deliver interactive media and graphics on the latest must-have devices.
- Multicore technology
 - Single to quad-core implementation for performance orientated applications
 - Supports symmetric and asymmetric OS implementations
 - Arm big.LITTLE compatible
- Advanced extensions
 - Thumb-2 for optimal code size and performance
 - TrustZone Security Extensions for trusted computing
 - Jazelle technology for accelerating execution environments such as Java, .Net, MSIL, Python, and Perl
- Ideal for mobile Internet
 - Native support for technologies like Adobe Flash
 - High-performance Neon engine for broad support of media codecs

Arm Cortex-A Series Processors

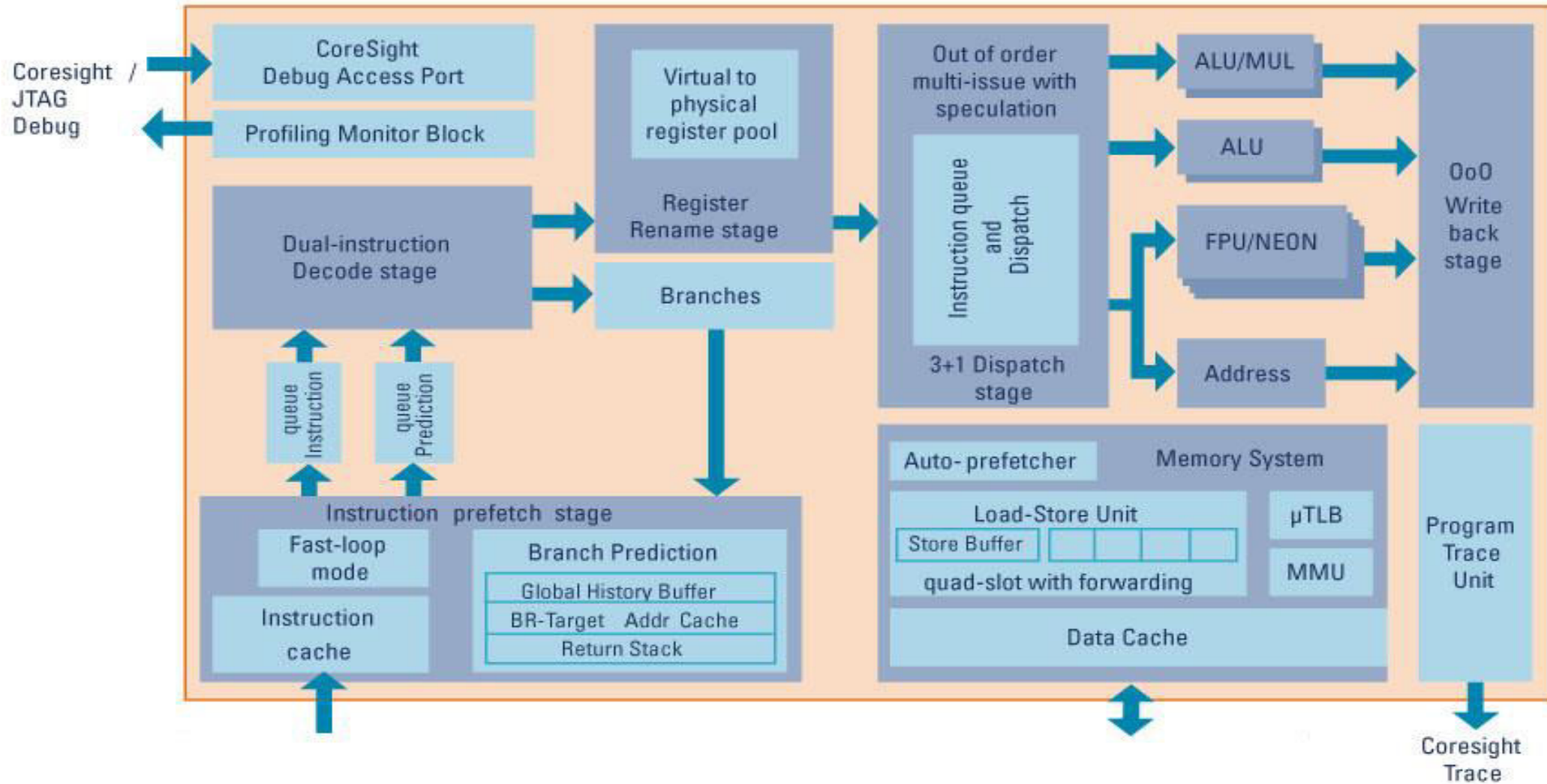
Processor	Performance	Typical Frequency	Architecture	Year	Comments
Cortex-A5	1.57 DMIPS/MHz /core	400-800 MHz	Armv7-A	2009	Cost-effective processor core
Cortex-A7	1.9 DMIPS/MHz /core	800 MHz-1.2 GHz	Armv7-A	2011	High-energy and area-efficient core
Cortex-A8	2.0 DMIPS/MHz/core	600 MHz-1 GHz	Armv7-A	2005	First one supporting Armv7-A architecture
Cortex-A9	2.5 DMIPS/MHz /core	800 MHz-2 GHz	Armv7-A	2007	Widely deployed Armv7-A-based processor
Cortex-A12	3.0 DMIPS/MHz /core		Armv7-A	2013	
Cortex-A15	>3.5 DMIPS/MHz /core	Up to 2.5GHz	Armv7-A	2010	High-performance core
Cortex-A17		Up to 2.5GHz	Armv7-A	2014	The most efficient Armv7-A-based processor
Cortex-A53	2.3 DMIPS/MHz		Armv8-A	2013	Most efficient 32/64-bit processor
Cortex-A57	>4.1 DMIPS/MHz		Armv8-A	2013	Proven high-performance 32/64-bit core for mobile and enterprise computing
Cortex-A72		Up to 2.5GHz	Armv8-A	2015	Arm's highest-performance processor

Arm Cortex-A9 Processor

- Arm Cortex-A9 features
 - Armv7 architecture: Thumb-2, ThumbEE
 - 0.8GHz to 2GHz
 - 2.5 DMIPS/MHz/core
 - Single core or 4x MPCore solution
 - Up to 20k DMIPS (2GHz, quad-core)
 - Power-efficient and high-performance processor
 - Dynamic length pipeline (8–11 stages)
 - Up to 64KB L1 I/D cache
 - Up to 8MB of L2 cache
 - Optional Neon media and/or floating point processing engine

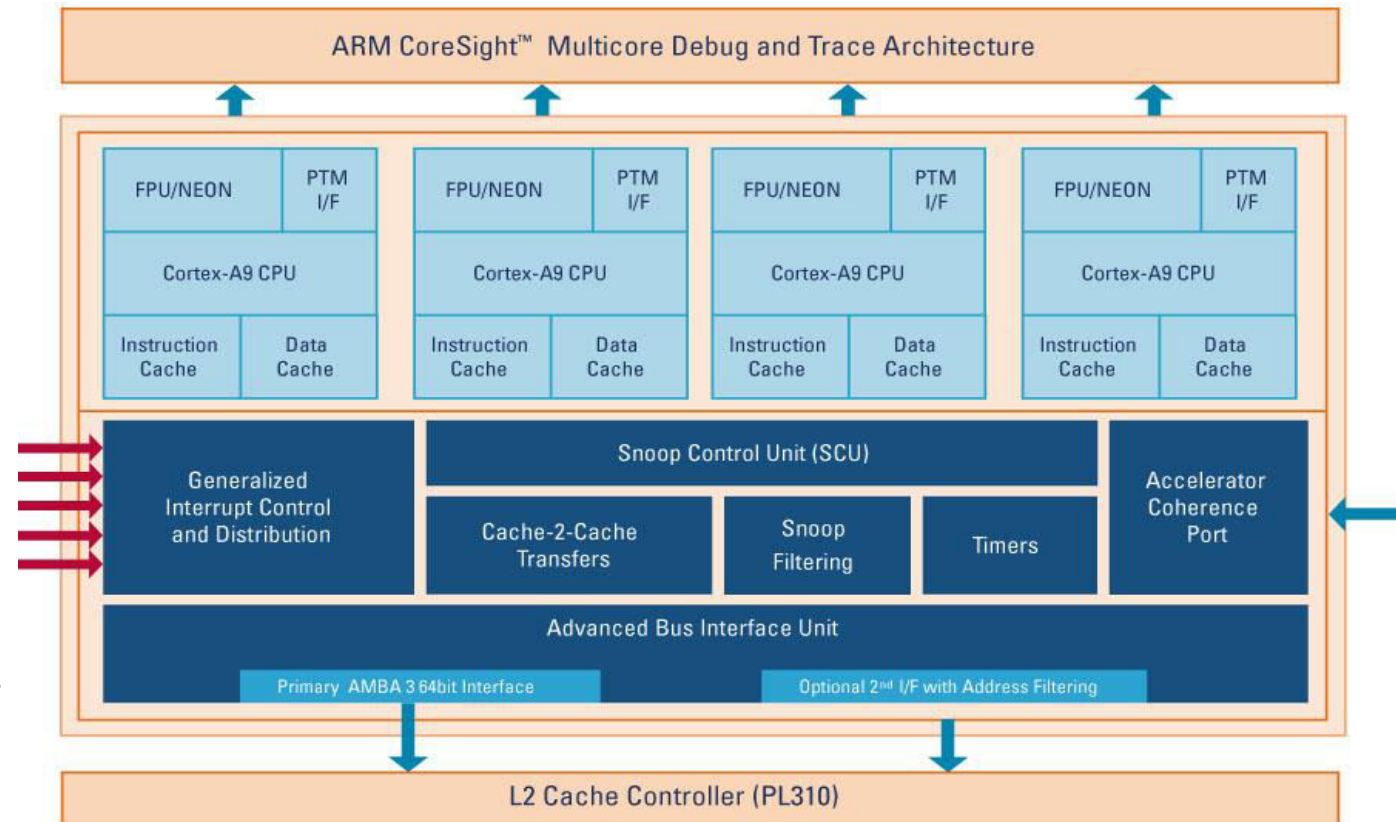


Cortex-A9 Diagram

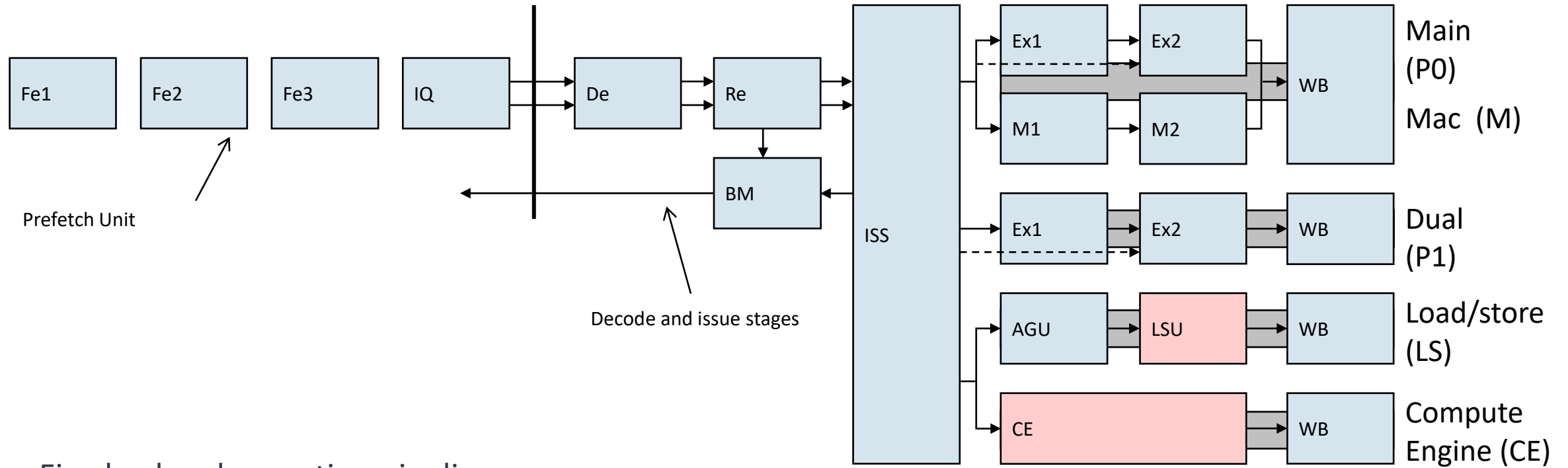


Cortex-A9 MPCore

- Contains up to four Cortex-A9 processors
- SCU
 - Maintains L1 data cache coherency between processors
 - Arbitrates accesses to the L2 memory system, through one or two external 64-bit AXI Manager interfaces
 - Optional ACP for maintaining coherency with DMA controller, graphics processor, or similar
- Integrated interrupt controller
 - Same programmer's model as Arm Generic Interrupt Controller (GIC): the PL390 PrimeCell



Cortex-A9 Pipeline

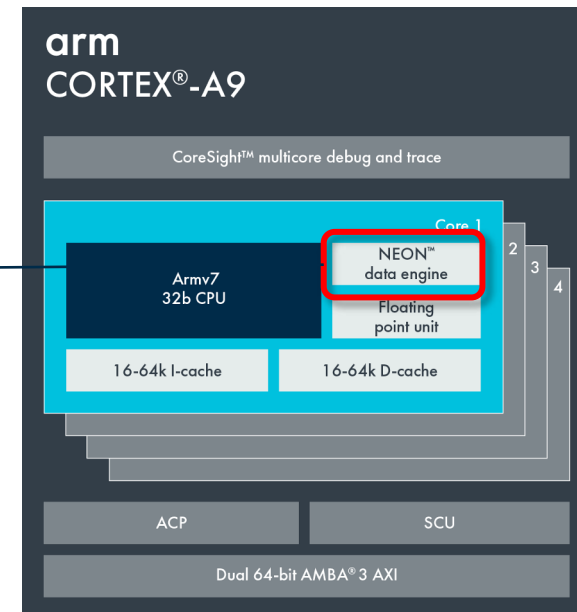
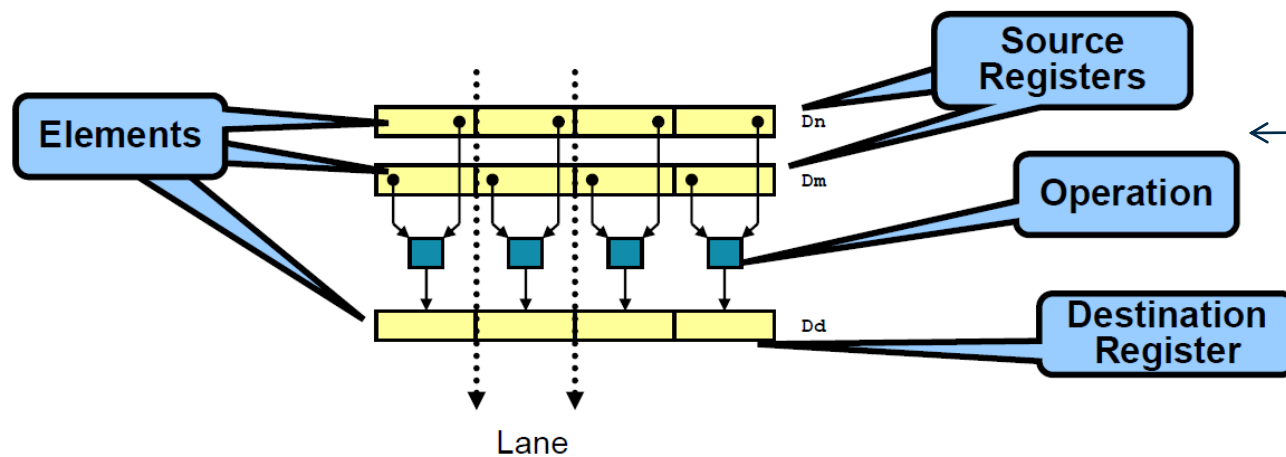


- Five backend execution pipelines
- Pipelines are clustered into three different issue groups.
 - Main, or multiply accumulate (Mac)
 - Dual execution (also known as secondary)
 - Load/store, or compute engine (Neon or floating point)

Core can issue up to 3 instructions per cycle.

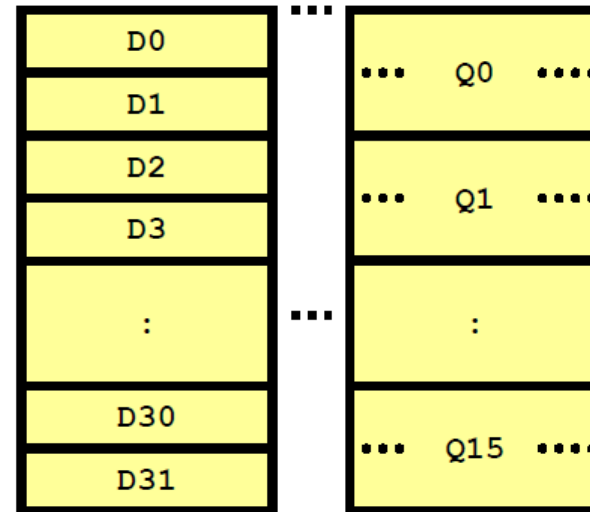
What Is Neon?

- Neon is a wide SIMD data processing architecture.
 - Extension of the Arm instruction set
 - Thirty-two registers, 64 bits wide (dual view as sixteen registers, 128 bits wide)
- Neon instructions perform packed SIMD processing.
 - Registers are considered vectors of elements of the same data type.
 - Data types can be signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single precision, and float.
 - Instructions perform the same operation in all lanes.

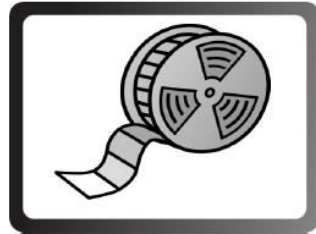


Neon Registers

- Neon provides a 256-byte register file.
 - Distinct from the core registers
 - Extension to the VFPv2 register file (VFPv3)
- Two explicitly aliased views
 - 32 × 64-bit registers (D0–D31)
 - 16 × 128-bit registers (Q0–Q15)
- Enables register trade-off
 - Vector length
 - Available registers



Neon: Enhancing User Experiences



Watch any video in **any** format



Edit & Enhance captured videos
Video stabilization



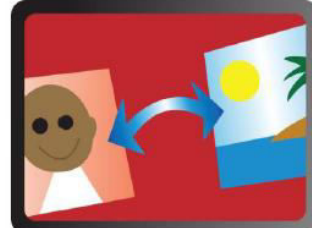
Antialiased rendering & compositing



Advanced User Interfaces



Game processing



Process megapixel photos quickly



Voice recognition



Powerful multi-channel hi-fi audio processing