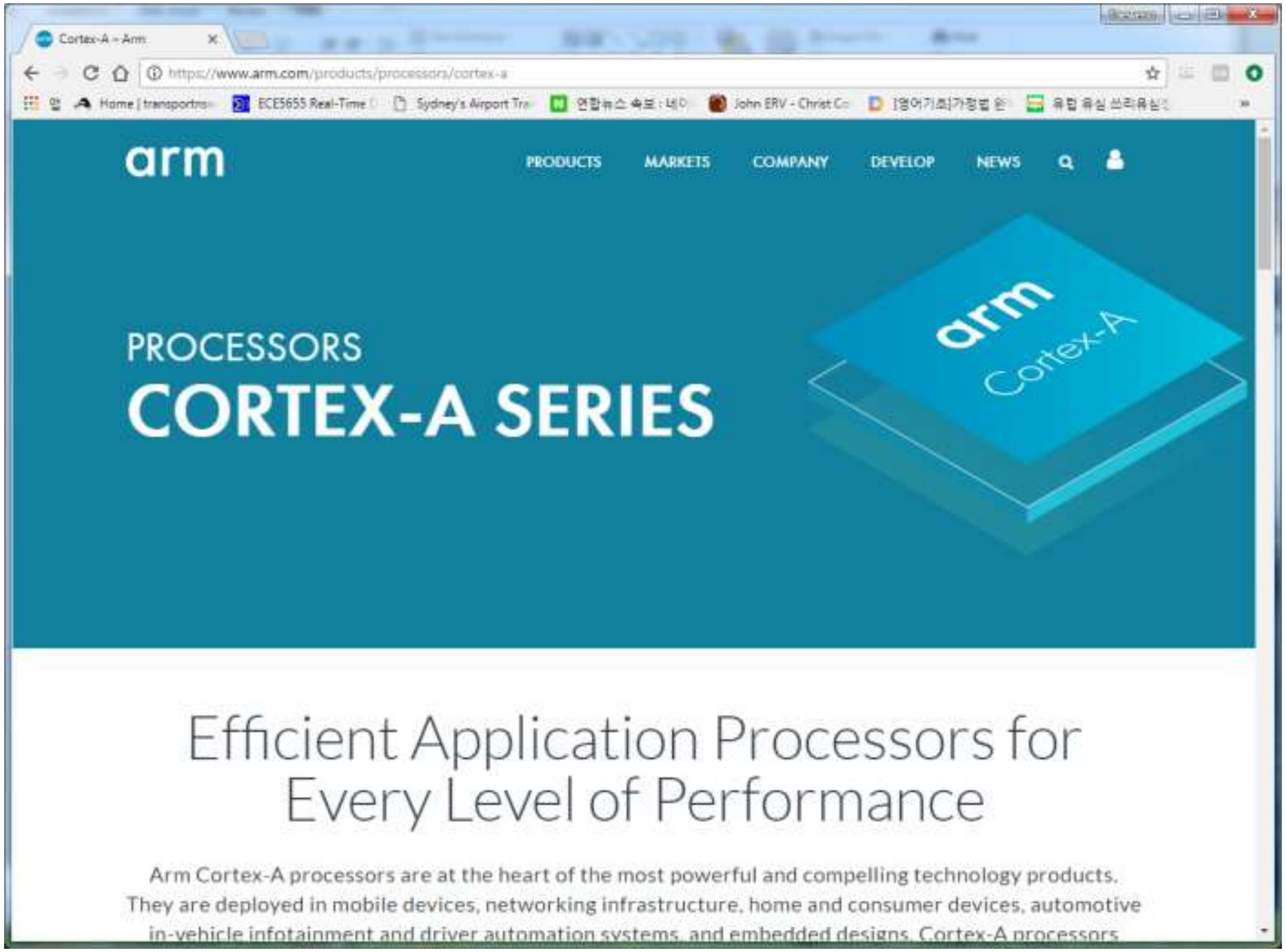
고려대학교
KOREA UNIVERSITY

COSE321 Computer Systems Design

Lecture 1. ARM Intro.

Prof. Taeweon Suh
Computer Science & Engineering
Korea University

www.arm.com (as of 2018)



Cortex-A – Arm

https://www.arm.com/products/processors/cortex-a

PROCESSORS

CORTEX-A SERIES

Efficient Application Processors for Every Level of Performance

Arm Cortex-A processors are at the heart of the most powerful and compelling technology products. They are deployed in mobile devices, networking infrastructure, home and consumer devices, automotive in-vehicle infotainment and driver automation systems, and embedded designs. Cortex-A processors

ARM Brief

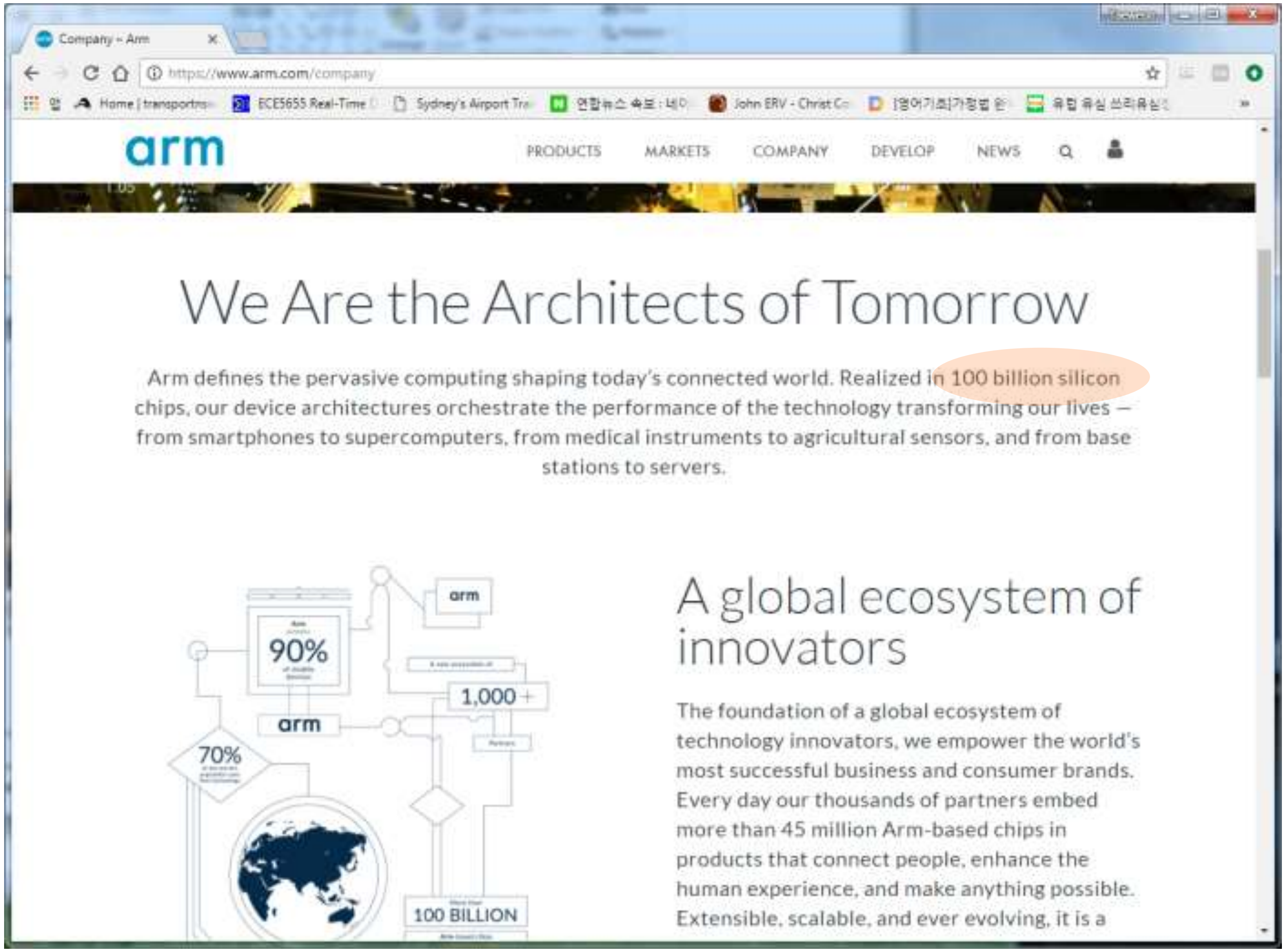
- ARM architecture was first developed in the **1980s** by Acorn
- Spin off from Acorn in **1990**
- As of 2013, Arm architecture is the most widely used **32-bit ISA** in terms of quantity produced
- **64-bit architecture** (Armv8) was announced in Oct. **2011**.
- Armv9 (64-bit architecture) was announced in March **2021**
  - Machine learning, Enhanced security, and Improved vector and DSP capabilities

(DSP: Digital Signal Processing)

- In 2010 alone, 6.1 billion Arm-based processors shipped, representing
  - 95% of smartphones, 35% of digital TV and set-top boxes, 10% of mobile computers
- Softbank, a Japanese company, surprised the technology world with the arm acquisition in **2016** at $31.4 billion

Source: Wikipedia
https://www.anandtech.com/show/16584/arm-announces-armv9-architecture

Vision (as of 2018)



We Are the Architects of Tomorrow

Arm defines the pervasive computing shaping today's connected world. Realized in 100 billion silicon chips, our device architectures orchestrate the performance of the technology transforming our lives – from smartphones to supercomputers, from medical instruments to agricultural sensors, and from base stations to servers.

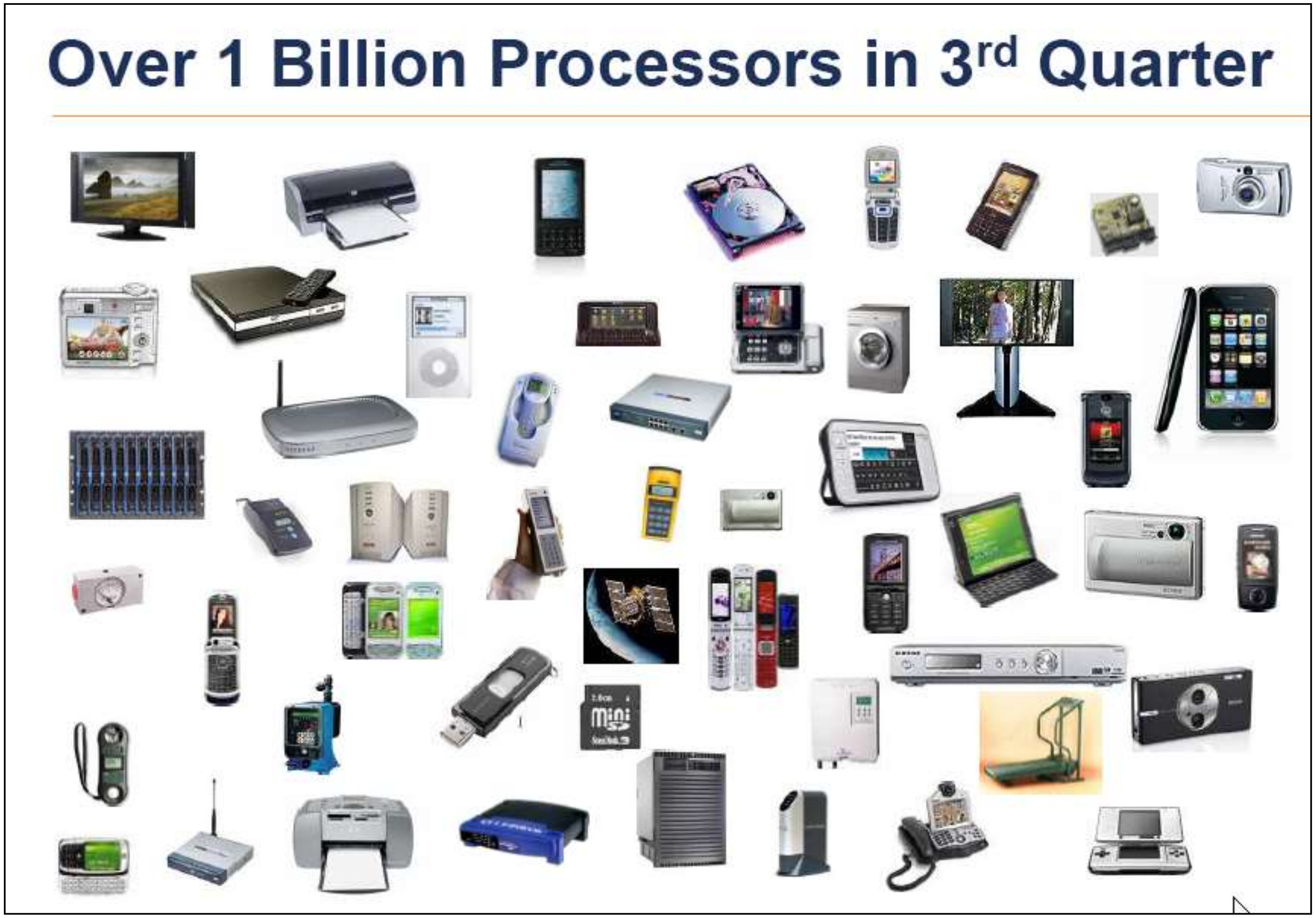
A global ecosystem of innovators

The foundation of a global ecosystem of technology innovators, we empower the world's most successful business and consumer brands. Every day our thousands of partners embed more than 45 million Arm-based chips in products that connect people, enhance the human experience, and make anything possible. Extensible, scalable, and ever evolving, it is a

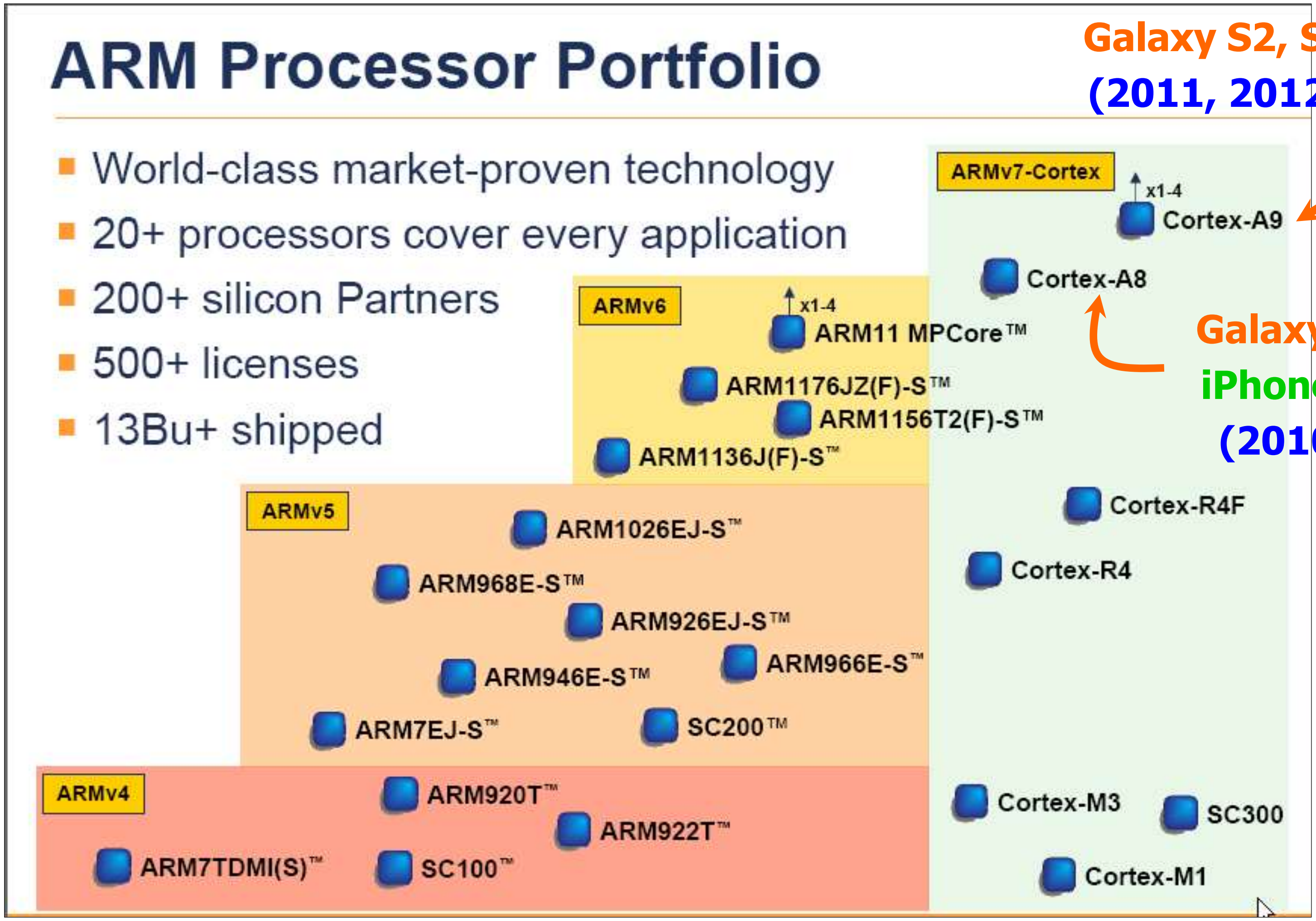
ARM Partners

Source: 2008 Embedded SW Insight Conference

ARM (as of 2008)



ARM Processor Portfolio



ARM Processor Portfolio

- World-class market-proven technology
- 20+ processors cover every application
- 200+ silicon Partners
- 500+ licenses
- 13Bu+ shipped

Galaxy S2, S3
(2011, 2012)

Galaxy S
iPhone 4
(2010)

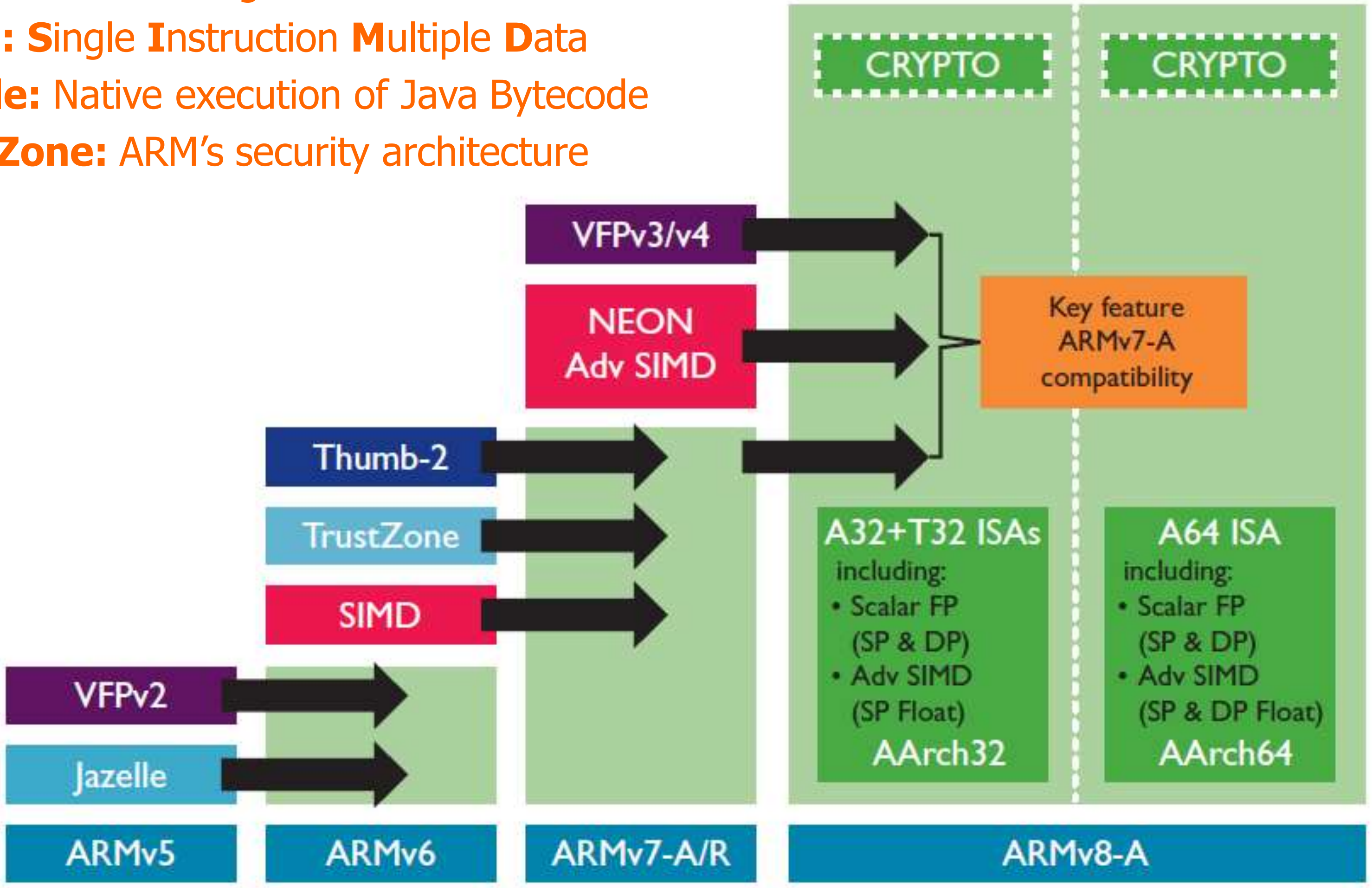
~1990s ~2000s

Product Code

- T: Thumb
- T2: Thumb-2 Enhancement
- D: Debug
- M: Multiplier
- I: Embedded ICE (In-Circuit Emulation)
- E: Enhanced DSP Extension
- J: Jazelle
  - Direct execution of 8-bit Java bytecode in hardware
- S: Synthesizable core
- Z: Should be TrustZone?

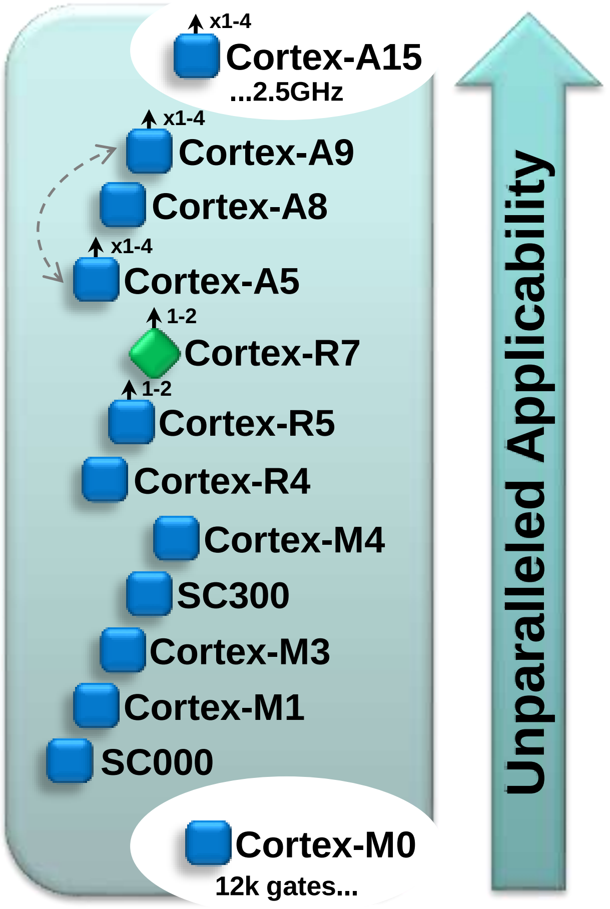
ARMv5, v6, v7, v8

- **VFP: V**ector **F**loating **P**oint
- **SIMD: S**ingle **I**nstruction **M**ultiple **D**ata
- **Jazelle:** Native execution of Java Bytecode
- **TrustZone:** ARM's security architecture



ARM Cortex Series

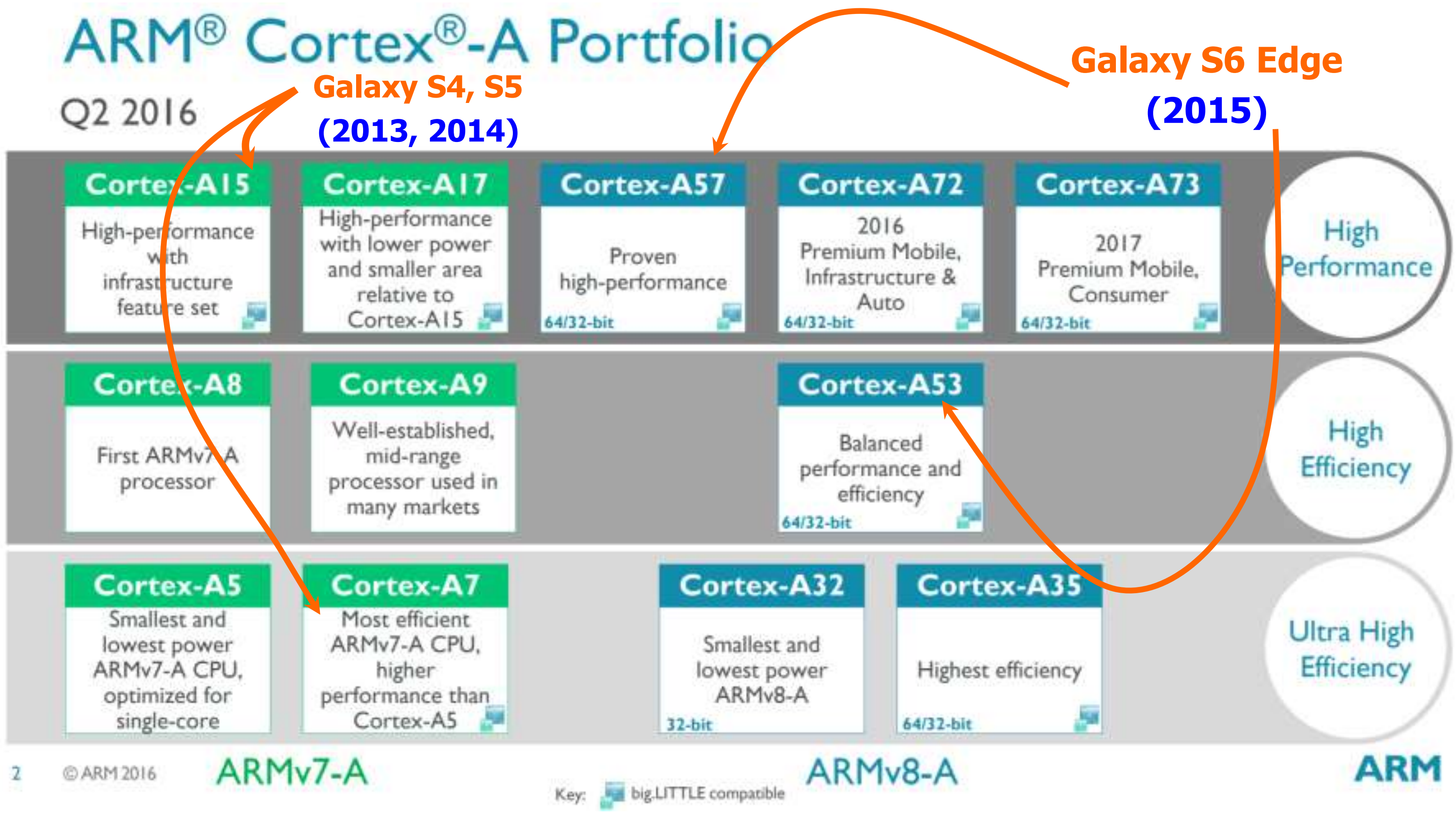
- ARM Cortex-**A** family:
  - Applications processors for feature-rich OS and 3rd party applications
- ARM Cortex-**R** family:
  - Embedded processors for Real-time signal processing, control applications
- ARM Cortex-**M** family:
  - Microcontroller-oriented processors for MCU, ASSP, and SoC applications



MCU: Microcontroller Unit
ASSP: Application Specific Standard Product
SoC: System-on-chip

Source: ARM Processor Portfolio 2011

Cortex-A Portfolio



ARM® Cortex®-A Portfolio

Q2 2016

Galaxy S4, S5 (2013, 2014)

Galaxy S6 Edge (2015)

Cortex-A15 High-performance with infrastructure feature set

Cortex-A17 High-performance with lower power and smaller area relative to Cortex-A15

Cortex-A57 Proven high-performance 64/32-bit

Cortex-A72 2016 Premium Mobile, Infrastructure & Auto 64/32-bit

Cortex-A73 2017 Premium Mobile, Consumer 64/32-bit

High Performance

Cortex-A8 First ARMv7-A processor

Cortex-A9 Well-established, mid-range processor used in many markets

Cortex-A53 Balanced performance and efficiency 64/32-bit

High Efficiency

Cortex-A5 Smallest and lowest power ARMv7-A CPU, optimized for single-core

Cortex-A7 Most efficient ARMv7-A CPU, higher performance than Cortex-A5

Cortex-A32 Smallest and lowest power ARMv8-A 32-bit

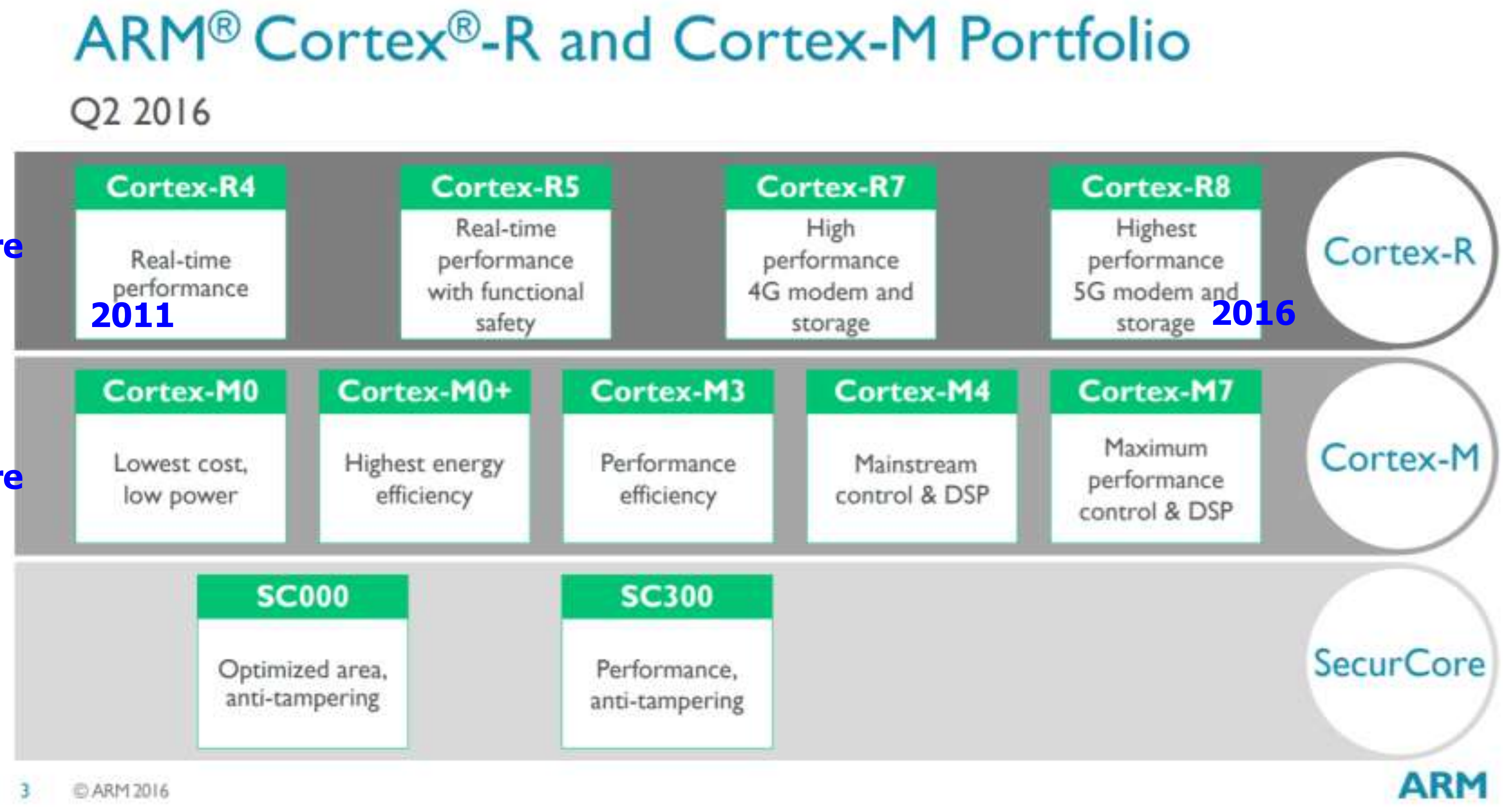
Cortex-A35 Highest efficiency 64/32-bit

Ultra High Efficiency

2 © ARM 2016 ARMv7-A Key: big.LITTLE compatible ARMv8-A ARM

32-bit architecture **64-bit architecture**

Cortex-R & Cortex-M Portfolio



ARM® Cortex®-R and Cortex-M Portfolio

Q2 2016

Cortex-R	Cortex-R4: Real-time performance	Cortex-R5: Real-time performance with functional safety	Cortex-R7: High performance 4G modem and storage	Cortex-R8: Highest performance 5G modem and storage	
Cortex-M	Cortex-M0: Lowest cost, low power	Cortex-M0+: Highest energy efficiency	Cortex-M3: Performance efficiency	Cortex-M4: Mainstream control & DSP	Cortex-M7: Maximum performance control & DSP
SecurCore	SC000: Optimized area, anti-tampering	SC300: Performance, anti-tampering			

3 © ARM 2016

ARM

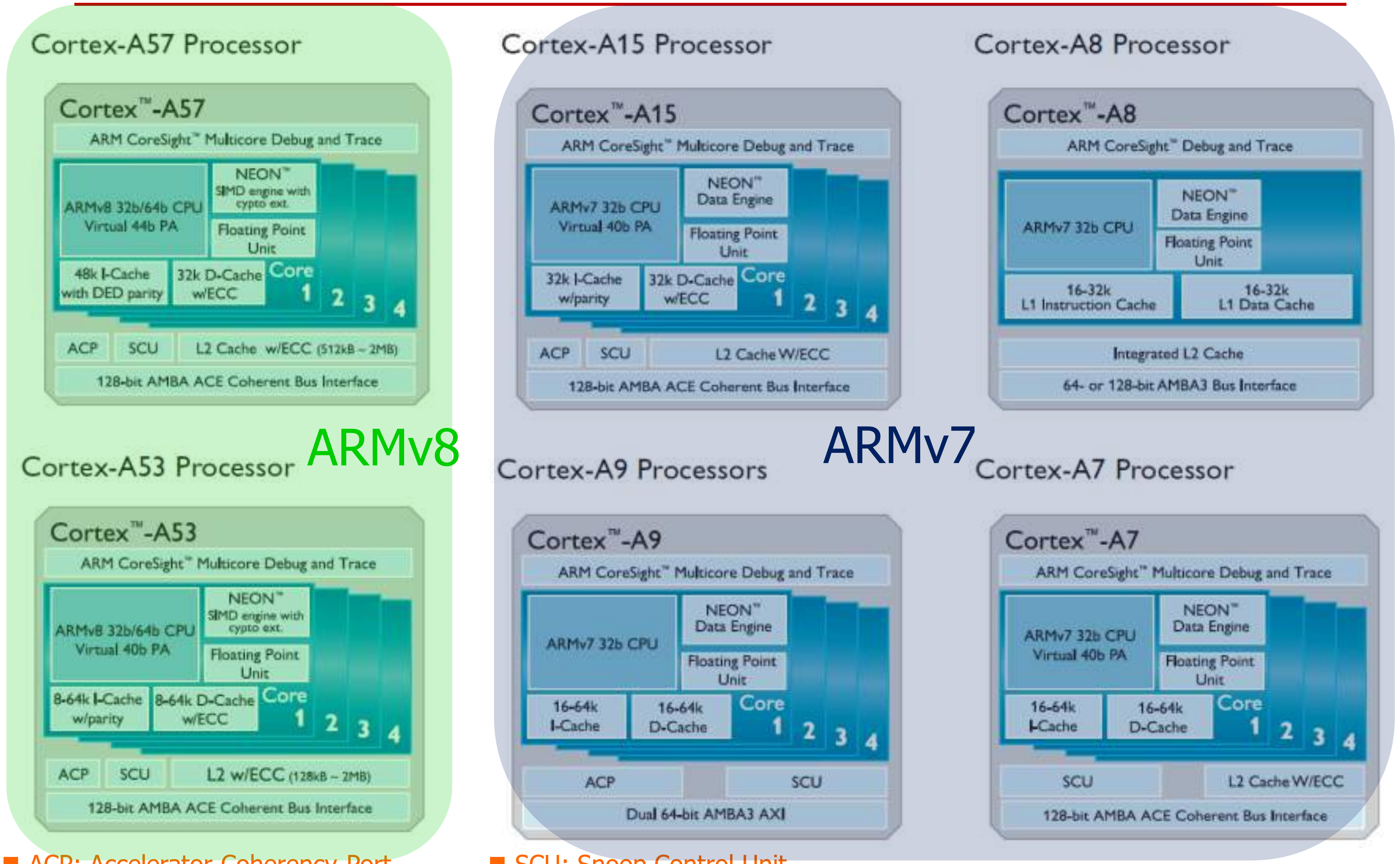
32-bit architecture

2011

2016

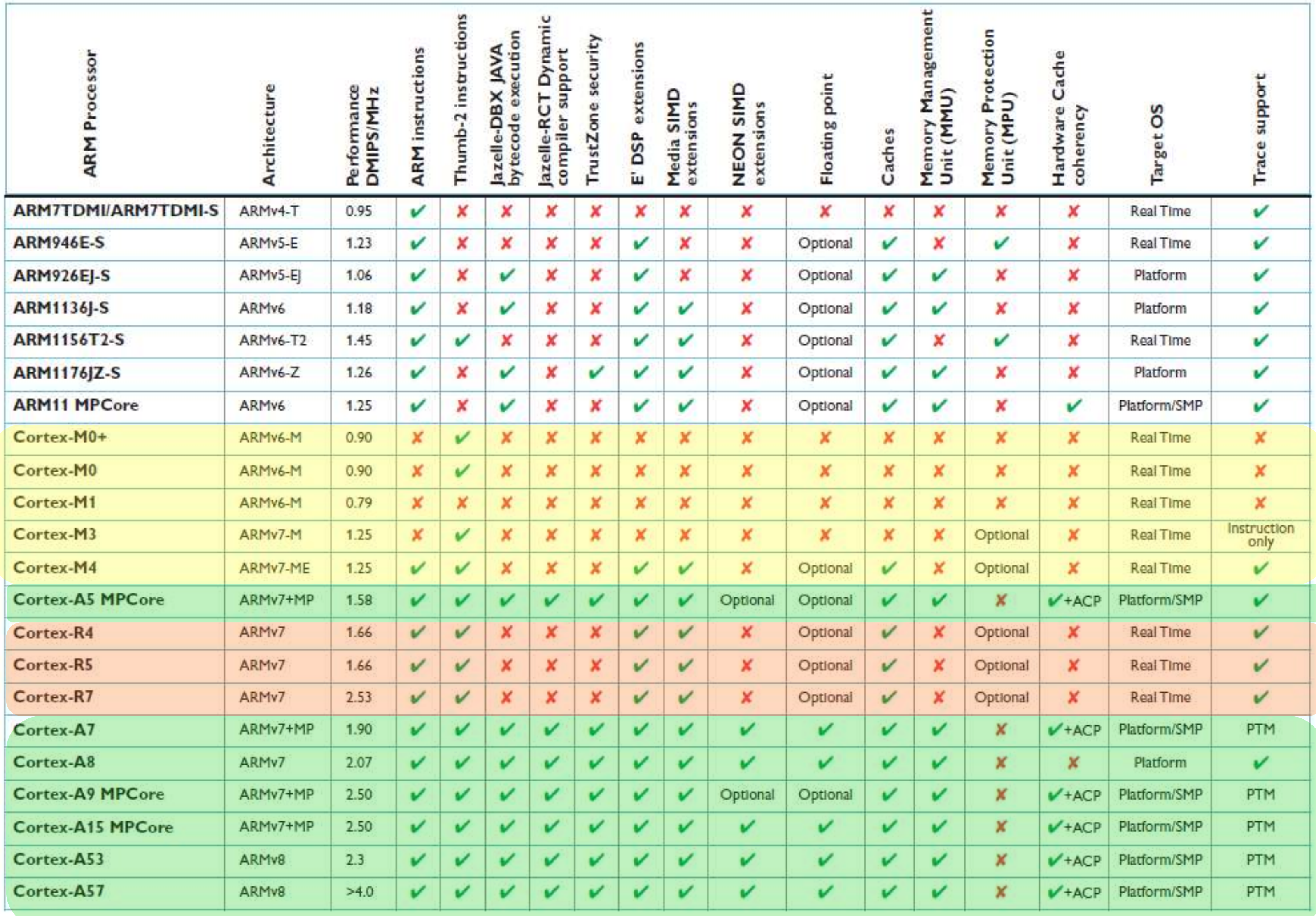
32-bit architecture

ARMv8-A & ARMv7-A



- ACP: Accelerator Coherency Port
- SCU: Snoop Control Unit

ARM Processor Selector



ARM Processor	Architecture	Performance DMIPS/MHz	ARM instructions	Thumb-2 instructions	Jazelle-DBX JAVA bytecode 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
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	✓
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	✓
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
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 Processor Brief

	#pipeline stages	Frequency	Architecture	Process
ARM6 (1992)	3	~33MHz	ARMv3	1.2μm
ARM7TDMI	3	~70MHz	ARMv4	0.13μm
ARM920T	5	~400MHz	ARMv4	90nm
ARM1136J	8	~1Ghz	ARMv6	65nm
Cortex-A9	8~11 (OoO)	~2GHz	ARMv7	32nm
Cortex-A15	15~24 (OoO)	~2.5GHz	ARMv7	22nm

- OOO: Out Of Order

Abstraction

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - An abstract interface between the hardware and the low-level software interface

Abstraction in Computer

Users

Application programming using APIs

Abstraction layer

Operating Systems

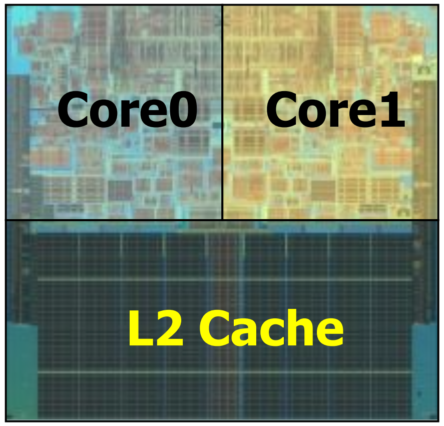
Abstraction layer

Instruction Set Architecture (ISA)

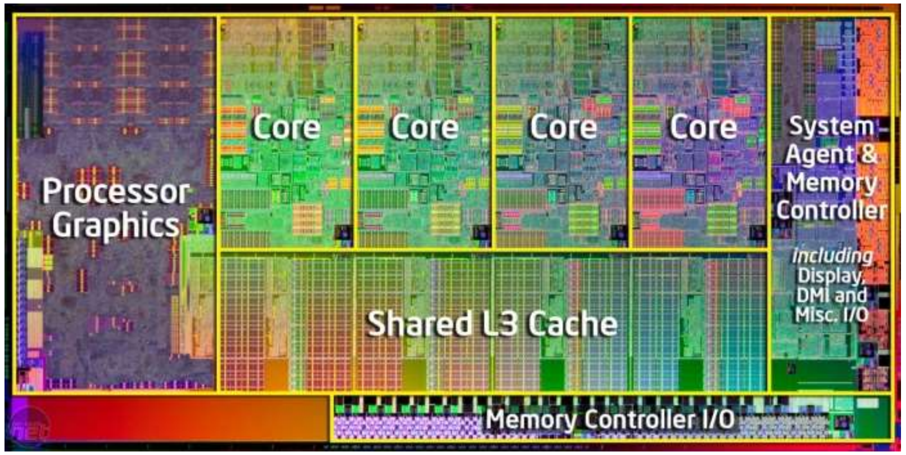
Machine language

Assembly language

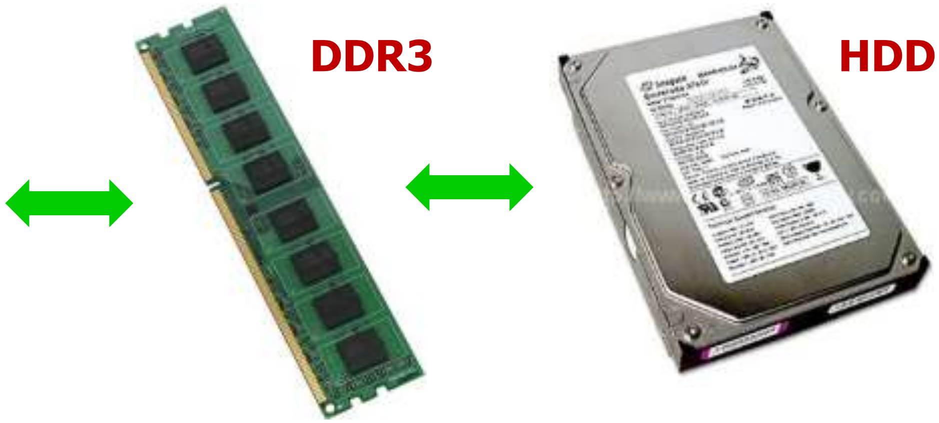
Hardware implementation



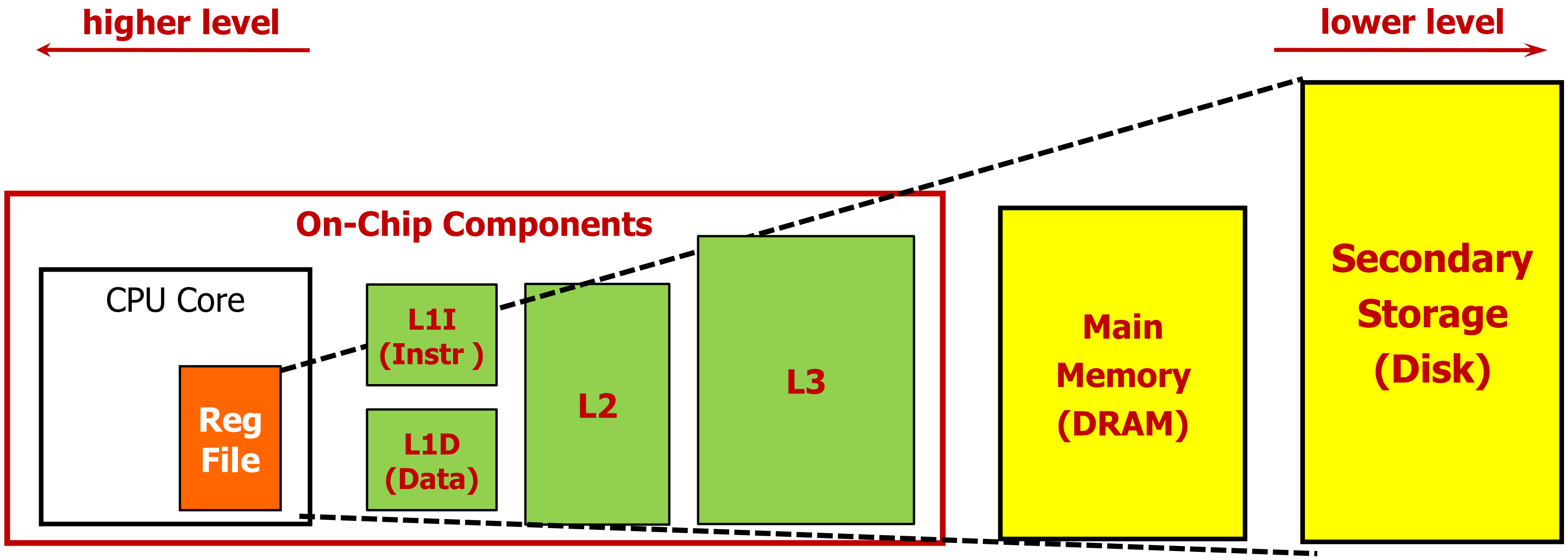
A Memory Hierarchy



2nd Gen. Core i7
(2011)



A Memory Hierarchy



Speed (cycles):	½'s	1's	10's	100's	10,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost:	highest				lowest

Typical and Essential Instructions

- CPU provides many instructions
  - It would be time-consuming to study all the instructions CPU provides
  - There are essential and common instructions

- Instruction categories
  - Data processing instructions
    - Arithmetic and Logical (Integer)
  - Memory access instructions
    - Load and Store
  - Branch instructions

High-level Code to Binary (x86)

```
%gcc -g simple_sub.c -o simple_sub
%objdump -SD -Mintel simple_sub
```

```
#include <stdio.h>

#define  A 3
#define  B 5

int main() {
printf("%d - %d = %d",
          A, B, mysub(A, B));
return 0;
}

int mysub(int op1, int op2) {
int myres;
myres = op1 - op2;
return myres;
}
```

simple_sub.c

C Compiler

address

Instructions (human-readable)

```
int mysub(int op1, int op2) {

 8048429:   55             push   ebp
 804842a:   89 e5          mov    ebp,esp
 804842c:   83 ec 10       sub    esp,0x10
  int myres;

  myres = op1 - op2;

 804842f:   8b 45 0c       mov    eax,DWORD PTR [ebp+0xc]
 8048432:   8b 55 08       mov    edx,DWORD PTR [ebp+0x8]
 8048435:   89 d1          mov    ecx,edx
 8048437:   29 c1          sub    ecx,eax
 8048439:   89 c8          mov    eax,ecx
 804843b:   89 45 fc       mov    DWORD PTR [ebp-0x4],eax

  return myres;
 804843e:   8b 45 fc       mov    eax,DWORD PTR [ebp-0x4]

}
 8048441:   c9             leave
 8048442:   c3             ret
```

Representation in hexadecimal (machine-readable)

Levels of Program Code (ARM)

- High-level language program (in C)

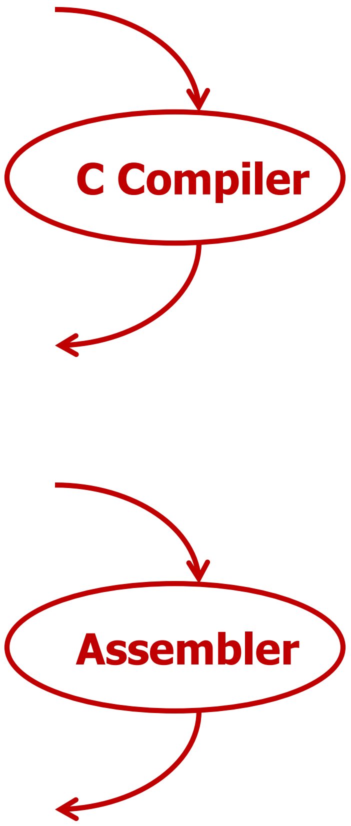
```
swap (int v[], int k)
{       int temp;
         temp = v[k];
         v[k] = v[k+1];
         v[k+1] = temp;
}
```

- **Assembly language** program

```
swap:   sll     R2, R5, #2
        add     R2, R4, R2
        ldr     R12, 0(R2)
        ldr     R10, 4(R2)
        str     R10, 0(R2)
        str     R12, 4(R2)
        b       exit
```

- **Machine (object, binary)** code

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
  . . .
```



CISC vs RISC

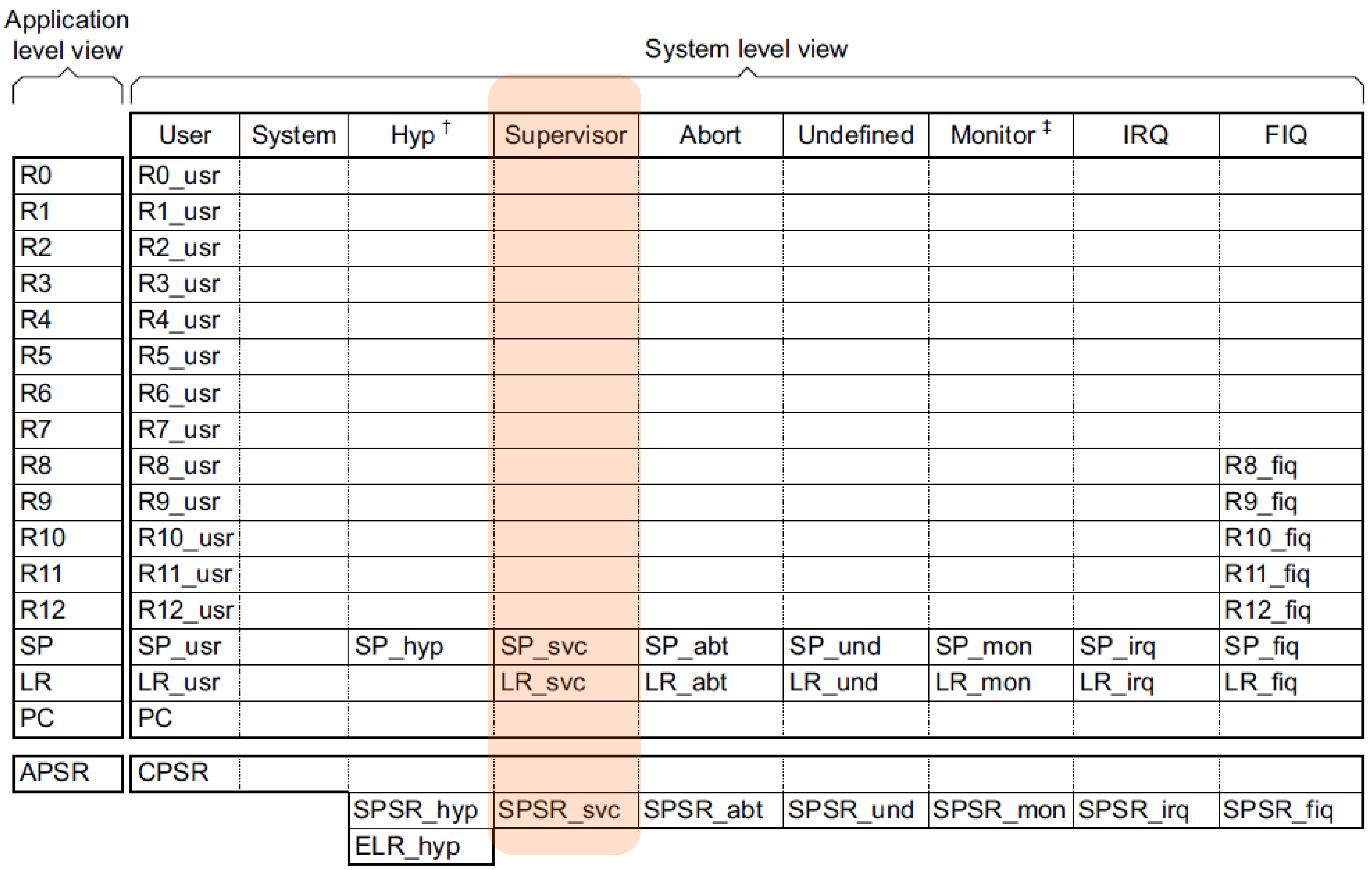
- CISC (Complex Instruction Set Computer)
  - One instruction does **many (complex)** jobs
    - Example: `movs` in x86
  - Variable length instruction
  - Examples: x86 (Intel, AMD), Motorola 68k

- RISC (Reduced Instruction Set Computer)
  - Each instruction does a **small (unit)** job
    - Example: `add, lw, sw, beq` in RISC-V
  - Fixed-length instruction
  - Load/Store Architecture
  - Examples: RISC-V, Arm, MIPS

ARM Architecture

- ARM is RISC (Reduced Instruction Set Computer)
  - x86 ISA is based on CISC (Complex Instruction Set Computer) even though x86 internally implements RISC-like microcode and pipelining
- Suitable for embedded systems (Cortex-M)
  - Very small die size (low price)
  - Low power consumption (longer battery life)

ARM Registers



Application level view	System level view								
	User	System	Hyp †	Supervisor	Abort	Undefined	Monitor ‡	IRQ	FIQ
R0	R0_usr								
R1	R1_usr								
R2	R2_usr								
R3	R3_usr								
R4	R4_usr								
R5	R5_usr								
R6	R6_usr								
R7	R7_usr								
R8	R8_usr								R8_fiq
R9	R9_usr								R9_fiq
R10	R10_usr								R10_fiq
R11	R11_usr								R11_fiq
R12	R12_usr								R12_fiq
SP	SP_usr		SP_hyp	SP_svc	SP_abt	SP_und	SP_mon	SP_irq	SP_fiq
LR	LR_usr			LR_svc	LR_abt	LR_und	LR_mon	LR_irq	LR_fiq
PC	PC								
APSR	CPSR								
			SPSR_hyp	SPSR_svc	SPSR_abt	SPSR_und	SPSR_mon	SPSR_irq	SPSR_fiq
			ELR_hyp						

‡ Part of the Security Extensions. Exists only in Secure state.

† Part of the Virtualization Extensions. Exists only in Non-secure state.

Cells with no entry indicate that the User mode register is used.

At reset, ARM goes to the "Supervisor" mode

(see a backup slide, "exception vector", p35)

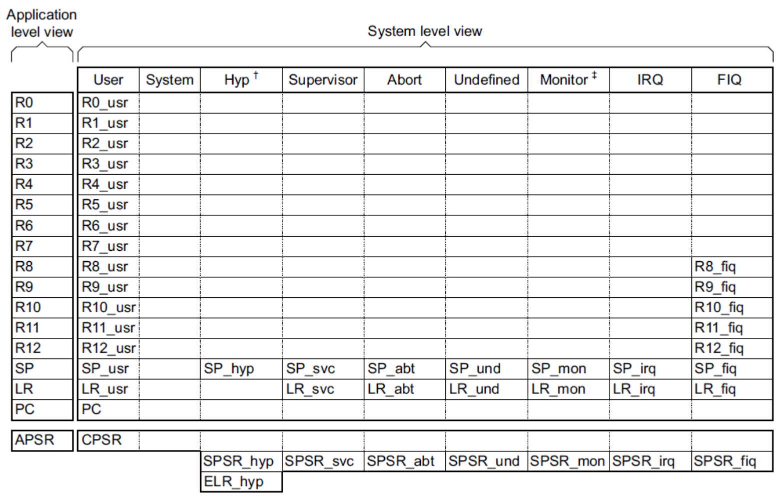
Figure B1-2 ARM core registers, PSRs, and ELR_hyp, showing register banking

Source: ARMv7 Architecture Reference Manual

ARM Registers

- Unbanked registers: R0 ~ R7
  - Each of them refers to the same 32-bit physical register in all processor modes.
  - They are completely general-purpose registers, with no special uses implied by the architecture

- Banked registers: R8 ~ R14
  - R8 ~ R12 have no dedicated special purposes
    - FIQ mode has dedicated registers for fast interrupt processing
  - R13 and R14 are dedicated for special purposes for each mode



Application level view	System level view								
	User	System	Hyp †	Supervisor	Abort	Undefined	Monitor ‡	IRQ	FIQ
R0	R0_usr								
R1	R1_usr								
R2	R2_usr								
R3	R3_usr								
R4	R4_usr								
R5	R5_usr								
R6	R6_usr								
R7	R7_usr								
R8	R8_usr								R8_fiq
R9	R9_usr								R9_fiq
R10	R10_usr								R10_fiq
R11	R11_usr								R11_fiq
R12	R12_usr								R12_fiq
SP	SP_usr		SP_hyp	SP_svc	SP_abt	SP_und	SP_mon	SP_irq	SP_fiq
LR	LR_usr			LR_svc	LR_abt	LR_und	LR_mon	LR_irq	LR_fiq
PC	PC								
APSR	CPSR								
			SPSR_hyp	SPSR_svc	SPSR_abt	SPSR_und	SPSR_mon	SPSR_irq	SPSR_fiq
			ELR_hyp						

‡ Part of the Security Extensions. Exists only in Secure state.
† Part of the Virtualization Extensions. Exists only in Non-secure state.
Cells with no entry indicate that the User mode register is used.

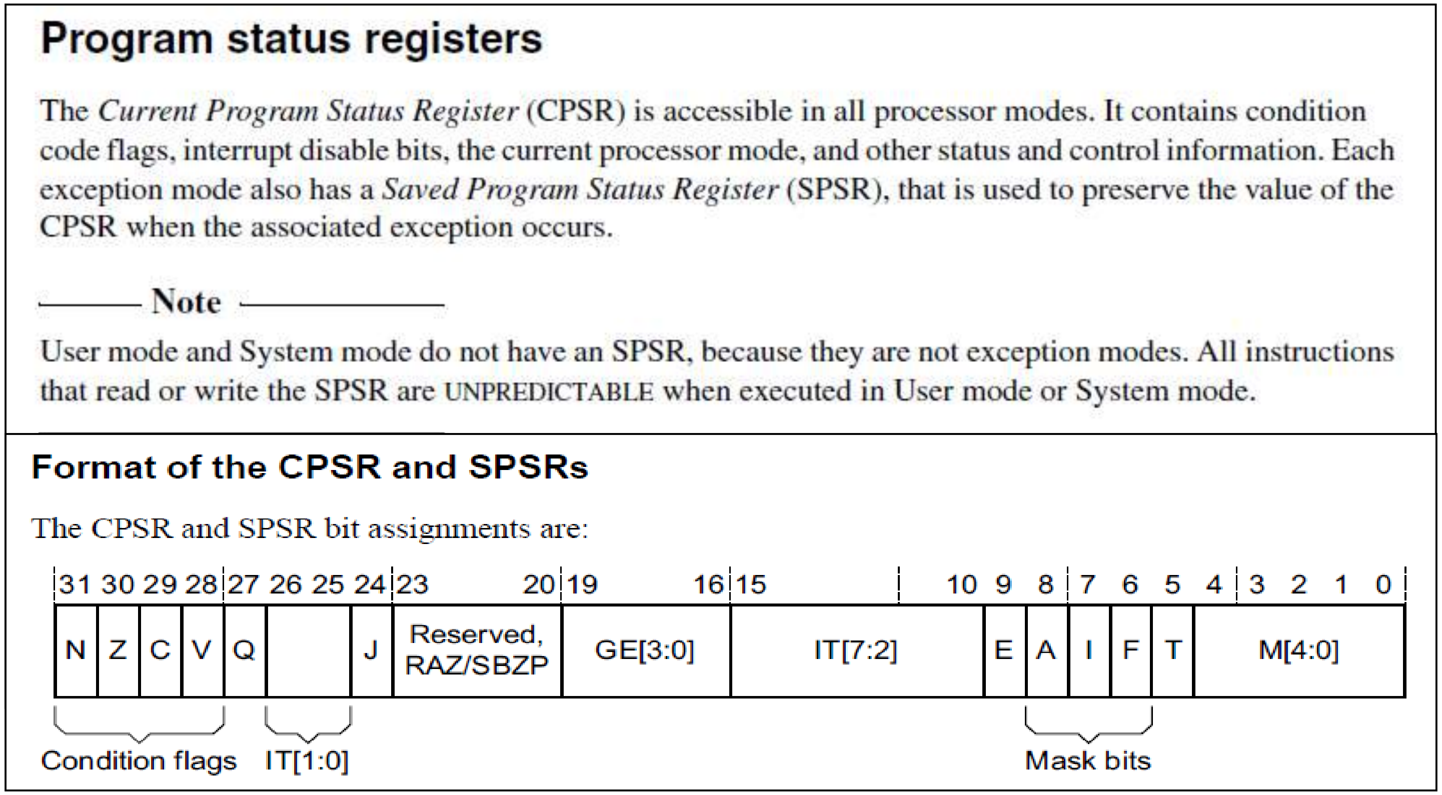
Figure B1-2 ARM core registers, PSRs, and ELR_hyp, showing register banking

R13, R14, and R15

- Some registers in ARM are used for special purposes
  - R15 == PC (Program Counter)
    - x86 uses a terminology called IP (Instruction Pointer)
  - R14 == LR (Link Register)
  - R13 == SP (Stack Pointer)

CPSR

- Current Program Status Register (CPSR) is accessible in all modes
- Contains all condition flags, interrupt disable bits, the current processor mode



Program status registers

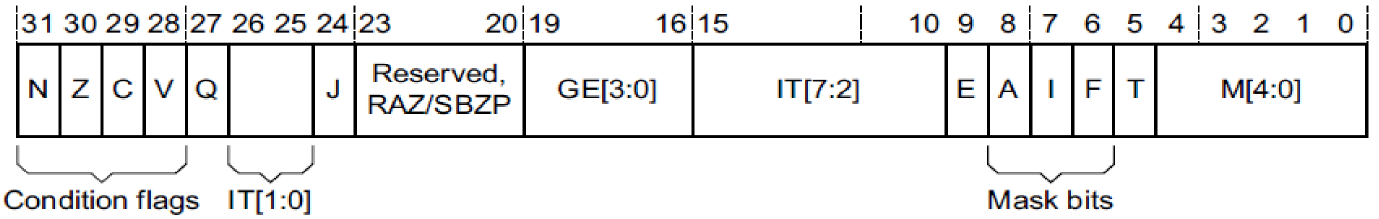
The *Current Program Status Register* (CPSR) is accessible in all processor modes. It contains condition code flags, interrupt disable bits, the current processor mode, and other status and control information. Each exception mode also has a *Saved Program Status Register* (SPSR), that is used to preserve the value of the CPSR when the associated exception occurs.

Note

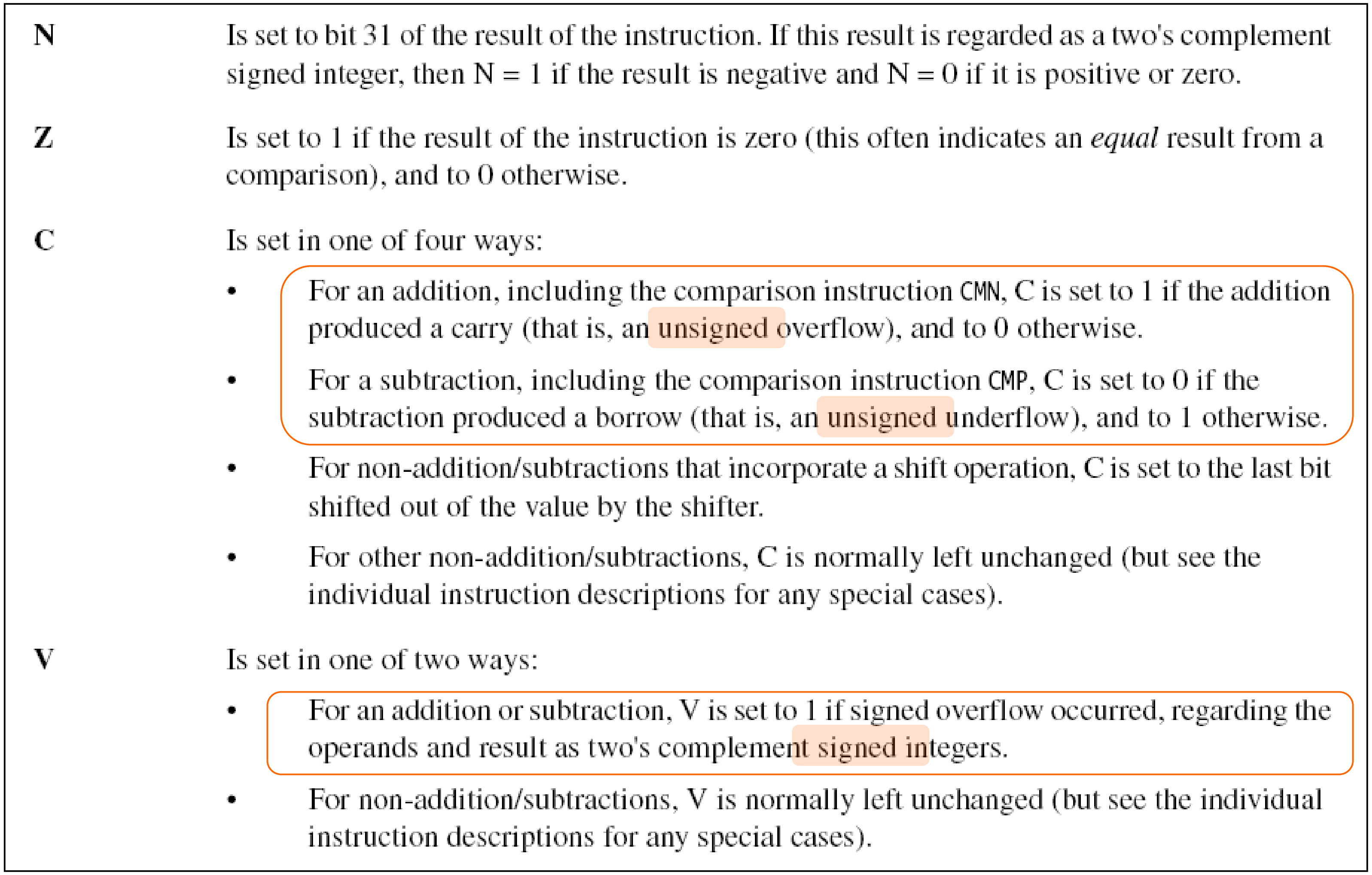
User mode and System mode do not have an SPSR, because they are not exception modes. All instructions that read or write the SPSR are UNPREDICTABLE when executed in User mode or System mode.

Format of the CPSR and SPSRs

The CPSR and SPSR bit assignments are:

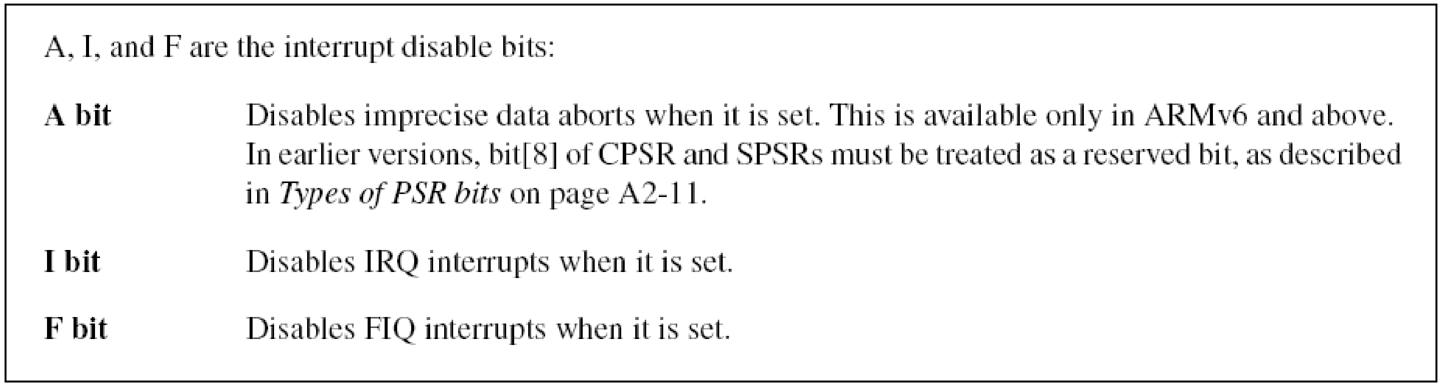


CPSR in ARM



N	Is set to bit 31 of the result of the instruction. If this result is regarded as a two's complement signed integer, then N = 1 if the result is negative and N = 0 if it is positive or zero.
Z	Is set to 1 if the result of the instruction is zero (this often indicates an *equal* result from a comparison), and to 0 otherwise.
C	Is set in one of four ways: • For an addition, including the comparison instruction `CMN`, C is set to 1 if the addition produced a carry (that is, an unsigned overflow), and to 0 otherwise. • For a subtraction, including the comparison instruction `CMP`, C is set to 0 if the subtraction produced a borrow (that is, an unsigned underflow), and to 1 otherwise. • For non-addition/subtractions that incorporate a shift operation, C is set to the last bit shifted out of the value by the shifter. • For other non-addition/subtractions, C is normally left unchanged (but see the individual instruction descriptions for any special cases).
V	Is set in one of two ways: • For an addition or subtraction, V is set to 1 if signed overflow occurred, regarding the operands and result as two's complement signed integers. • For non-addition/subtractions, V is normally left unchanged (but see the individual instruction descriptions for any special cases).

CPSR bits



A, I, and F are the interrupt disable bits:

A bit	Disables imprecise data aborts when it is set. This is available only in ARMv6 and above. In earlier versions, bit[8] of CPSR and SPSRs must be treated as a reserved bit, as described in *Types of PSR bits* on page A2-11.
I bit	Disables IRQ interrupts when it is set.
F bit	Disables FIQ interrupts when it is set.

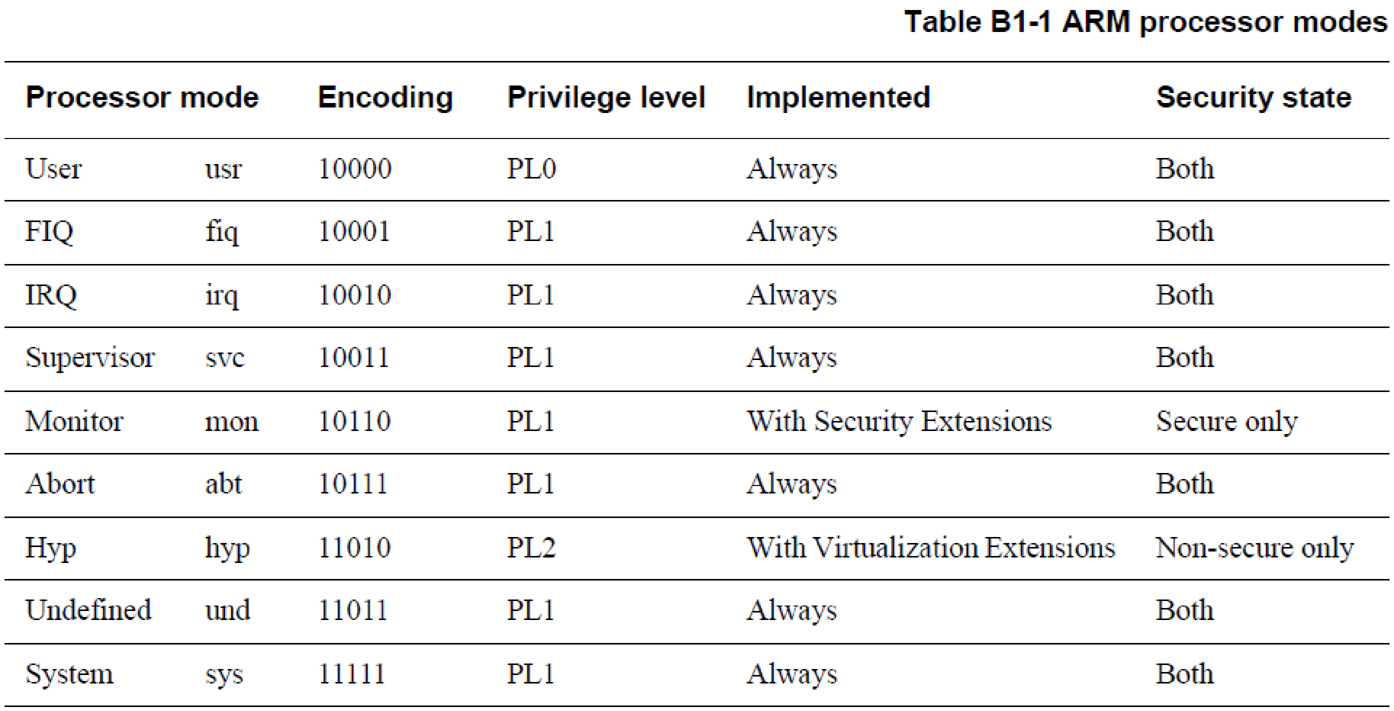


Table B1-1 ARM processor modes

Processor mode		Encoding	Privilege level	Implemented	Security state
User	usr	10000	PL0	Always	Both
FIQ	fiq	10001	PL1	Always	Both
IRQ	irq	10010	PL1	Always	Both
Supervisor	svc	10011	PL1	Always	Both
Monitor	mon	10110	PL1	With Security Extensions	Secure only
Abort	abt	10111	PL1	Always	Both
Hyp	hyp	11010	PL2	With Virtualization Extensions	Non-secure only
Undefined	und	11011	PL1	Always	Both
System	sys	11111	PL1	Always	Both

CPSR bits

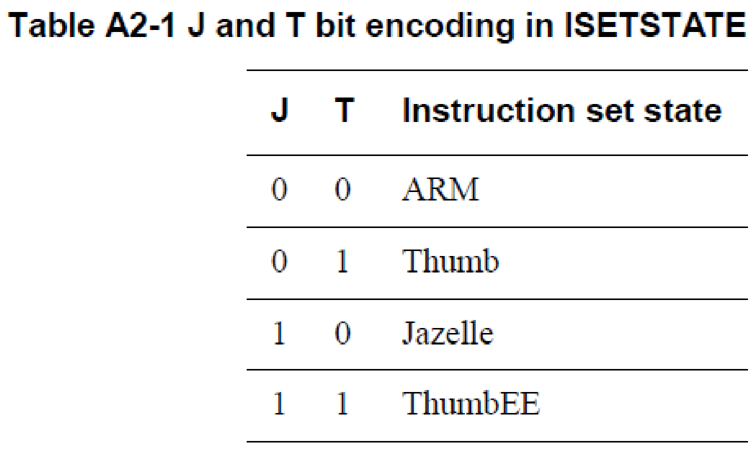


Table A2-1 J and T bit encoding in ISETSTATE

J	T	Instruction set state
0	0	ARM
0	1	Thumb
1	0	Jazelle
1	1	ThumbEE

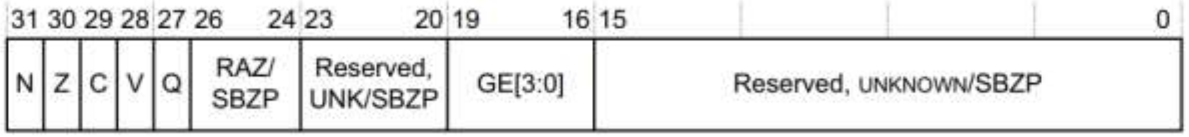
- ARM: 32-bit instructions
- Thumb, Thumb2: 16-bit or 32-bit instructions
- Jazelle: Special mode for JAVA acceleration
- ThumbEE (Thumb Execution Environment)
  - First appeared in 2005 (in Cortex-A8)
  - 4th instruction set state, making small changes to Thumb2
  - ARM **deprecates** any use of ThumbEE ISA and ARMv8 removes support for ThumbEE

APSR (Application Program Status Register)

In ARMv7-A and ARMv7-R, the APSR is the same register as the CPSR, but the APSR must be used only to access the N, Z, C, V, Q, and GE[3:0] bits. For more information, see *Program Status Registers (PSRs)* on page B1-1147.

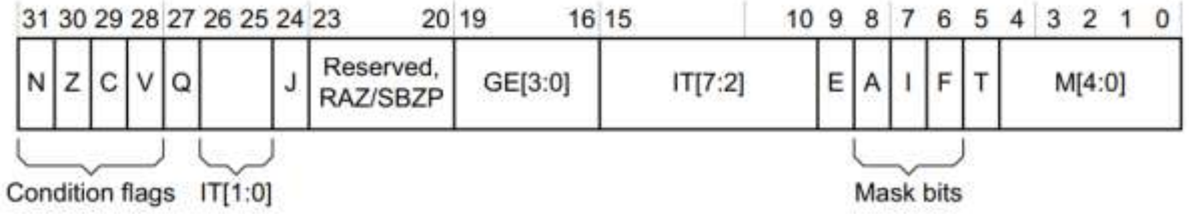
A2.4 The Application Program Status Register (APSR)

Program status is reported in the 32-bit *Application Program Status Register* (APSR). The APSR bit assignments are:



Format of the CPSR and SPSRs

The CPSR and SPSR bit assignments are:



RAZ/SBZP	Read-As-Zero, Should-Be-Zero-or-Preserved on writes.
RAZ/WI	Read-As-Zero, Writes Ignored.

http://esca.korea.ac.kr/teaching/cose321_CSD/labs/DDI0406C_C_armv7AR_architecture_reference_manual.pdf

Interrupt

- **Interrupt** is an asynchronous signal from hardware indicating the need for attention or a synchronous event in software indicating the need for a change in execution.
  - Hardware interrupt causes the processor (CPU) to save its state of execution via a context switch, and begins execution of an interrupt handler.
  - Software interrupt is usually implemented as an instruction in the instruction set, which causes a context switch to an interrupt handler similar to a hardware interrupt.
- Interrupt is a commonly used technique in computer system for communication between CPU and peripheral devices
- Operating systems also extensively use interrupt (timer interrupt) for task (process, thread) scheduling

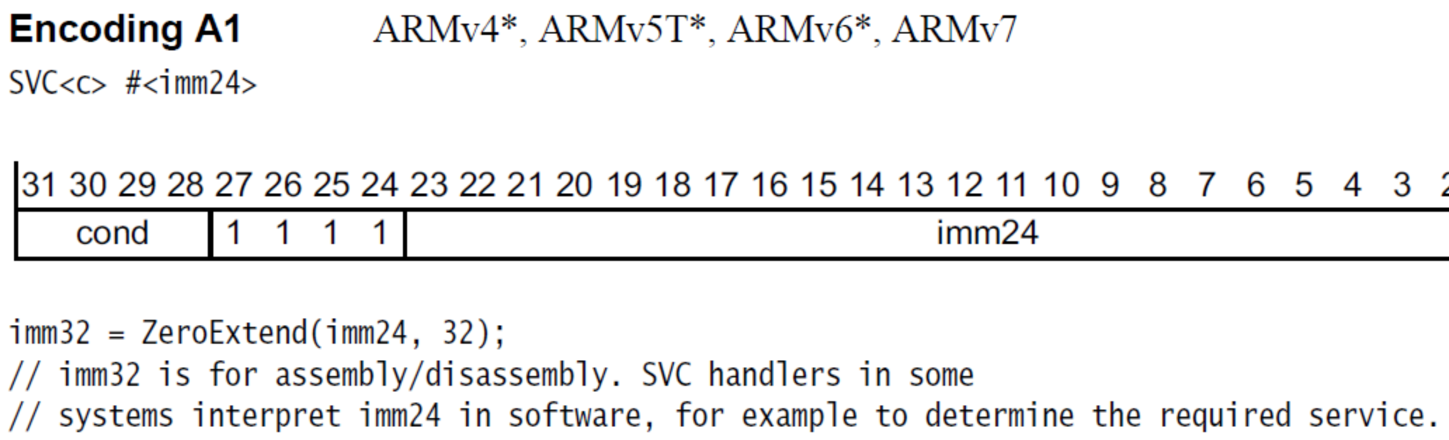
Hardware Interrupt in ARM

- IRQ (Normal interrupt request)
  - Informed to CPU by asserting IRQ pin
  - PC is changed to (vector base + 0x18)
  - ARM goes to IRQ mode

- FIQ (Fast interrupt request)
  - Informed to CPU by asserting FIQ pin
  - Has a higher priority than IRQ
  - PC is changed to (vector base + 0x1C)
  - ARM goes to FIQ mode

Supervisor Call in ARM

- Supervisor Call (previously, Software Interrupt)
  - `SVC` instruction (previously, `SWI` instruction)
  - PC is changed to (vector base + 0x08)
  - ARM goes to Supervisor mode
- Supervisor call is commonly used by OS for system calls
  - Modern OSs have hundreds of system calls (Linux has over 300 system calls)
  - Examples: `open()`, `close()`, `read()`, `write()`, `exit()`.. etc



Encoding A1 ARMv4*, ARMv5T*, ARMv6*, ARMv7

`SVC<c> #<imm24>`

31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond	1 1 1 1	imm24

```
imm32 = ZeroExtend(imm24, 32);
// imm32 is for assembly/disassembly. SVC handlers in some
// systems interpret imm24 in software, for example to determine the required service.
```

Exception Vectors in ARM

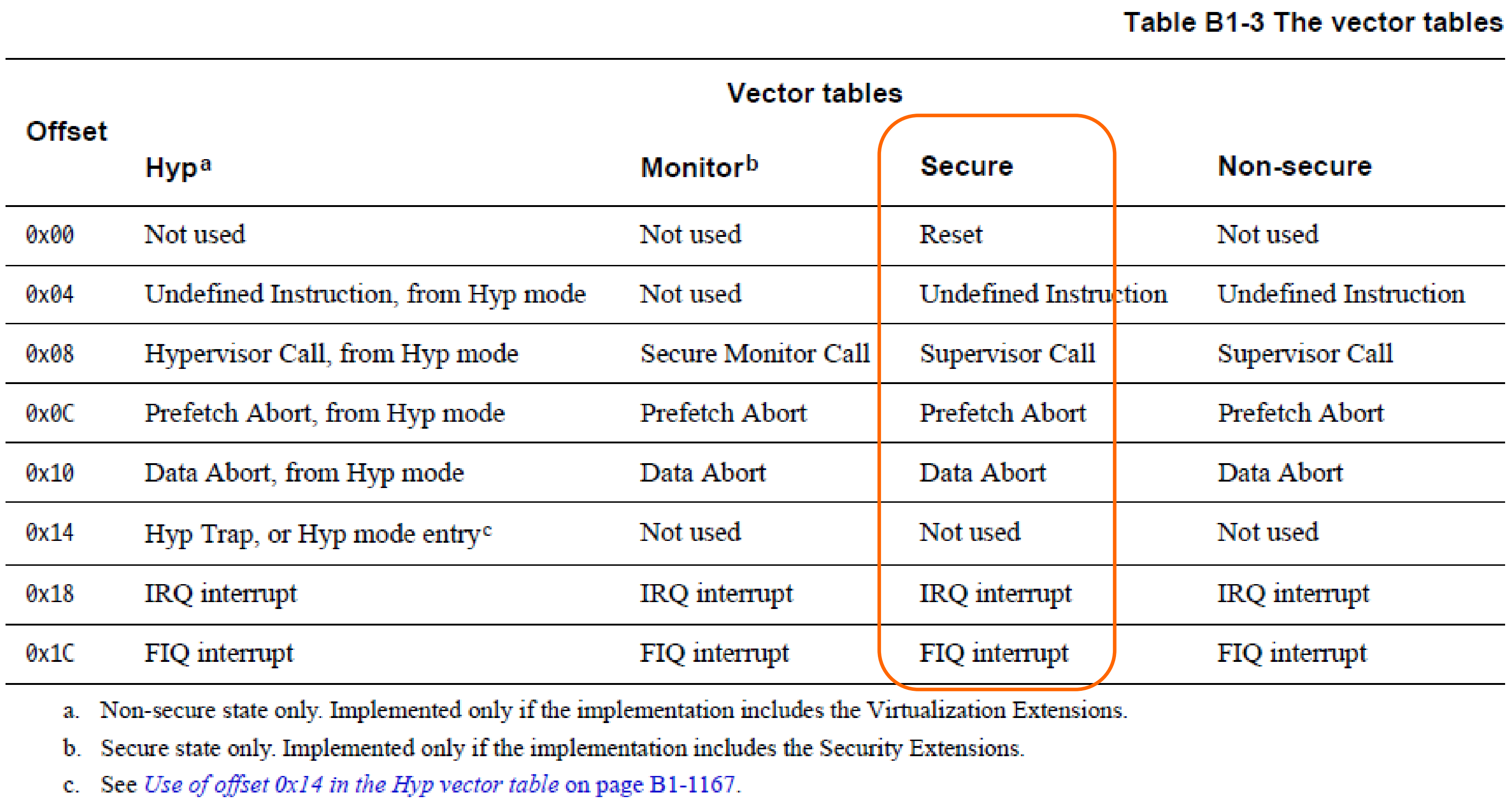


Table B1-3 The vector tables

Offset	Vector tables			
	Hyp[a]	Monitor[b]	Secure	Non-secure
0x00	Not used	Not used	Reset	Not used
0x04	Undefined Instruction, from Hyp mode	Not used	Undefined Instruction	Undefined Instruction
0x08	Hypervisor Call, from Hyp mode	Secure Monitor Call	Supervisor Call	Supervisor Call
0x0C	Prefetch Abort, from Hyp mode	Prefetch Abort	Prefetch Abort	Prefetch Abort
0x10	Data Abort, from Hyp mode	Data Abort	Data Abort	Data Abort
0x14	Hyp Trap, or Hyp mode entry[c]	Not used	Not used	Not used
0x18	IRQ interrupt	IRQ interrupt	IRQ interrupt	IRQ interrupt
0x1C	FIQ interrupt	FIQ interrupt	FIQ interrupt	FIQ interrupt

a. Non-secure state only. Implemented only if the implementation includes the Virtualization Extensions.

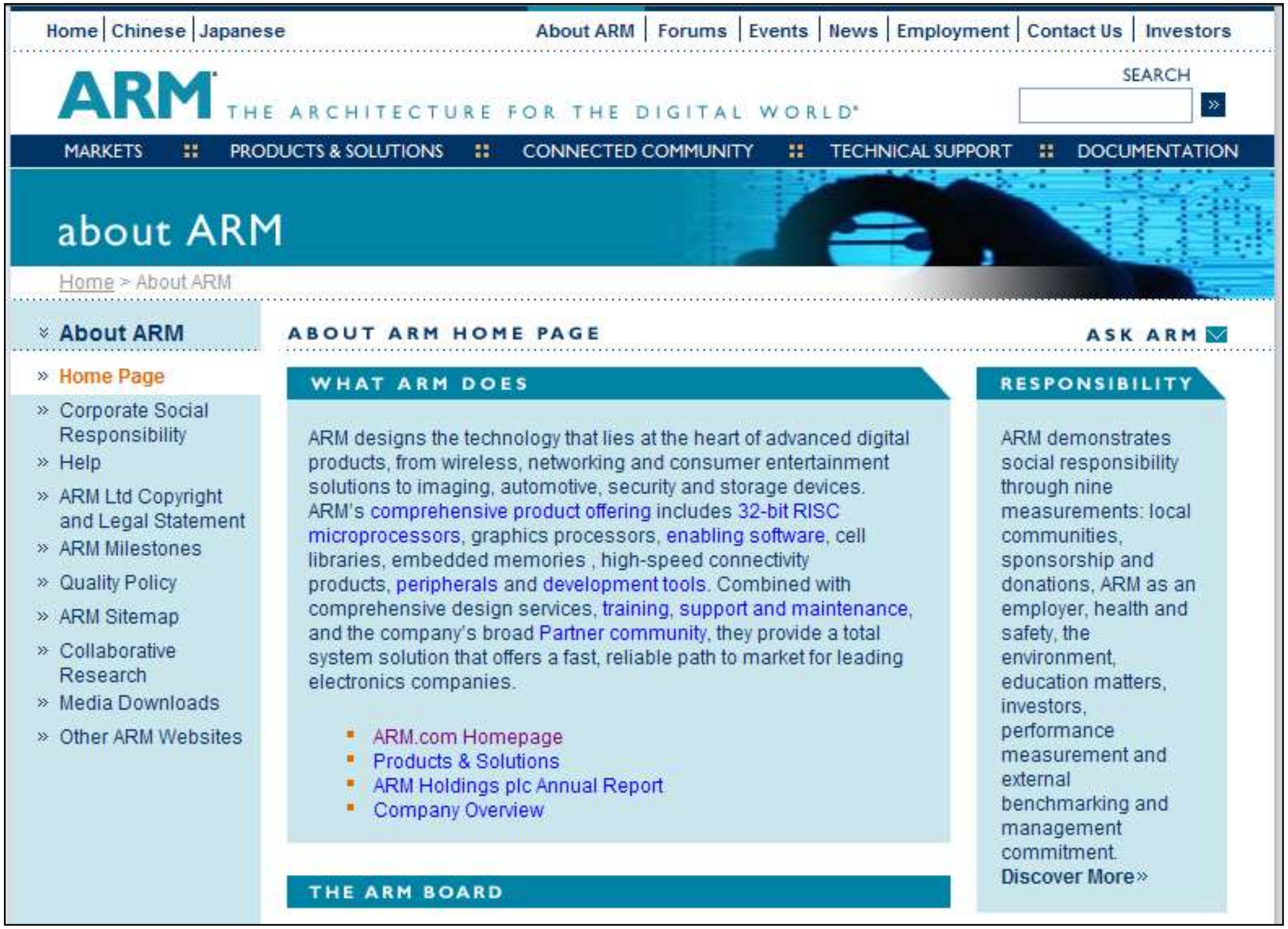
b. Secure state only. Implemented only if the implementation includes the Security Extensions.

c. See *Use of offset 0x14 in the Hyp vector table* on page B1-1167.

RAZ: Read As Zero

Backup Slides

ARM (www.arm.com)



Home | Chinese | Japanese   About ARM | Forums | Events | News | Employment | Contact Us | Investors

ARM THE ARCHITECTURE FOR THE DIGITAL WORLD®

SEARCH

MARKETS :: PRODUCTS & SOLUTIONS :: CONNECTED COMMUNITY :: TECHNICAL SUPPORT :: DOCUMENTATION

about ARM

Home > About ARM

About ARM

- Home Page
- Corporate Social Responsibility
- Help
- ARM Ltd Copyright and Legal Statement
- ARM Milestones
- Quality Policy
- ARM Sitemap
- Collaborative Research
- Media Downloads
- Other ARM Websites

ABOUT ARM HOME PAGE   ASK ARM

WHAT ARM DOES

ARM designs the technology that lies at the heart of advanced digital products, from wireless, networking and consumer entertainment solutions to imaging, automotive, security and storage devices. ARM's comprehensive product offering includes 32-bit RISC microprocessors, graphics processors, enabling software, cell libraries, embedded memories , high-speed connectivity products, peripherals and development tools. Combined with comprehensive design services, training, support and maintenance, and the company's broad Partner community, they provide a total system solution that offers a fast, reliable path to market for leading electronics companies.

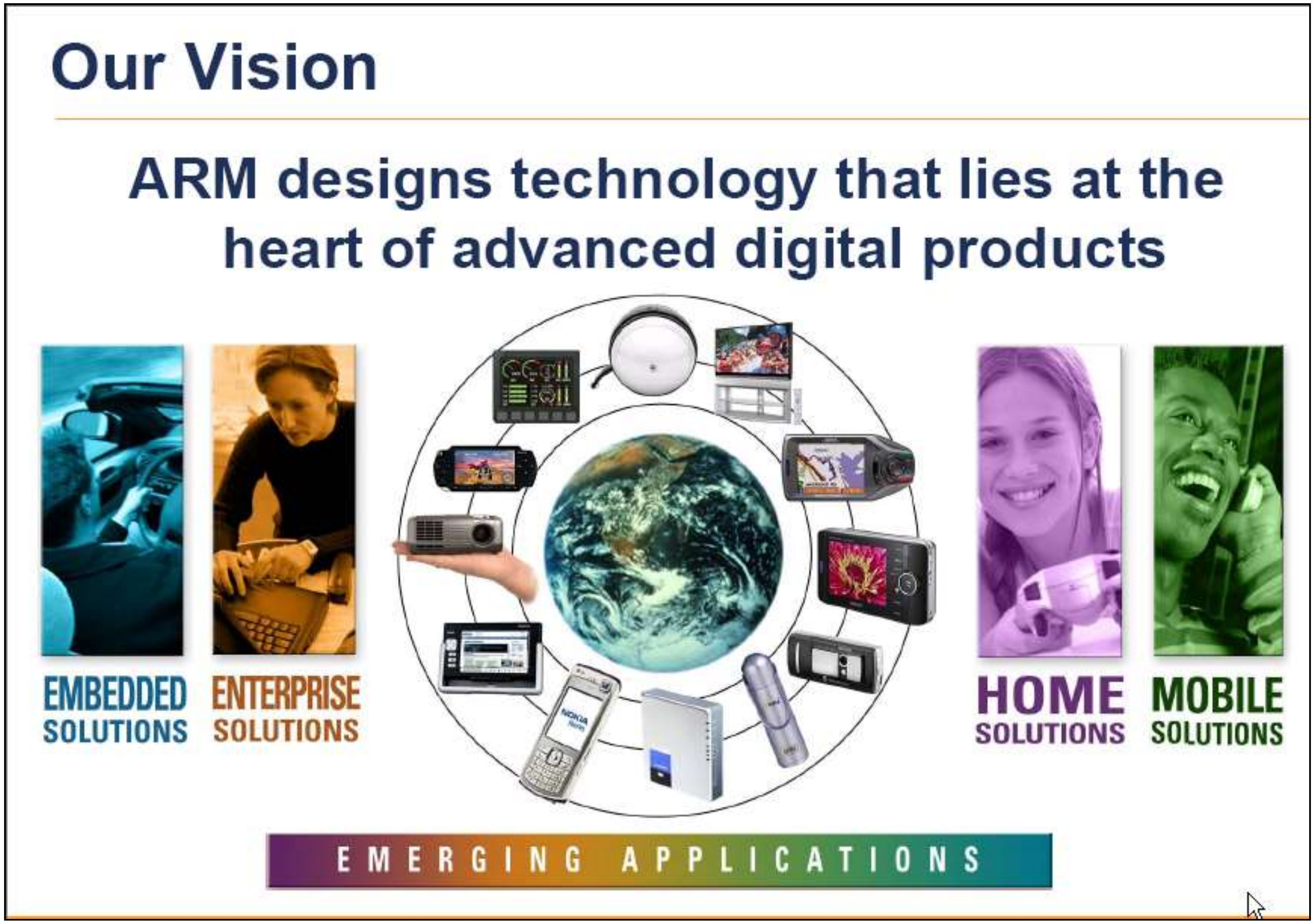
- ARM.com Homepage
- Products & Solutions
- ARM Holdings plc Annual Report
- Company Overview

THE ARM BOARD

RESPONSIBILITY

ARM demonstrates social responsibility through nine measurements: local communities, sponsorship and donations, ARM as an employer, health and safety, the environment, education matters, investors, performance measurement and external benchmarking and management commitment.
Discover More»

ARM



Our Vision

ARM designs technology that lies at the heart of advanced digital products

EMBEDDED SOLUTIONS

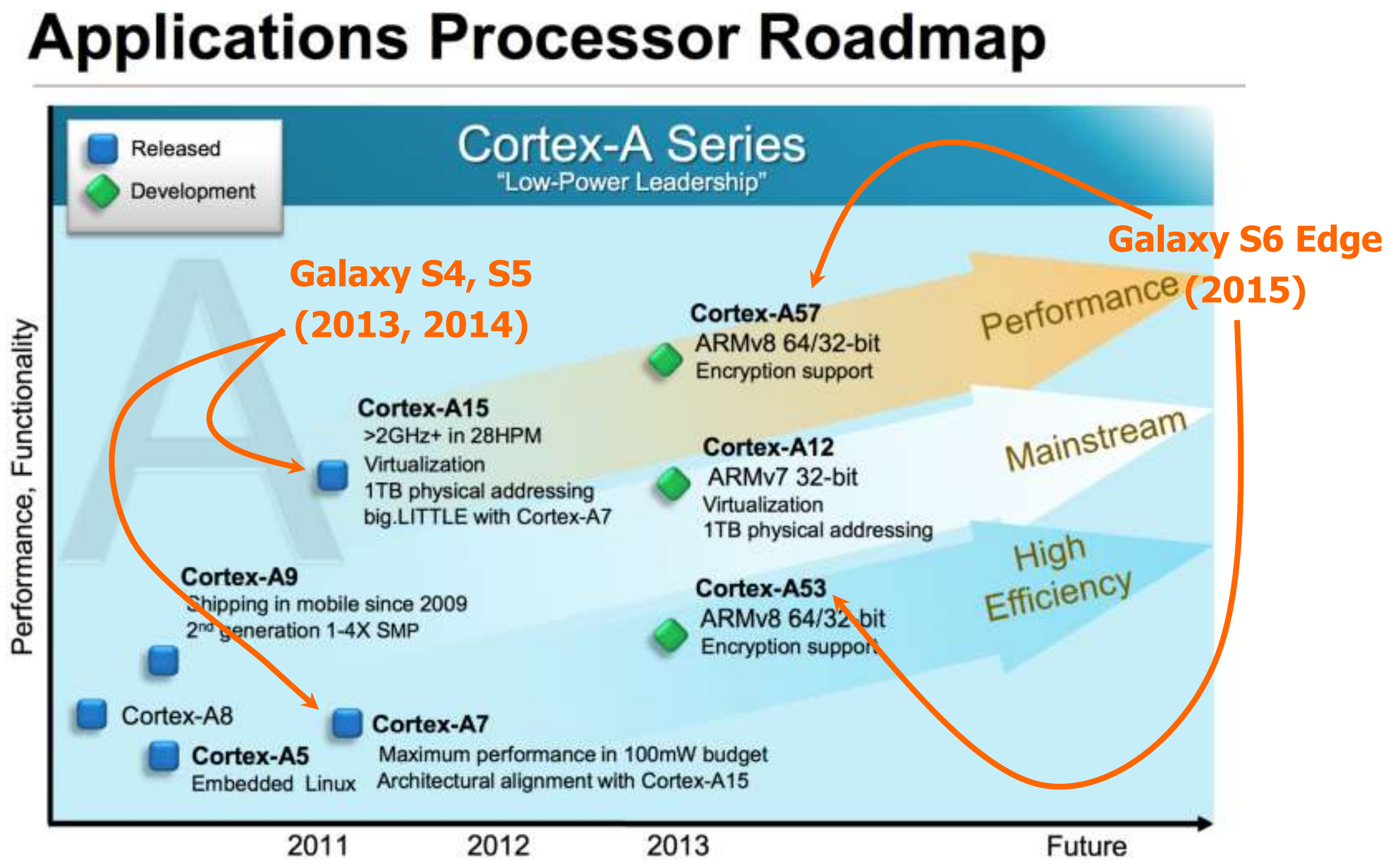
ENTERPRISE SOLUTIONS

HOME SOLUTIONS

MOBILE SOLUTIONS

EMERGING APPLICATIONS

ARM Processor Portfolio (Cont.)



Exception Vectors in ARM

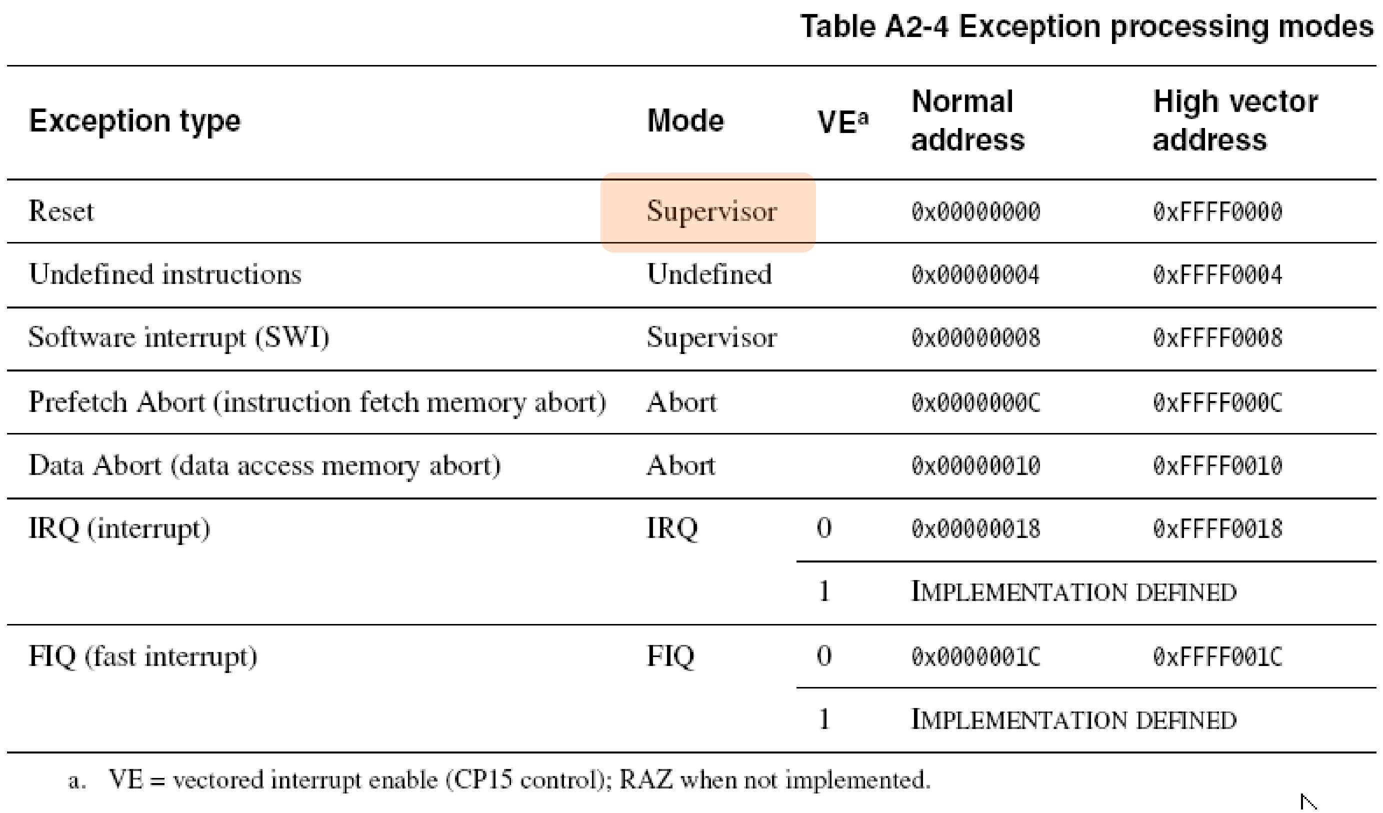


Table A2-4 Exception processing modes

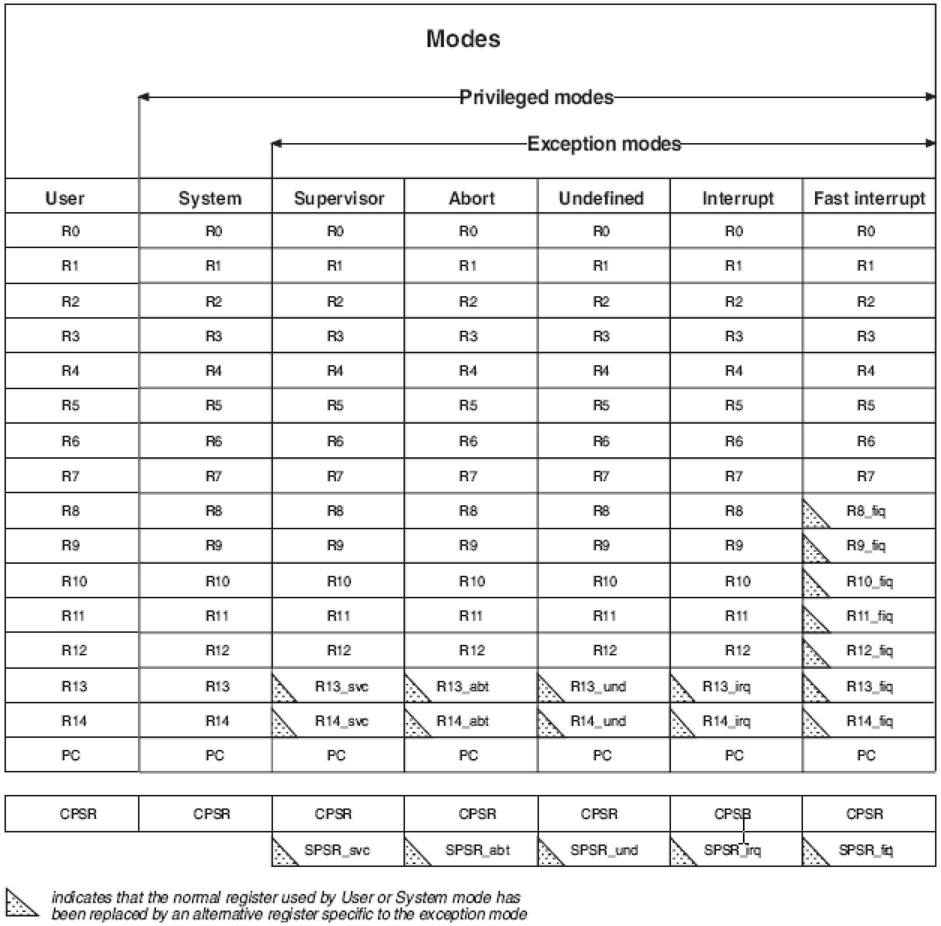
Exception type	Mode	VE[a]	Normal address	High vector address
Reset	Supervisor		0x00000000	0xFFFF0000
Undefined instructions	Undefined		0x00000004	0xFFFF0004
Software interrupt (SWI)	Supervisor		0x00000008	0xFFFF0008
Prefetch Abort (instruction fetch memory abort)	Abort		0x0000000C	0xFFFF000C
Data Abort (data access memory abort)	Abort		0x00000010	0xFFFF0010
IRQ (interrupt)	IRQ	0	0x00000018	0xFFFF0018
		1	IMPLEMENTATION DEFINED	
FIQ (fast interrupt)	FIQ	0	0x0000001C	0xFFFF001C
		1	IMPLEMENTATION DEFINED	

a. VE = vectored interrupt enable (CP15 control); RAZ when not implemented.

RAZ: Read As Zero

ARM Registers

- ARM has 31 general purpose registers and 6 status registers (32-bit each)



Modes						
	Privileged modes					
		Exception modes				
User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	◺ R8_fiq
R9	R9	R9	R9	R9	R9	◺ R9_fiq
R10	R10	R10	R10	R10	R10	◺ R10_fiq
R11	R11	R11	R11	R11	R11	◺ R11_fiq
R12	R12	R12	R12	R12	R12	◺ R12_fiq
R13	R13	◺ R13_svc	◺ R13_abt	◺ R13_und	◺ R13_irq	◺ R13_fiq
R14	R14	◺ R14_svc	◺ R14_abt	◺ R14_und	◺ R14_irq	◺ R14_fiq
PC	PC	PC	PC	PC	PC	PC

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		◺ SPSR_svc	◺ SPSR_abt	◺ SPSR_und	◺ SPSR_irq	◺ SPSR_fiq

◺ *indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode*

Levels of Program Code (x86)

Code with High-level Language

```
int main()
{
   int a, b, c;
   a = 3;
   b = 9;
   c = a + b;
   return c;
}
```

C Compiler

Machine Code

```
a = 3;
c7 45 f0 03 00 00 00  movl   $0x3,-0x10(%ebp)
b = 9;
c7 45 f4 09 00 00 00  movl   $0x9,-0xc(%ebp)

c = a + b;
8b 55 f4              mov    -0xc(%ebp),%edx
8b 45 f0              mov    -0x10(%ebp),%eax
01 d0                 add    %edx,%eax
89 45 f8              mov    %eax,-0x8(%ebp)
```

Representation in hexadecimal (machine-readable)

Instructions (human-readable)