

18-349: Embedded Real-Time Systems

Lecture 2: ARM Architecture

Anthony Rowe

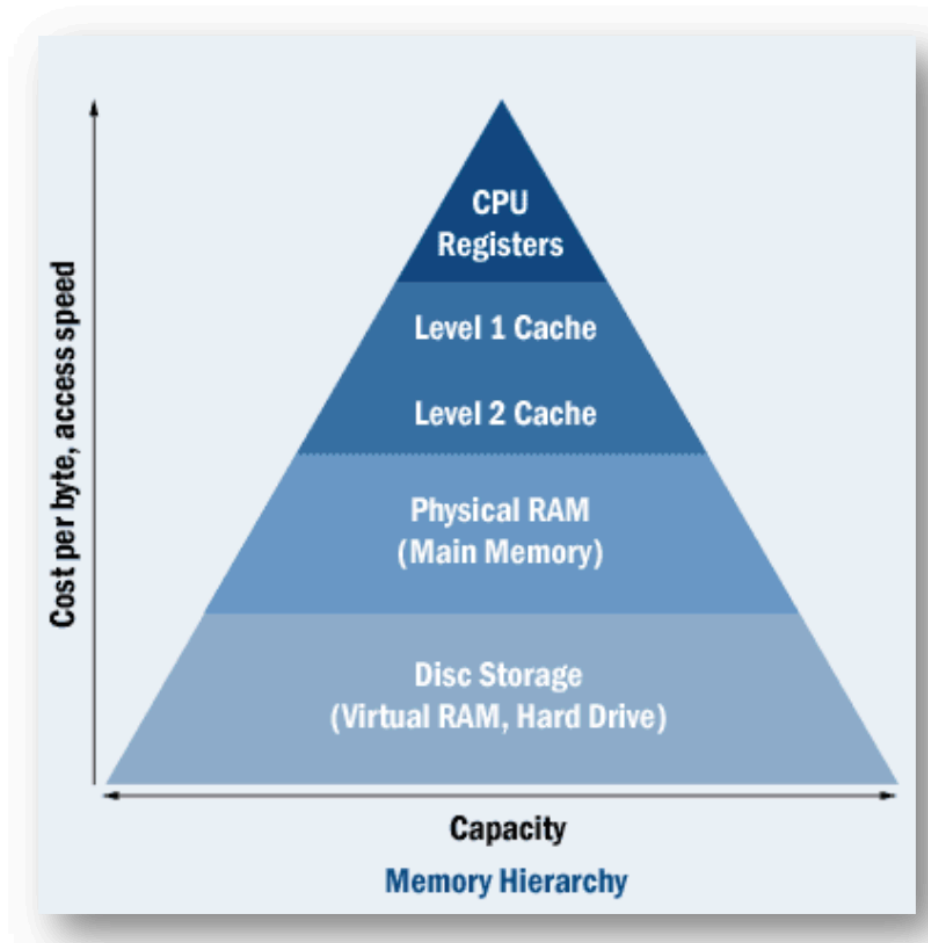
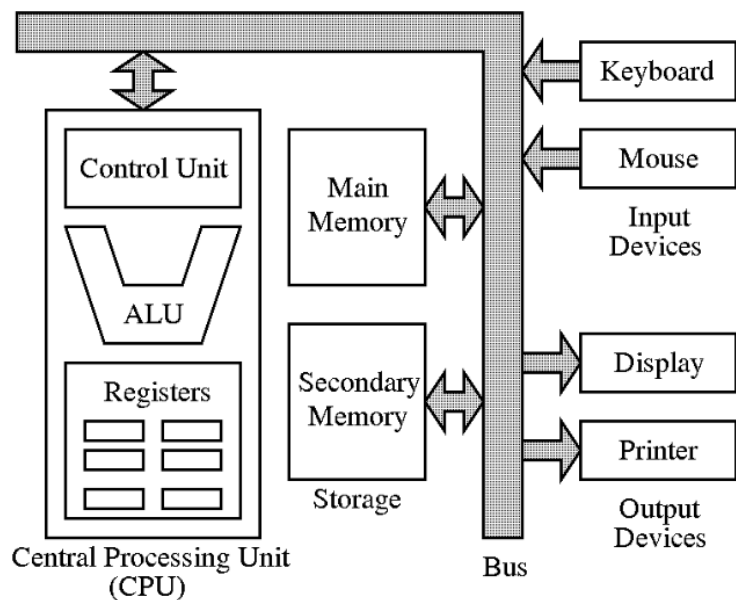
Electrical and Computer Engineering
Carnegie Mellon University



Electrical & Computer
ENGINEERING

Carnegie Mellon University

Basic Computer Architecture



Memory Types



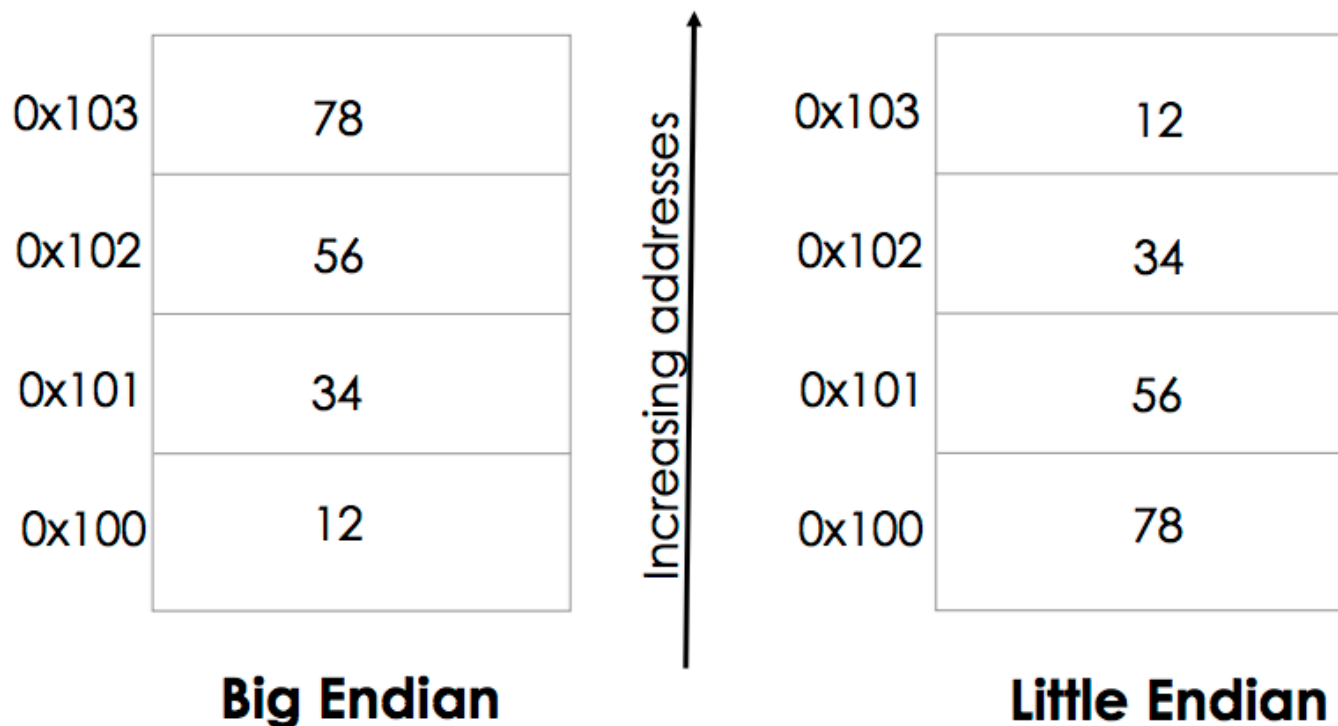
- **DRAM: Dynamic Random Access Memory**
 - Upside: very dense (1 transistor per bit) and cheap
 - Downside: requires refresh and often slow
 - Used as main memory

- **SRAM: Static Random Access Memory**
 - Upside: fast and no refresh required
 - Downside: not so dense, not so cheap
 - Often used for caches

- **EEPROM: Electronically Erasable Programmable Read-only Memory**
 - Used for bootstrapping
 - Require wear-leveling

Big Endian vs. Little Endian

- How is a word, say, 0x1234567 stored in memory?



Big Endian vs. Little Endian

- **Big-endian (big end first)**
 - Most significant byte of any multi-byte data field is stored at the lowest memory address
 - Reading from left to right
 - SPARC & Motorola
- **Little-endian (little end first)**
 - Least significant byte of any multi-byte data field is stored at the lowest memory address
 - Reading from right to left instead
 - Intel processors
- **Bi-endian (ARM, PowerPC, Alpha)**

CISC vs. RISC

- RISC – Reduced Instruction Set Computers
- CISC – Complex Instruction Set Computers
- Different architectures for doing the same operations
- Suppose you wanted to multiply two numbers in memory locations *mem0* & *mem1* and store the results back in *mem0*
 - Same result but the complexity of operations and the number of steps used in the two cases differ

CISC Approach	RISC Approach
<code>mull mem0, mem1</code>	<code>ldr r0, mem0</code> <code>ldr r1, mem1</code> <code>mul r0, r0, r1</code> <code>str mem0, r0</code>

CISC vs. RISC (1)

CISC	RISC
Example – Intel x86 chips	Examples – SPARC, PowerPC, ARM
Large number of instructions	Few instructions, typically less than 100
Variable-length instructions, instructions can range from 1-15 bytes	Fixed-length instructions, all instructions have the same number of bytes
Some instructions can have long execution times	No instruction with a long execution times execution time

CISC vs. RISC (2)

CISC	RISC
Arithmetic and logical operations can be applied to memory and register operands	Arithmetic and logical operations only use register operands <ul style="list-style-type: none">• Memory contents have to be loaded into registers first• Referred to as load/store architecture
Stack-intensive procedure linkage <ul style="list-style-type: none">• Stack is used for procedure arguments and return values	Register-intensive procedure linkage <ul style="list-style-type: none">• Registers used for procedure arguments and return values

CISC vs. RISC (3)

CISC	RISC
Compact code size is typically small	Compiled code size is larger
More transistors = more power	Fewer transistors = less power

ARM, Ltd.



- **Founded in November 1990**
 - Spun out of Acorn Computers based in U.K.
 - ARM was originally Acorn RISC Machine; then, Advanced RISC Machine
- **Most widely used 32-bit instruction-set architecture in terms of volume**
 - 6.1 billion ARM processors in 2010 up to 15 billion in 2015
 - 95% of all smartphones, 35% of digital TVs and set-top boxes
- **ARM architecture can be licensed, with licensees (former and/or current)**
 - AMD, Apple, Freescale, Microsoft, Nintendo, Xilinx, Qualcomm, TI, etc.
- **Companies design custom CPU cores with ARM instruction set**
 - Qualcomm's Snapdragon, Apple's A8, etc.
- **ARM Ltd. does not fabricate processors itself**
 - Also develops technologies to help with the design-in of ARM devices
 - Software tools, boards, debug hardware, application software, buses, peripherals

History of ARM's Usage

Example ARM component	Architecture Generation	Example Application	Approximate date of introduction
ARM1	ARMv1	Acorn Computer in internal testing	1985
ARM2	ARMv2	Acorn Archimedes (Macintosh-era PC)	1987
ARM6	ARMv3	Apple Newton MessagePad 100 series	1994
ARM7TDMI	ARMv4	Game Boy Advance, Nintendo DS*, iPod	2001
ARM9E	ARMv5	Nintendo DS*, Nokia N-Gage, Airport Extreme N basestation	2004
ARM11	ARMv6	iPhone, iPhone 3G, iPod touch	2007
Cortex-A8	ARMv7	Palm Pre, iPhone 3GS	2009

SoftBank Acquisition

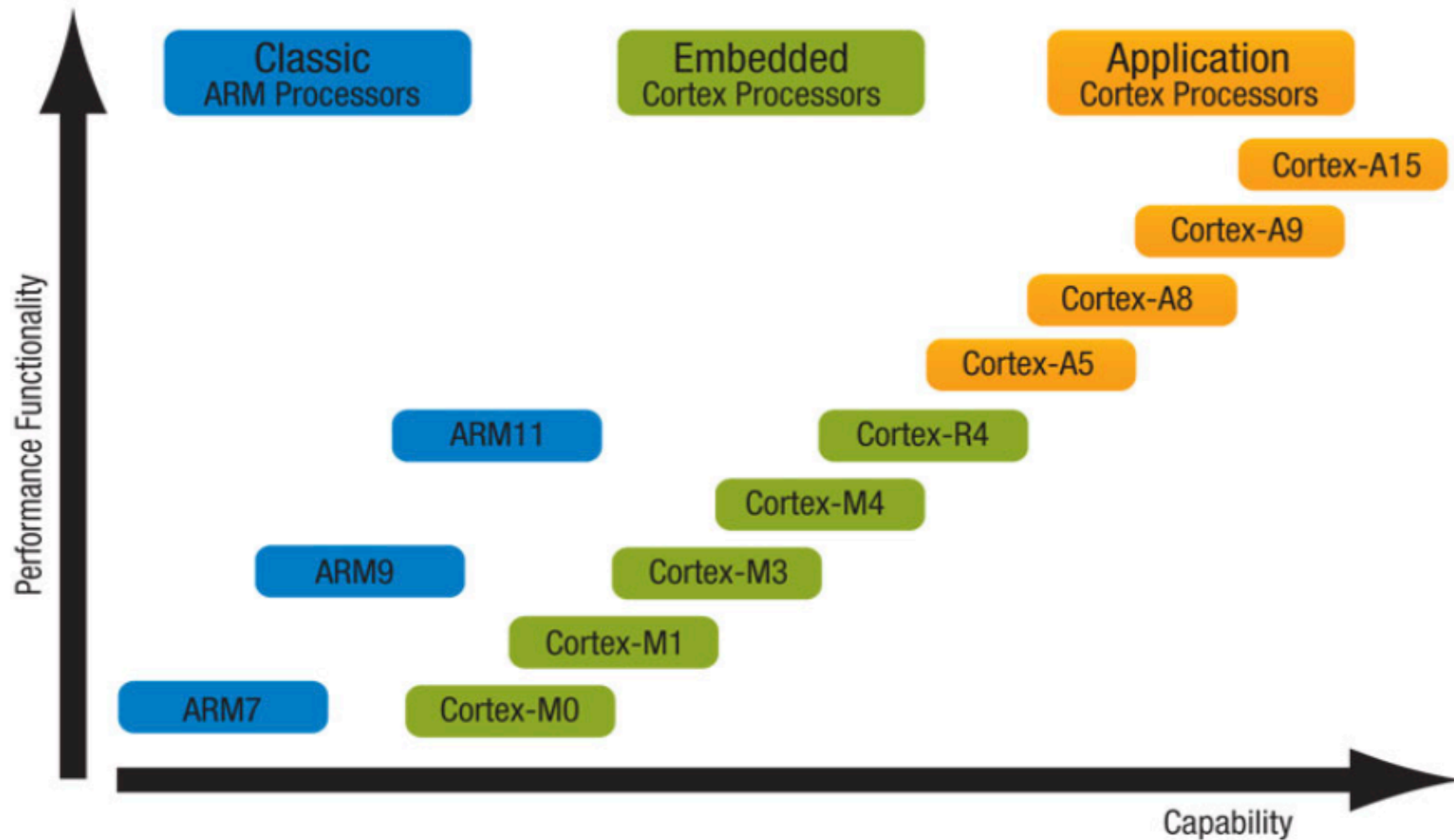
- Japanese multinational company
- July 18th 2016
- Purchased for £23.4 billion
- Speculating on IoT market

ARM Everywhere



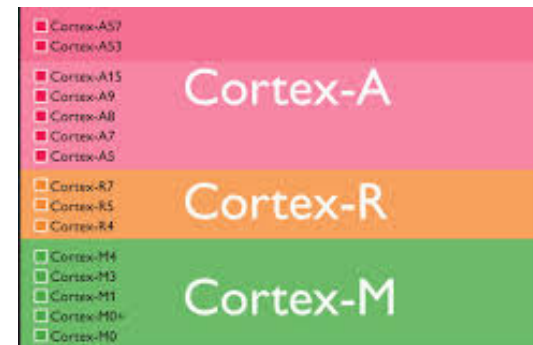
<p>Mobile</p>	<p>3% ↑ in 2012</p>	
<p>Embedded</p>	<p>25% ↑ in 2012</p>	
<p>Enterprise</p>	<p>10% ↑ in 2012</p>	
<p>Home</p>	<p>40% ↑ in 2012</p>	

The ARM Family

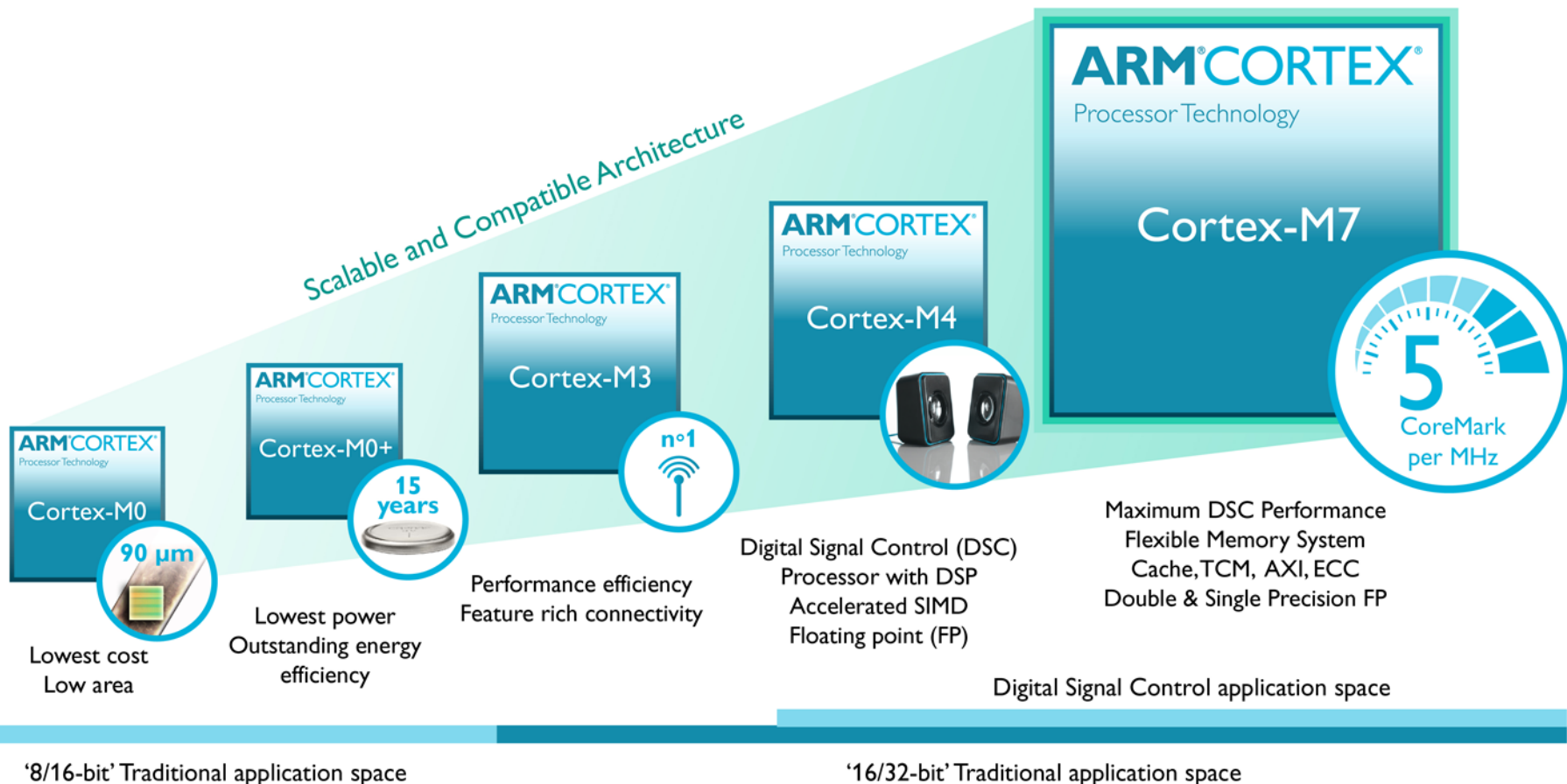


Different ARM Core Families

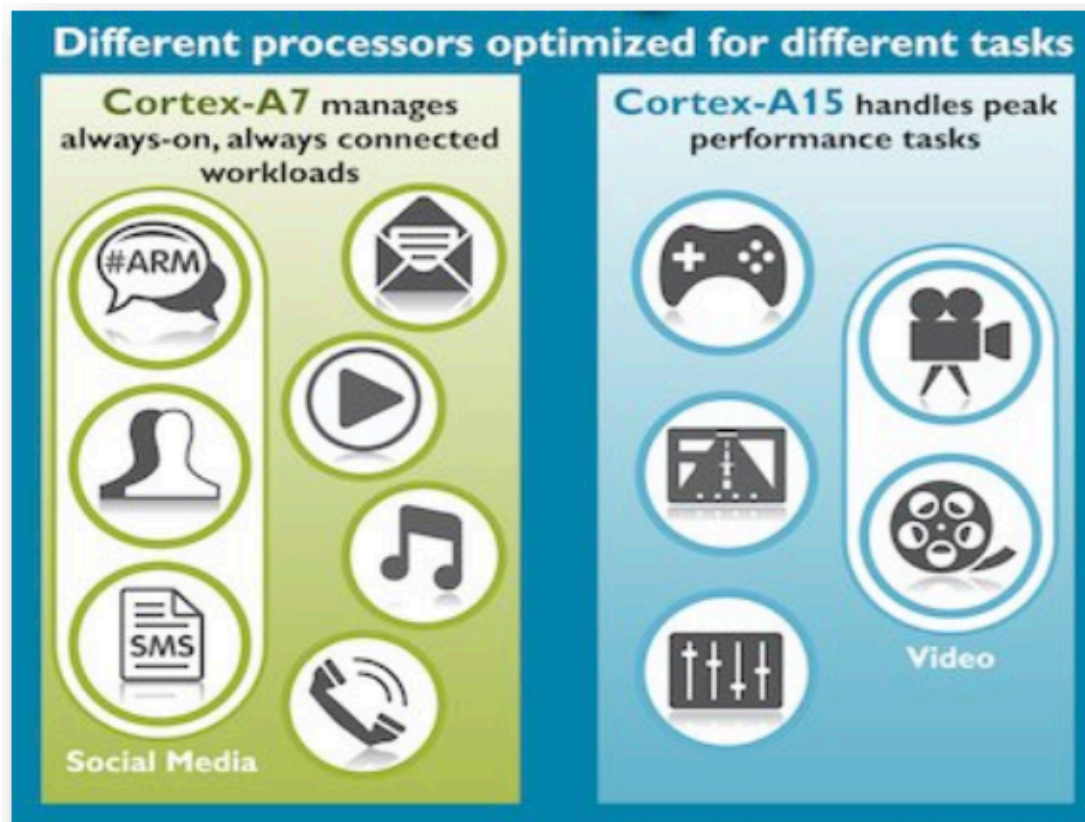
- Cortex-A
 - Application processors
 - Single-core or multi-core
 - Optional multimedia processing
 - Optional floating-point units
 - Smartphones, tablets, digital TVs, eBook readers
- Cortex-R
 - Deeply embedded real-time applications
 - Low power, good interrupt behavior with good performance
 - Automotive braking systems, printers, storage controllers
- Cortex-M
 - Cost-sensitive microcontrollers
 - Fast, deterministic interrupt management
 - Lowest possible power consumption
 - Automotive airbags, tire-pressure monitoring, smart meters, sensors



Cortex-M Family



Beyond Cortex



Break?

ARM Data Sizes & Instructions

- The ARM is a 32-bit RISC architecture
- When used in relation to the ARM
 - **Byte** means 8-bits
 - **Halfword** means 16 bits (two bytes)
 - **Word** means 32 bits (four bytes)
- Most ARM processors implement two instructions sets
 - 32-bit ARM Instructions Set
 - 16-bit Thumb Instruction Set
- Bi-endian
 - Can be configured to view words stored in memory as either Big-endian or Little-Endian Format

ARM is a RISC Architecture

- A large array of uniform registers
- A load/store model, where
 - Operations operate only on register and not directly on memory
 - All data must be loaded into registers before being used
 - Result (in a register) can be further processed or stored to memory
- A small number of addressing modes
 - All load / store addresses are determined from register and instruction fields
- A uniform fixed-length instruction (32-bit)

Programmer's Model

- ARM supports seven processor modes
 - Characterized by specific behavior, privileges, associated registers
 - Mode changes can be made under software control, or be caused by external interrupts or exception processing
- Most applications execute in **User mode**
 - Program cannot access certain protected resources
 - Program cannot change mode without causing an exception
- The 6 modes other than user mode are called **privileged modes**
 - 5 of these privileged modes are called **exception modes**
 - The remaining one is called the **System mode** (same as User mode, but with access to protected resources)

The Seven Modes

Processor mode		Description
User	usr	Normal program execution mode
FIQ	fiq	Fast Interrupt for high-speed data transfer
IRQ	irq	Used for general-purpose interrupt handling
Supervisor	svc	A protected mode for the operating system
Abort	abt	Implements virtual memory and/or memory protection
Undefined	und	Supports software emulation of hardware coprocessors
System	sys	Runs privileged operating system tasks

Register Set (1)



- ARM has a total of 37 registers all of which are 32-bit
 - 30 general-purpose registers
 - 1 dedicated program counter (pc)
 - 1 dedicated current program status register (cpsr)
 - 5 dedicated saved program status registers (spsr)
- In any mode, only a subset of these 37 registers are visible
 - The hidden registers are called **banked registers**
 - The current processor-mode governs which registers are visible

Register Set (2)
















- **r0 through r7**: Eight general-purpose registers that are always available, no matter which mode you're in (8)
- **r8 through r12**: Five general-purpose registers that are common to all processor modes other than fiq mode (5)
- **r8_fiq through r12_fiq**: Five registers that replace the normal r8-r12 when the processor is in fiq mode (5)
- **Special-purpose registers**
 - **r13 (stack pointer)**: Same for System and User mode, otherwise, r13_fiq, r13_svc, r13_abt, r13_irq, r13_und (6)
 - **r14 (link register)**: Same for System and User mode, otherwise, r14_fiq, r14_svc, r14_abt, r14_irq, r14_und (6)
 - r15 (program counter): A unique one across all modes (1)



Register Set (3)






- Status registers
 - **cpsr (current program status register)**: Holds current status of processor, including its mode (1)
 - **spsr (saved program status register)**: Holds processor status information before program changes into an exception mode, r13_fiq, r13_svc, r13_abt, r13_irq, r13_und (5)

Register Set (4)

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0 1	R0	R0	R0	R0	R0
R1 2	R1	R1	R1	R1	R1
R2 3	R2	R2	R2	R2	R2
R3 4	R3	R3	R3	R3	R3
R4 5	R4	R4	R4	R4	R4
R5 6	R5	R5	R5	R5	R5
R6 7	R6	R6	R6	R6	R6
R7 8	R7	R7	R7	R7	R7
R8 9	 R8_fiq 14	R8	R8	R8	R8
R9 10	 R9_fiq 15	R9	R9	R9	R9
R10 11	 R10_fiq 16	R10	R10	R10	R10
R11 12	 R11_fiq 17	R11	R11	R11	R11
R12 13	 R12_fiq 18	R12	R12	R12	R12
R13 19	 R13_fiq 21	 R13_svc 23	 R13_abt 25	 R13_irq 27	 R13_und 29
R14 20	 R14_fiq 22	 R14_svc 24	 R14_abt 26	 R14_irq 28	 R14_und 30
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

31

ARM State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	32
	 SPSR_fiq 33	 SPSR_svc 34	 SPSR_abt 35	 SPSR_irq 36	 SPSR_und 37	 = banked register

Banked Registers



- Banking of registers implies
 - The specific register depends not only on the number (r_0, r_1, \dots, r_{15}) but also on the processor mode
- Values stored in banked registers are preserved across mode changes
- Example: Assume that the processor is executing in User Mode
 - In **User** mode, assume that the processor writes 0 in r_0 and 8 in r_8
 - Processor now changes to **fiq** mode
 - In FIQ mode, the value of r_0 is _____
 - If processor now overwrites both r_0 and r_8 with 1 in **fiq** mode and changes back to **user** mode
 - The new value stored in r_0 (user mode) is _____
 - The new value stored in r_8 (user mode) is _____

Register Set in User Mode

Current Visible Registers

User Mode

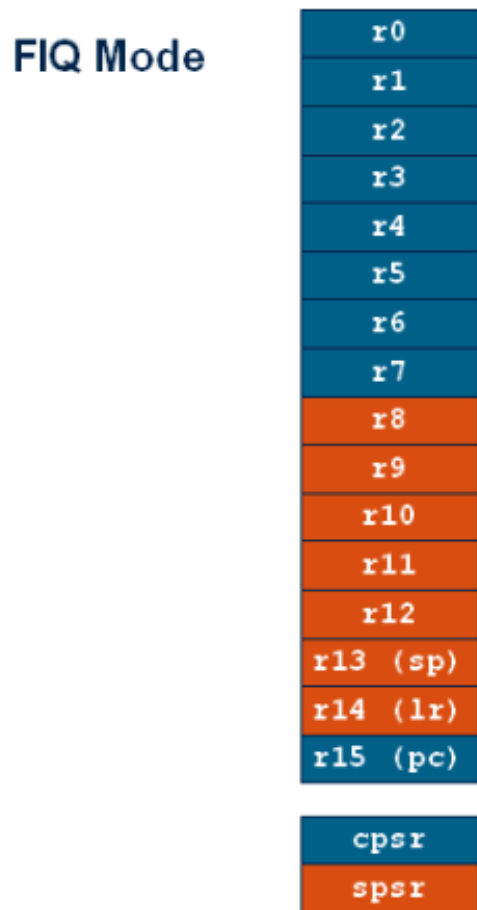
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr

Banked out Registers

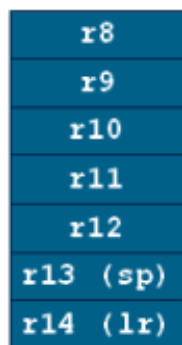
	FIQ	IRQ	SVC	Undef	Abort
	r8				
	r9				
	r10				
	r11				
	r12				
	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
	spsr	spsr	spsr	spsr	spsr

Register Set in `fiq` Mode

Current Visible Registers



User



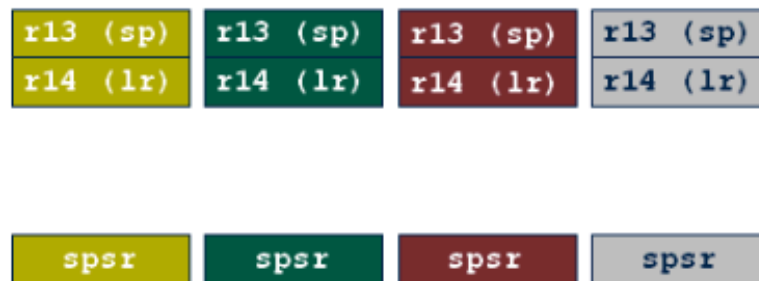
Banked out Registers

IRQ

SVC

Undef

Abort



Thumb Mode



- Thumb is a 16-bit instruction set
 - Optimized for code density from C code (~65% of ARM code size)
 - Improved performance from narrow memory
 - Subset of the functionality of the ARM instruction set

- Core has additional execution state – Thumb
 - Switch between ARM and Thumb using **BX** instruction

- For most instructions generated by compiler:
 - Conditional execution is not used
 - Source and destination registers identical
 - Only Low registers used
 - Constants are of limited size
 - Inline barrel shifter not used

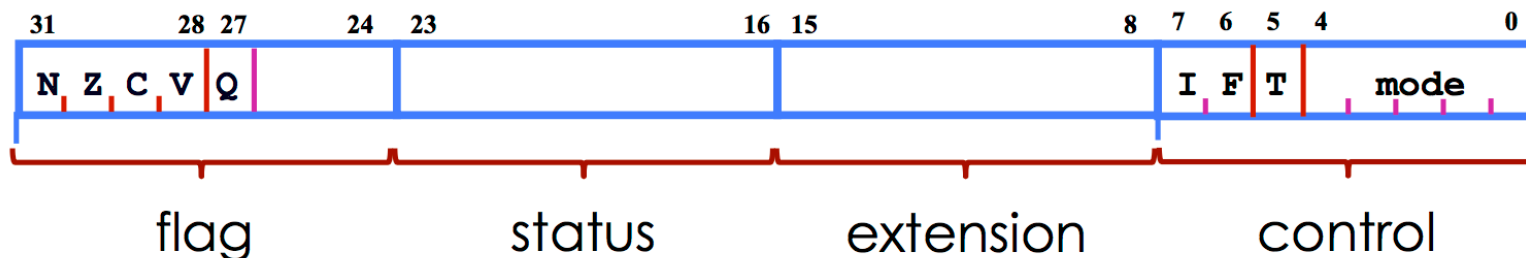
The CPSR (1)

- Current Program Status Register (`cpsr`) is a dedicated register
- Holds information about the most recently performed ALU operations
- Controls the enabling and disabling of interrupts (both IRQ and FIQ)
- Sets the processor operating mode
- Sets the processor state

The CPSR (2)



- cpsr has two important pieces of information
 - Flags: contains the condition flags
 - Control: contains the processor mode, state and interrupt mask bits
- All fields of the cpsr can be read/written in privileged modes
 - **Only the flag field** of cpsr can be written in User mode, all fields can be read in User mode



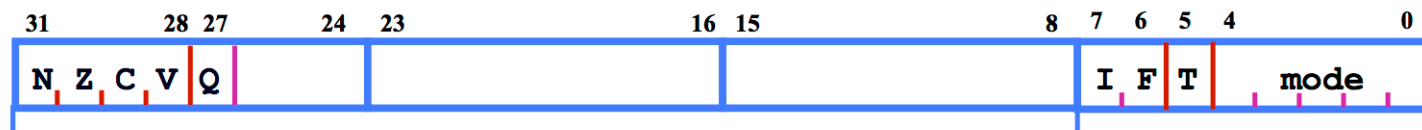
The CPSR (3)



- Interrupt Disable bits
 - I = 1: Disables IRQ
 - F = 1: Disables FIQ
- T Bit
 - T = 0: Processor in ARM state
 - T = 1: Processor in Thumb state
- Mode bits
 - Specify the processor mode

When exceptions occur cpsr gets copied to the corresponding `spsr_<mode>` register for storage

The CPSR (4)



- Will represent this as `nzcqvift_mode`
- Upper case letters will indicate that a certain bit has been set
- Examples
 - `nzcqvift_USER`: FIQs are masked and the processor is executing in user mode
 - `nzCqvift_SVC`: Carry flag is set and the processor is executing in supervisor mode

Exceptions vs. Interrupts

- Term exception and interrupt are often confused!
- Exception usually refers to an internal CPU event such as
 - Floating point overflow
 - MMU fault (e.g., page fault)
 - Trap (SWI)
- *Interrupt* usually refers to an external I/O event such as
 - I/O device request
 - Reset
- In the ARM architecture manuals, the two terms **are** mixed together and are considered interchangeable

ARM Exceptions

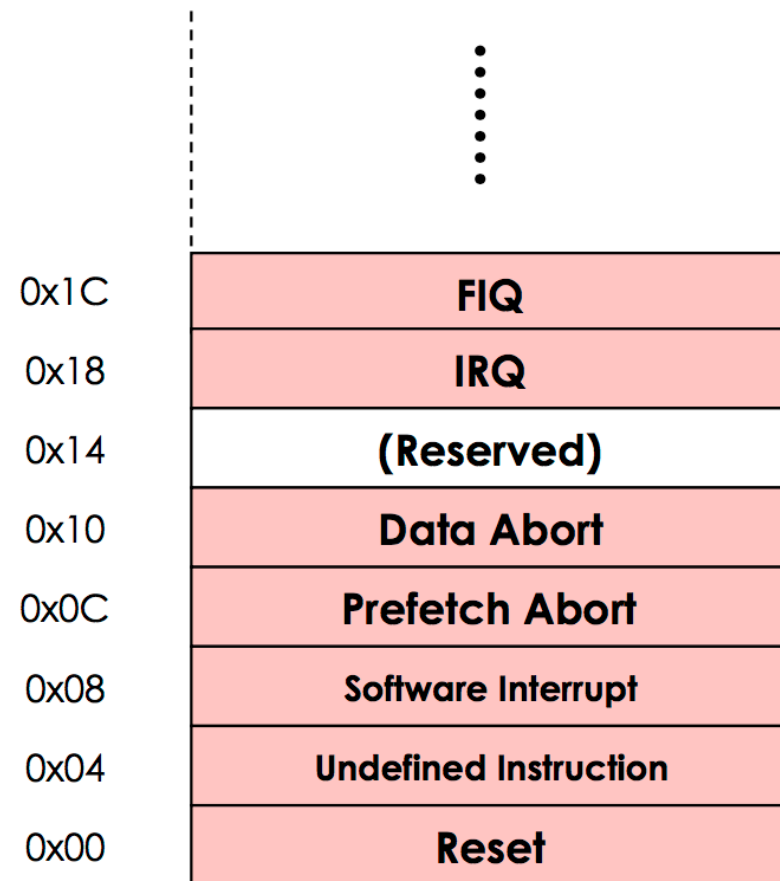
Exception	Mode	Description
Reset	Supervisor	Occurs when the processor's reset button is asserted. This exception is only expected to occur for signaling power up or for resetting the processor. A soft reset can be achieved by branching to reset vector 0x00000000 or letting the watchdog timer expire
Undefined Instruction	Undef	Occurs if neither the processor, nor any of the coprocessors, recognize the currently executing instruction
Software Interrupt	Supervisor	This is a user-defined synchronous interrupt. It allows a program running in the User mode to request privileged operations (for example an RTOS function) that run in Supervisor mode
Prefetch Abort	Abort	Occurs when a processor attempts to execute an instruction that was not fetched, because the address was illegal
Data Abort	Abort	Occurs when a data transfer instruction attempts to load or store data at an illegal address
IRQ	IRQ	Occurs when the processor's external interrupt request pin is asserted and the I bit in the cpsr is clear
FIQ	FIQ	Occurs when the processor's external fast interrupt request pin is asserted and the I (F-bit???) bit in the cpsr is clear

ARM Exception Handling (1)

- **Exception Handler**
 - Most exceptions have an associated software exception handler that executes when that particular exception occurs
- **Exception modes and registers**
 - Handling exceptions changes program from user to non-user mode
 - Each exception handler has access to its own set of registers
 - Its own r13 (stack pointer)
 - Its own r14 (link register)
 - Its own spsr (Saved Program Status Register)
 - Exception handlers must save (restore) other registers on entry (exit)

ARM Exception Handling (2)

- Where is the exception handler located?
- Vector table
 - Reserved area of 32 bytes at the end of the memory map (starting at address 0x0)
 - One word of space for each exception type
 - Contains a Branch or Load PC instruction for the exception handler



Vector Table

ARM Exception Handling (3)

- When an exception occurs, **the ARM processor:**
 - Copies cpsr into spsr_<mode>
 - Sets appropriate cpsr bits
 - Change to ARM state
 - Change to exception mode
 - Disable interrupts (if appropriate)
 - Stores the return address in lr_<mode>
 - Sets pc to vector address
- To return, **exception handler needs to:**
 - Restore cpsr from the spsr_<mode>
 - Restore pc from lr_<mode>

Simultaneous Exceptions?

Vector address	Exception type	Exception mode	Priority (1=high, 6=low)
0x0	Reset	Supervisor (SVC)	1
0x4	Undefined Instruction	Undef	6
0x8	Software Interrupt (SWI)	Supervisor (SVC)	6
0xC	Prefetch Abort	Abort	5
0x10	Data Abort	Abort	2
0x14	<i>Reserved</i>	<i>Not applicable</i>	<i>Not applicable</i>
0x18	Interrupt (IRQ)	Interrupt (IRQ)	4
0x1C	Fast Interrupt (FIQ)	Fast Interrupt (FIQ)	3

Why Exceptions?



- Functionality that would otherwise not be possible
 - Memory or Data Abort can be used to implement Virtual Memory
- SWI allows for system calls
- Undefined exceptions can be used to provide software emulation of coprocessor when the coprocessor is not physically present or could be used for special purpose instruction set extensions
 - If an unknown instruction is reached the processor changes to **Undefined** mode and executes the Undefined Instruction exception handler
 - In the exception handler, the coprocessor functionality can be provided in software (or the functionality provided by the enhanced instructions can be provided in software)

Summary



- What is an ARM?
 - CISC vs RISC
 - ARM family
 - Memory

- ARM Architecture from a programmer's viewpoint
 - Processor modes
 - General purpose registers
 - Special purpose registers
 - Exception handling

- **Next Time: Deeper into ARM ASM**