ARM Processor Fundamentals

Minsoo Ryu

Department of Computer Science and Engineering
Hanyang University

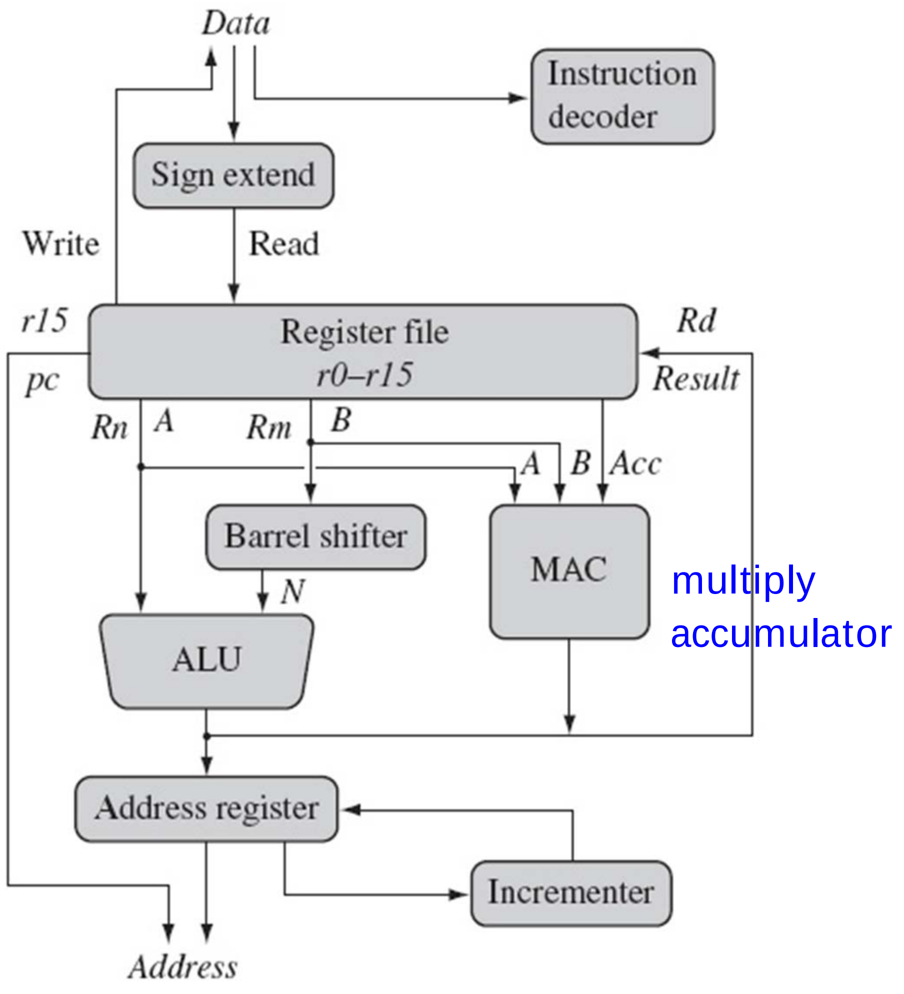
msryu@hanyang.ac.kr

Topics Covered

- ❐ ARM Processor Fundamentals
- ❐ ARM Core Dataflow Model
- ❐ Registers and Current Program Status Register
- ❐ Pipeline
- ❐ Exceptions, Interrupts, and the Vector Table
- ❐ Core Extensions
- ❐ ARM Architecture Revisions and Families

ARM Core Dataflow Model

- An ARM core can be viewed as functional units connected by data buses
- The data may be an <u>instruction or a data item</u>
  - The figure shows <u>a Von Neumann implementation of ARM</u> (data items and instructions share the same bus)
  - Harvard implementations of the ARM use two different buses



ARM Core Dataflow Model

- The instruction decoder translates instructions
- Data items are placed in the register file
  - A storage bank made up of 32-bit registers
  - Most instructions treat the registers as holding signed or unsigned 32-bit values
  - The sign extend hardware converts signed 8-bit and 16-bit numbers into 32-bit values
- ARM instructions typically have two source registers, *Rn* and *Rm*, and a single result or destination register, *Rd*
  - Source operands are read from the register file using the internal bus

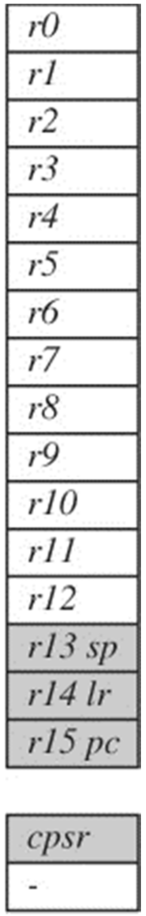
ARM Core Dataflow Model

- ❒ The ALU (arithmetic logic unit) or MAC (multiply-accumulate unit) takes the register values *Rn* and *Rm* from the A and B buses and computes a result
  - Data processing unit write the result in *Rd* directly to the register file
  - <u>Load and store instructions use the ALU to generate an address</u> to be held in the address register and broadcast on the Address bus
- ❒ For load and store instructions the incrementer updates the address register before the core reads or writes the next register value from or to the next sequential memory location

Registers and Current Program Status Register

- ❒ General purpose registers hold either data or an address

- ❒ The figure shows the active registers available in user mode
  - All the registers shown are 32 bits in size
  - 16 data registers + 2 processor status registers
  - Three registers, r13, r14, and r15, are assigned to a particular task or special function (the shaded registers)



Special Purpose Registers

- Register r13 is traditionally used as the stack pointer (sp) and stores the head of the stack in the current processor mode

- Register r14 is called the link register (lr) and is where the core puts the return address whenever it calls a subroutine

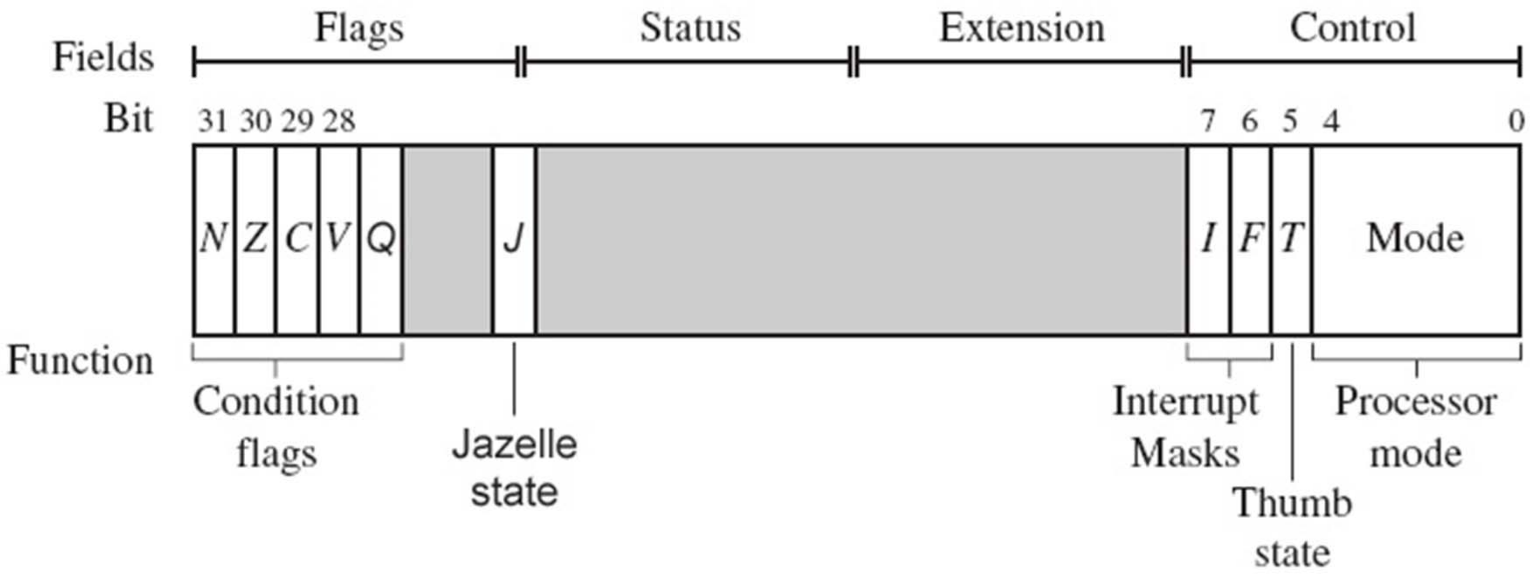
- Register r15 is the program counter (pc) and contains the address of the next instruction to be fetched by the processor

Special Purpose Registers

- Depending upon the context, registers r13 and r14 can also be used as general-purpose registers, which can be particularly useful since these registers are banked during a processor mode change
  - However, it is dangerous to use r13 as a general register when the processor is running ant form of operating system because operating systems often assume that r13 always points to a valid stack frame
- Registers r0 to r13 are orthogonal
  - Any instruction that you can apply to r0 you can equally well apply to any other registers
- There are instructions that treat r14 and r15 in a special way

Current Program Status Register

- The ARM core uses the cpsr to monitor and control internal operations
  - Divided into four fields: flags, status, extension, and control
  - In current designs the extension and status fields are reserved for future use



Current Program Status Register

- ❒ **The control field**
  - Processor mode
  - State
  - Interrupt mask bits

- ❒ **The flags field**
  - Condition flags

Processor Modes

- **The processor mode determines which registers are active and the access rights to the cpsr register itself**
  - **A privileged mode allows full read-write access to the cpsr**
  - **A nonprivileged mode only allows read access to the control field in the cpsr, but still allows full read-write access to the condition flags**

- **Seven processor modes** 최근에 + hypervioser 모드 (가상화 지원)
  - **Six privileged modes**
    - Abort, fast interrupt request, interrupt request, supervisor, system, and undefined
  - **One nonprivileged mode**
    - user

fulll virtualization - os를 전혀 고치지 않고 virtual 지원

linux os windows os
hypervisor
h/w

-> os가 직접 h/w에 어세스 할 수 없게 중계 역활을 한다 . 예를 들어, os가 실행할 명령어를 미리 읽어서 만약 in,out 명령어를 software interrupt로 바꿔서 supervisor가 스스로 decision을 내려서 h/w 어세스를 제어 한다. binary tranlation why? os가 hypervisor을 거치지 않고 h/w에 거치지 않게 하기 위해서
paravirtualization - os를 고쳐서 h/w access를 hyper call로 소스 코드를 변경시킨다.

arm의 경우 user
svc
hpyervisor <- 모드를 추가해서 h/w access와 같은 명령어를 실행 가능하게 한다. 기존 svc에서는 h/w access 불가능

Processor Modes

- **Abort mode** 불법적인 메모리 접근시 abort exception이 호출되며, abort mode로 바뀐다.
  - **When there is a failed attempt to access memory**
- **Fast interrupt request and interrupt request modes**
  - **Correspond to the two interrupt levels**

reset, software interrupt가 걸리면 supervisor mode로 변경된다.
bancked regitser 정의
lr 저장

- **Supervisor mode** 처음 실행시 reset exception이 발생하여, supervisor mode로 변경 된다.
  - **The processor is in after reset (when power is applied) and is generally the mode that an operating system kernel operates in**

system mode의 경우 cpsr의 모드를 변경해야만 갈 수 있다.
bancked register 정의 x
lr 저장 x

- **System mode**
  - **Special version of user mode that allows full read-write access to the cpsr**
- **Undefined mode** 1. 오류로 인해 명령어가 변경됨.(컴파일, 메모리 오류 등등) 2. thumb 모드가 아닐때 (16비트) thubm명령어 실행 했을 때 (모드와 명령어 불일치)
  - **When the processor encounters an instruction that is undefined or not supported by the implementation**
- **User mode**
  - **Used for programs and applications**
- **Hyp mode (ARMv8)** 기존 서버 시장에서 virtual 지원 , 현재는 자동차,스마트폰 등 가상화 지원 예상 자동차의 경우 기존 여러개의 흩어진 ecu들을 한개의 ecu로 합쳐서 여러가지의 os(기존 ecu에 돌아가던 os들)를 한곳에서 돌리게 한다. why? 기존 ecu들을 연결하기 위해서 전선이 너무 나도 많이 필요하기 때문에
  - **A hypervisor mode introduced in armv-7a for cortex-A15 processor for providing hardware virtualization support**

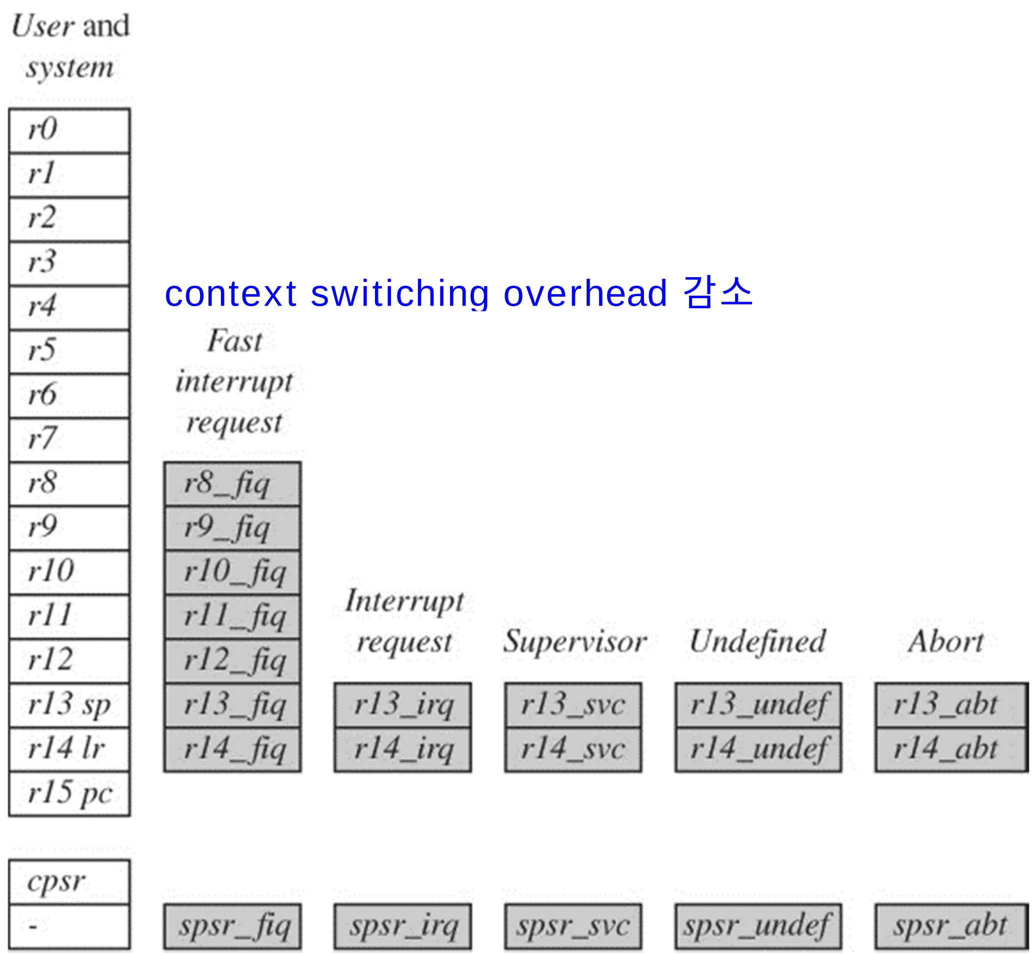
Processor Modes

Mode	Abbreviation	Privileged	Bits [4:0]
Abort	abt	Yes	10111
Fast Interrupt	fiq	Yes	10001
Interrupt request	irq	Yes	10010
Supervisor	svc	Yes	10011
System	sys	Yes	11111
Undefined	und	Yes	11011
User	usr	No	10000
Hyp (ARMv8)	hyp	Yes	11010

Banked Registers

- There are 37 registers in the register file
  - 20 registers are hidden from a program at different times
  - These registers are called banked registers and are identified by the shading in the program

context switiching overhead 감소



Banked Registers

- ❒ <u>Banked registers are available only when the processor is in a particular mode</u>
  - Abort mode has banked registers r13_abt, r14_abt, and spsr_abt

- ❒ <u>Every processor mode except user mode can change mode by writing directly to the mode bits of the cpsr</u>

- ❒ <u>A banked register maps one-to-one onto a user mode register</u>
  - If you change processor mode, a banked register from the new mode will <u>replace an existing register</u>

Banked Registers

- When the processor is in the interrupt request mode, the instructions still access registers named r13 and r14
  - However, these registers are the banked registers r13_irq and r14_irq
  - The user mode registers r13 and r14 are not affected by the instruction referencing these registers
  - A program still has normal access to the other registers r0 to r12

Mode Change

- Two ways of mode change
  - By a program that writes directly to the cpsr
  - By hardware when the core responds to an exception or interrupt
- The following exceptions and interrupts cause a mode change
  - Reset, interrupt request, fast interrupt request, software interrupt, data abort, prefetch abort, and undefined instruction
  - Exceptions and interrupts suspend the normal execution of sequential instructions and jump to a specific location

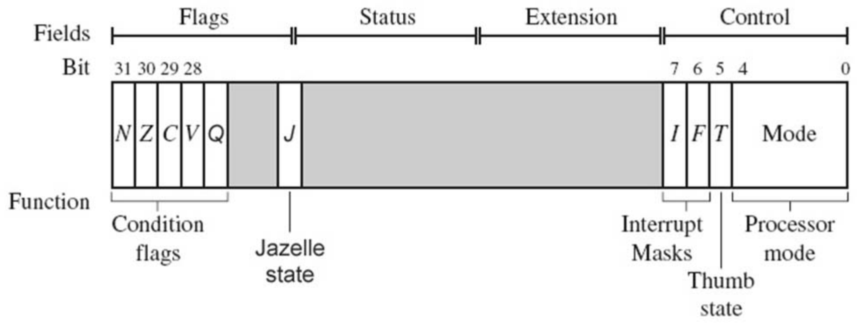
Mode Change from User to Interrupt Request

- The saved program status register (spsr) appears in interrupt request mode
  - The cpsr is copied into spsr_irq
  - To return back to the user mode, a special instruction is used that instructs the core to restore the original cpsr from the spsr_irq and bank in the user registers r13 and r14

- Note that the spsr can only be modified and read in a privileged mode
  - There is no spsr available in user mode
- Note that the cpsr is not copied into the spsr when a mode change forced due to a program writing directly to the cpsr

States and Instruction Sets

- The state of the core determines which instruction set is being executed (three instruction sets)
  - ARM: active in ARM state
  - Thumb: active in Thumb state
  - Jazelle: active in Jazelle state
- The jazelle J and Thumb T bits in the cpsr reflect the state of the processor
  - When both J and T bits are 0, the processor is in ARM state and executes ARM instructions



Jazelle DBX (Direct Bytecode eXecution): ARM Architecture Extensions for Java

- ARM has introduced a set of extensions to the ARM architecture that will allow an ARM processor to directly execute Java byte code alongside exiting operating systems, middleware and application code

- To execute Java bytecodes, you require the Jazelle technology plus a specially modified version of the Java virtual machine
  - It is important to note that the hardware portion of Jazelle only supports a subset of the Java bytecodes
  - The rest are emulated in software

Jazelle DBX (Direct Bytecode eXecution): ARM Architecture Extensions for Java

- There is an ARM instruction: <u>'BXJ Rm' for entering Java state</u>
  - This first performs a test on one of the condition codes
  - <u>If the condition is met, it then stores the current PC, puts the processor into Java state, branches to a target address specified in Rm and begins executing Java byte codes</u>

- <u>Interrupts are handled as normal, and cause an immediate return from Java state to ARM state to run the interrupt handler</u>
  - At the end of the interrupt routine, the normal return mechanism will return the processor to Java state

States and Instruction Set Features

	ARM	Thumb	Jazelle
Instruction Size	32-bit	16-bit	8-bit
Core instructions	58	30	Over 60% of Java : H/W The rest : S/W

cpsr	T=0
	J=0
		T=1
		J=0
			T=0, J=1

Jazelle DBX Implementation

- The Jazelle extension uses low-level binary translation
  - Implemented as an extra stage between the fetch and decode stages in the processor instruction pipeline
  - Recognized bytecodes are converted into a string of one or more native ARM instructions

- Java bytecode: instruction set of the JVM
  - Load and store (e.g. aload_0, istore)
  - Arithmetic and logic (e.g. ladd, fcmpl)
  - Type conversion (e.g. i2b, d2i)
  - Object creation and manipulation (new, putfield)
  - Operand stack management (e.g. swap, dup2)
  - Control transfer (e.g. ifeq, goto)
  - Method invocation and return (e.g. invokespecial, areturn)

Java code compilation

```
outer:
for (int i = 2; i < 1000; i++) {
    for (int j = 2; j < i; j++) {
        if (i % j == 0)
            continue outer;
    }
    System.out.println (i);
}
```

Java code

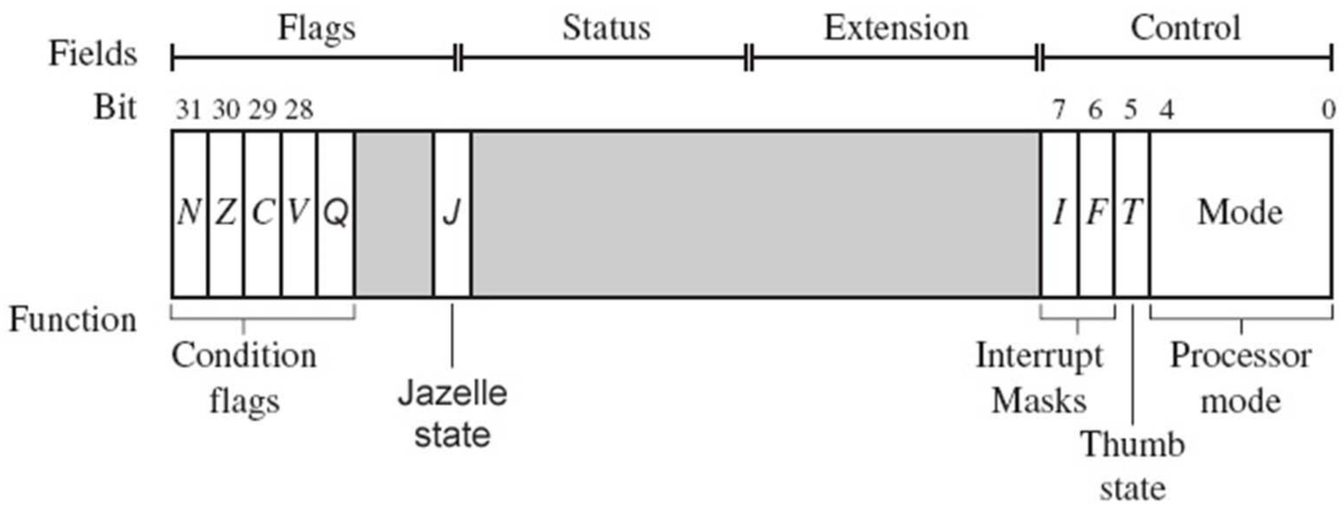
compile

```
0:   iconst_2
1:   istore_1
2:   iload_1
3:   sipush  1000
6:   if_icmpge      44
9:   iconst_2
10:  istore_2
11:  iload_2
12:  iload_1
13:  if_icmpge      31
16:  iload_1
17:  iload_2
18:  irem
19:  ifne    25
22:  goto    38
25:  iinc    2, 1
28:  goto    11
31:  getstatic
34:  iload_1
35:  invokevirtual
38:  iinc    1, 1
41:  goto    2
44:  return
```

Bytecode

Interrupt Masks

- Interrupt masks are used to stop specific interrupt requests from interrupting the processor
  - Two interrupt levels: interrupt request (IRQ) and fast interrupt request (FIQ)
  - The I bit in the cpsr masks IRQ when set to binary 1
  - The F bit in the cpsr masks FIQ when set to binary 1



Condition Flags

- Condition flags are updated by comparison and the result of ALU operations that specify the S instruction suffix
  - If a SUBS subtract instruction results in a register value of zero, then the Z flag in the cpsr is set

- Condition flags
  - N : Negative result from ALU
  - Z : Zero result from ALU
  - C : ALU operation Carried out unsigned overflow
  - V : ALU operation overflowed signed overflow
  - Q : Overflow & Saturation
    - ARMv5TEJ only

Conditional Execution

code density 증가
readability 안좋다. 매번 로직을 생각해야 하기 때문에

- Conditional execution controls whether or not the core will execute an instruction
  - The condition attribute is postfixed to the instruction mnemonic, which is encoded into the instruction
  - Prior to execution, the processor compares the condition attribute and with the condition flags in the cpsr
  - If they match, then the instruction is executed; otherwise the instruction is ignored

Conditional Execution

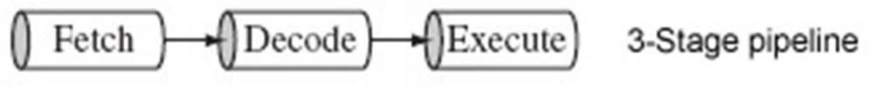
Mnemonic	Name	Condition flags
EQ	Equal	Z
NE	Not equal	z
CS/HS	Carry set / unsigned higher or same	C
CC/LO	Carry clear / unsigned lower	c
MI	Minus / negative	N
PL	Plus / positive or zero	n
VS	Overflow	V

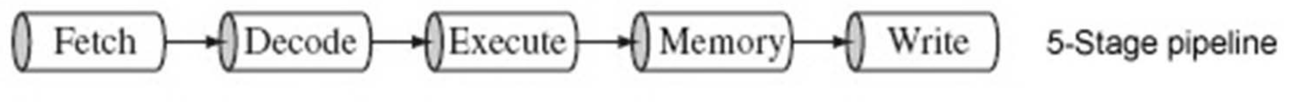
Conditional Execution

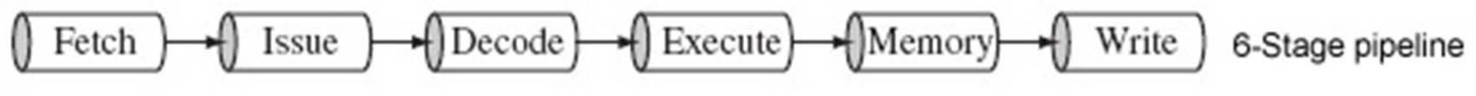
Mnemonic	Name	Condition flags
HI	Unsigned higher	zC
LS	Unsigned lower or same	Z or c
GE	Signed greater than or equal	NV or nv
LT	Signed less than	Nv or nV
GT	Signed greater than	NzV or nzv
LE	Signed less than or equal	Z or Nv or nV
AL	Always (unconditional)	Ignored

Pipeline

- The mechanism a RISC processor uses to execute instructions in parallel

ARM 7 

ARM 9 

ARM 10 

- As the pipeline length increases, the amount of work done at each stage is reduced, which allows the processor attain a higher operating frequency
  - This in turn increases the performance
  - This also increases the latency

Pipeline

- The ARM9 adds a memory and writeback stage
  - <u>1.1 Dhrystone MIPS per MHz</u>
  - Increase in instruction throughput by around <u>13%</u> compared with an ARM7

- The ARM10 adds an issue stage
  - <u>1.3 Dhrystone MIPS per MHz</u>
  - <u>34%</u> more throughput than an ARM7

- ARM9 and ARM10 use the same pipeline executing characteristics as an ARM7
  - <u>Code written for the ARM7 will execute on an ARM9 or ARM10</u>

Exceptions, Interrupts, and the Vector Table

- When an exception or interrupt occurs, the processor sets the pc to a specific memory address
  - The address is within a special address range called the vector table
  - The entries in the vector table are instructions that branch to specific routines designed to handle a particular exception or interrupt
- The memory map address 0x00000000 is reserved for the vector table, a set of 32-bit words
  - On some processors the vector table can be optionally located at a higher address in memory (starting at the offset 0xffff0000)
  - Operating systems such as Linux and MS's embedded products can take advantage of this feature

Exception Vectors

- ❒ Reset: When power is applied
- ❒ Undefined instruction: When the processor cannot decode an instruction
- ❒ Software interrupt: When the processor meet an SWI instruction
- ❒ Prefetch abort: When the processor attempts to fetch an instruction from an address without the correct access permission
- ❒ Data abort: When an instruction attempts to access data memory without the correct access permissions
- ❒ Interrupt request (IRQ): When an external hardware interrupts the normal execution flow of the processor
- ❒ Fast interrupt request (FIQ): When an hardware requiring faster response times interrupts the normal execution flow of the processor

Exception Vector Table

reset handler
~ boot loader <-

Exception	Shorthand	Vector address	High address
Reset	RESET	0x00000000	0xffff0000
Undefined instruction	UNDEF	0x00000004	0xffff0004
Software interrupt	SWI	0x00000008	0xffff0008
Prefetch abort	PABT	0x0000000c	0xffff000c
Data abort	DABT	0x00000010	0xffff0010
Reserved	-	0x00000014	0xffff0014
Interrupt request	IRQ	0x00000018	0xffff0018
Fast interrupt request	FIQ	0x0000001c	0xffff001c

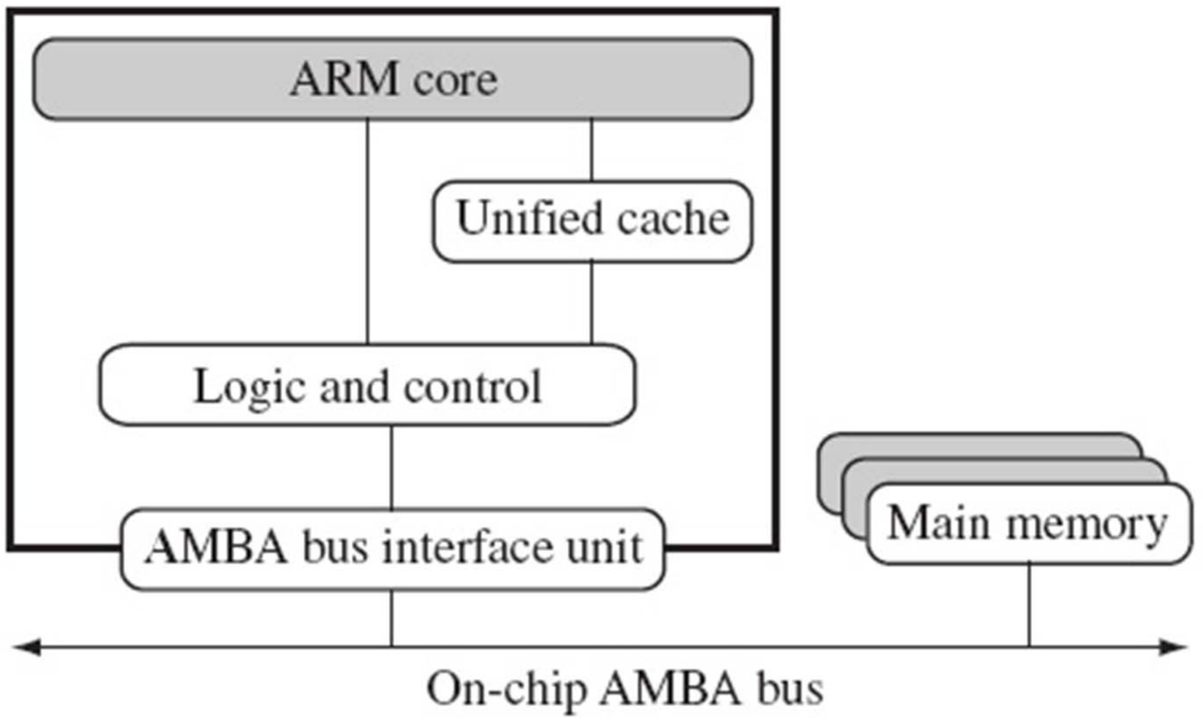
Core Extensions

- ❒ There are some hardware extensions that are standard components <u>placed next to the ARM core</u>
  - <u>Cache and tightly coupled memory</u>
  - <u>Memory management unit</u>
  - <u>Coprocessors</u>

Cache and Tightly Coupled Memory

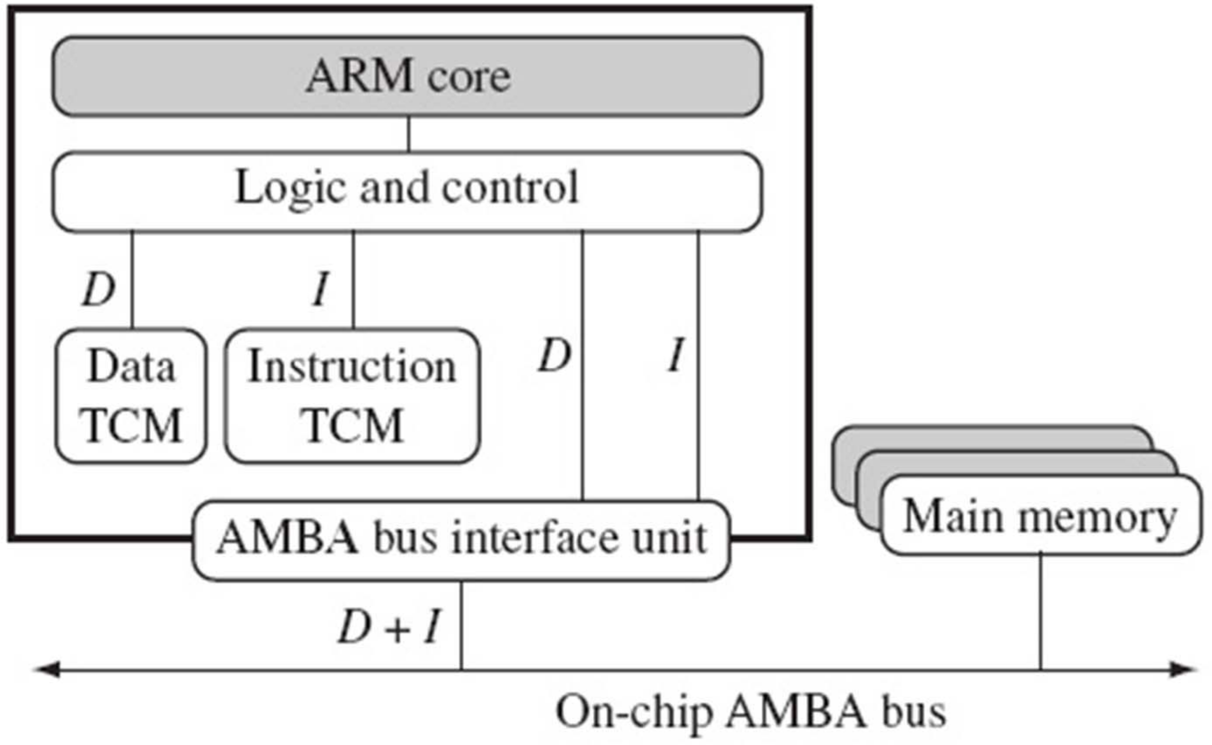
- **The cache is a block of memory placed between main memory and the core**
  - **With a cache the processor core can run for the majority of the time without having to wait for data from slow external memory**
  - **Most ARM-based embedded systems use a single-level cache internal to the processor**
- **ARM has two forms of cache**
  - **The first is found attached to the Von Neumann-style (Princeton) cores**
    - It combines both data and instruction into a single unified cache
  - **The second is attached to the Harvard-style cores**
    - It has separate caches for data and instruction

A Simplified Von Neumann Architecture with Cache

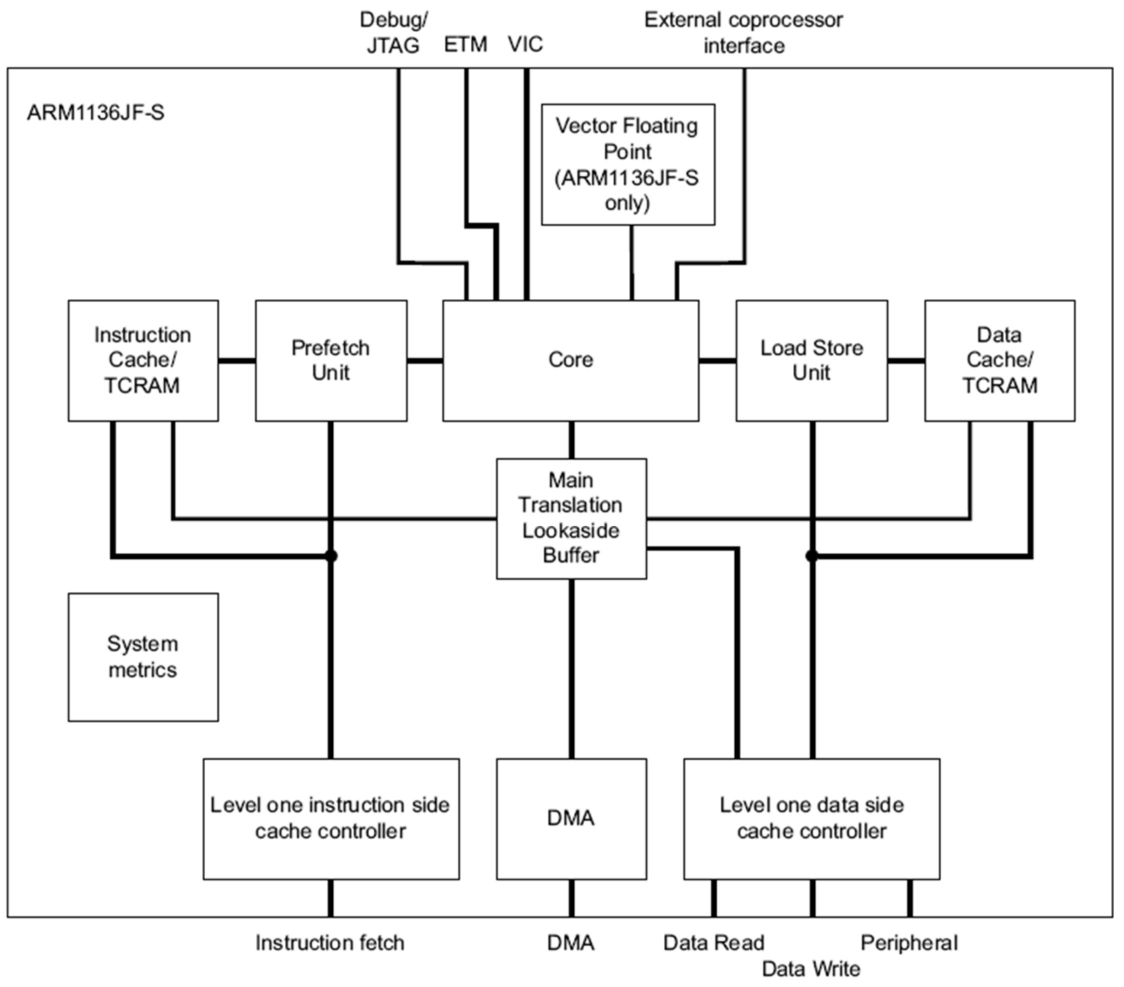


- The logic and control is the glue logic that connects the memory system to the AMBA bus

A Simplified Harvard Architecture with TCM



ARM1136JF-S Processor Block Diagram



Debug/ JTAG
ETM
VIC
External coprocessor interface
ARM1136JF-S
Vector Floating Point (ARM1136JF-S only)
Instruction Cache/ TCRAM
Prefetch Unit
Core
Load Store Unit
Data Cache/ TCRAM
Main Translation Lookaside Buffer
System metrics
Level one instruction side cache controller
DMA
Level one data side cache controller
Instruction fetch
DMA
Data Read
Data Write
Peripheral

Figure 1-1 ARM1136JF-S processor block diagram

Tightly Coupled Memory (TCM) ~ scratch pad memory

- ❒ A cache provides <u>an overall increase in performance</u> but <u>at the expense of predictable execution</u>

cache와 TCM 차이 cache - 프로그래머한테 안보임(architecture가 알아서 해줌)
TCM - 프로그래머한테 visible (메모리중 일부를 sram으로 처리) ex) interrupt handler code -> determistic 보장해준다.

- ❒ But for real-time systems, it is paramount that code execution is deterministic
  - <u>The time taken for loading and storing instructions and data must be predictable</u>
  - This is achieved using a form of memory called TCM
  - <u>TCM is fast SRAM located close to the core and guarantees the clock cycles required to fetch instructions or data</u>
  - <u>TCMs appear as memory in the address map</u> and can be accessed as fast memory

Memory Management Unit

- **Three types of memory management hardware**
  - **Non-protected memory**
    - Small embedded systems that require no protection from rouge application
  - **MPU (Memory Protection Unit)**
    - Simple systems that uses a limited number of memory regions
    - The memory regions are controlled with a set of coprocessor registers, and each region is defined with specific access permissions
  - **MMU (Memory Management Unit)**
    - Uses a set of translation tables to support a virtual-to-physical address map
    - More sophisticated platform operating systems that support multitasking

Coprocessors

- A coprocessor extends the processing features of a core by <u>extending the instruction set</u> or by <u>providing configuration registers</u>
  - More than one coprocessors can be added to the ARM core via the coprocessor interface

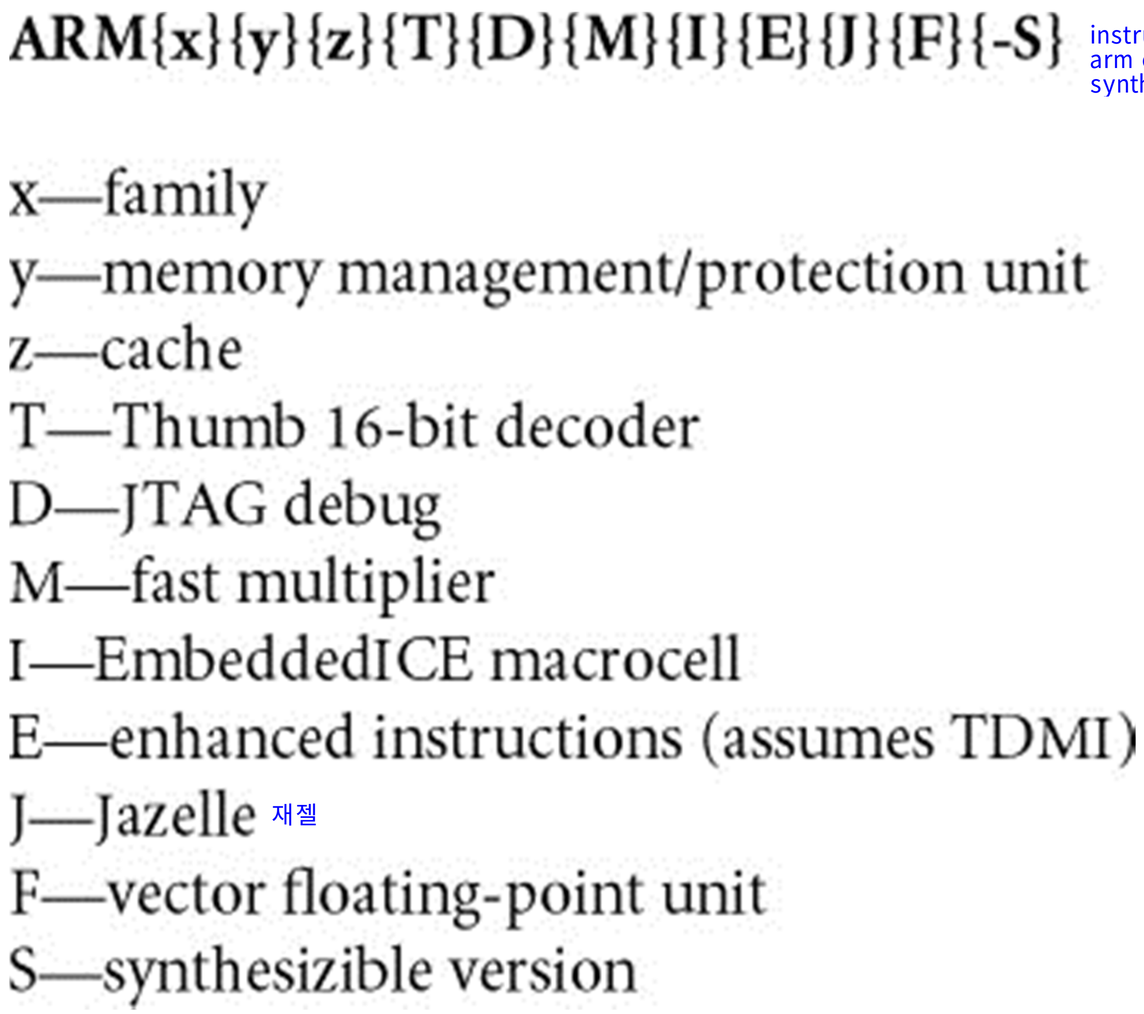
Coprocessors

- The coprocessor can extend the instruction set by providing a specialized group of new instructions
  - Vector floating-point (VFP) operations can be added
  - These new instructions are processed in the decode stage
  - If the decode stage sees a coprocessor instruction, then it offers it to the relevant coprocessor
  - But if the coprocessor is not present or doesn't recognize the instruction, the ARM takes an undefined instruction exception
- The coprocessor can also be accessed through configuration registers
  - Coprocessor 15 registers can be used to control cache, TCMs, and memory management

Architecture Revisions and Families

- The ISA has evolved to keep up with the demands of the embedded market
  - This evolution has been carefully managed by ARM, so that code written to execute on an earlier architecture will also execute on a later revision of the architecture

Nomenclature



ARM{x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{-S}

instruction set licencing
arm core 내부 설계 licencing
synthesizible - netlist까지 제공

x—family
y—memory management/protection unit
z—cache
T—Thumb 16-bit decoder
D—JTAG debug
M—fast multiplier
I—EmbeddedICE macrocell
E—enhanced instructions (assumes TDMI)
J—Jazelle 재젤
F—vector floating-point unit
S—synthesizible version

Nomenclature

- All ARM cores after the ARM7TDMI include the TDMI features even though they may not include those letters

- The processor family is a group of processor implementations that share the same characteristics
  - The ARM7TDMI, ARM740T, and ARM720T all share the same family and belong to the ARM7 family

- JTAG is described by IEEE 1149.1 Standard Test Access Port and boundary scan architecture
  - It is a serial protocol used by ARM to send and receive debug information between the core and test equipment

Nomenclature

- <u>EmbeddedICE macrocell is the debug hardware built into the processor that allows breakpoints and watchpoints to be set</u>

- <u>Synthesizable means that the processor core is supplied as source code</u> that can be compiled into a form easily used by EDA tools
  - Also known as soft cores that are delivered in a <u>HDL or gate netlist</u>
  - Can be used as building blocks within ASIC chip design or FPGA logic designs
  - Soft cores follow the SPR design flow (synthesis, placement, and route)

Architecture Evolution

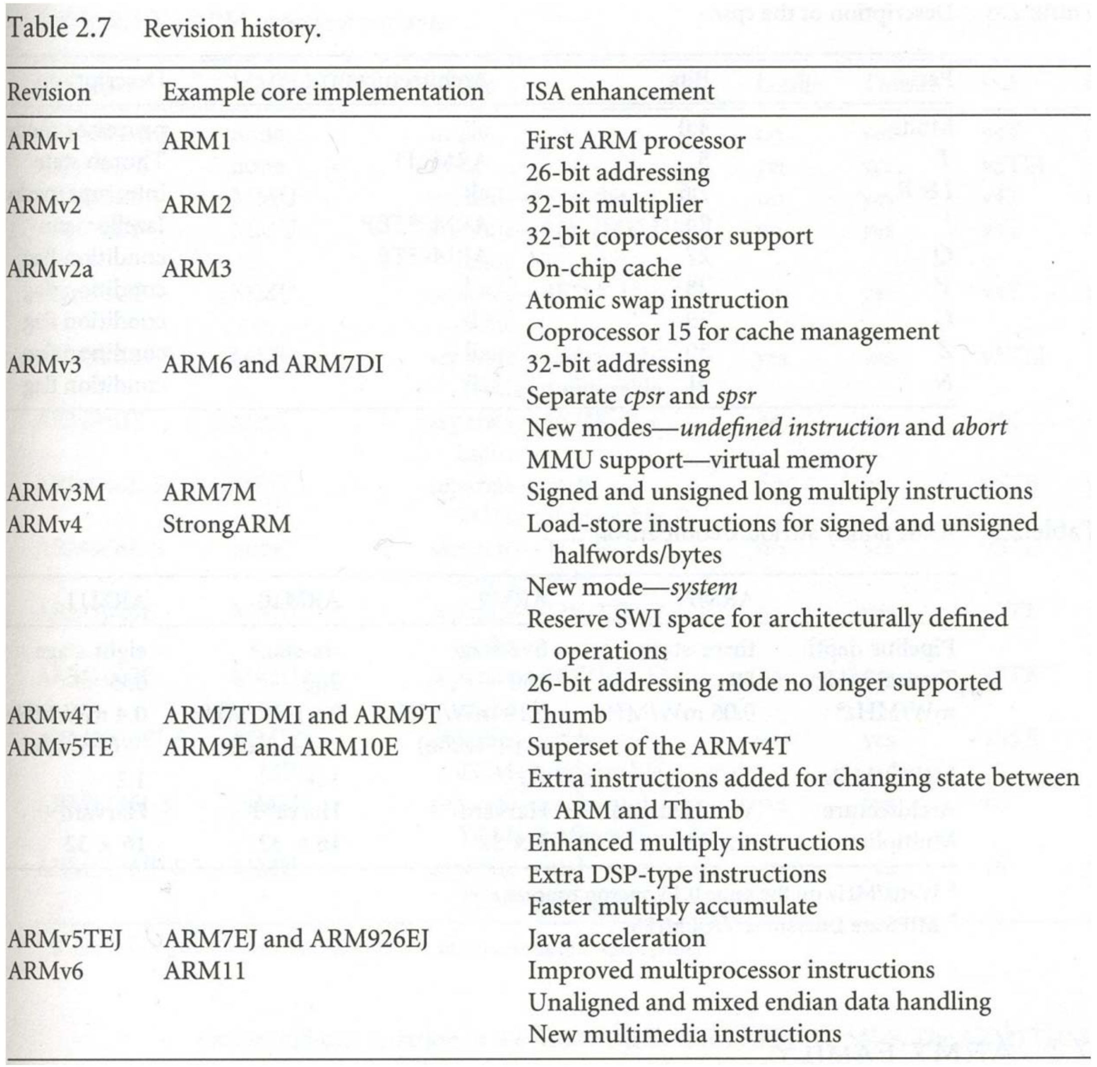
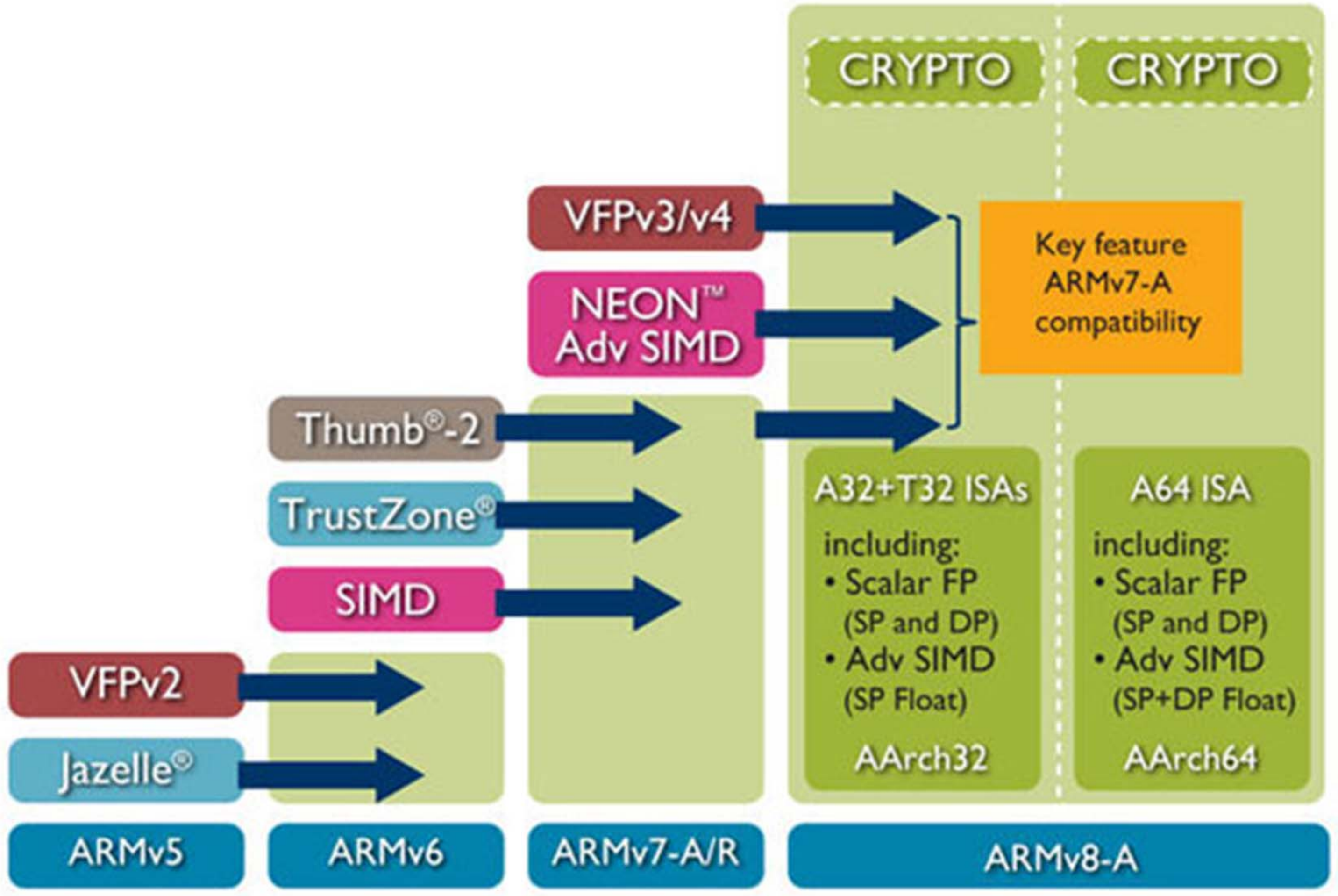


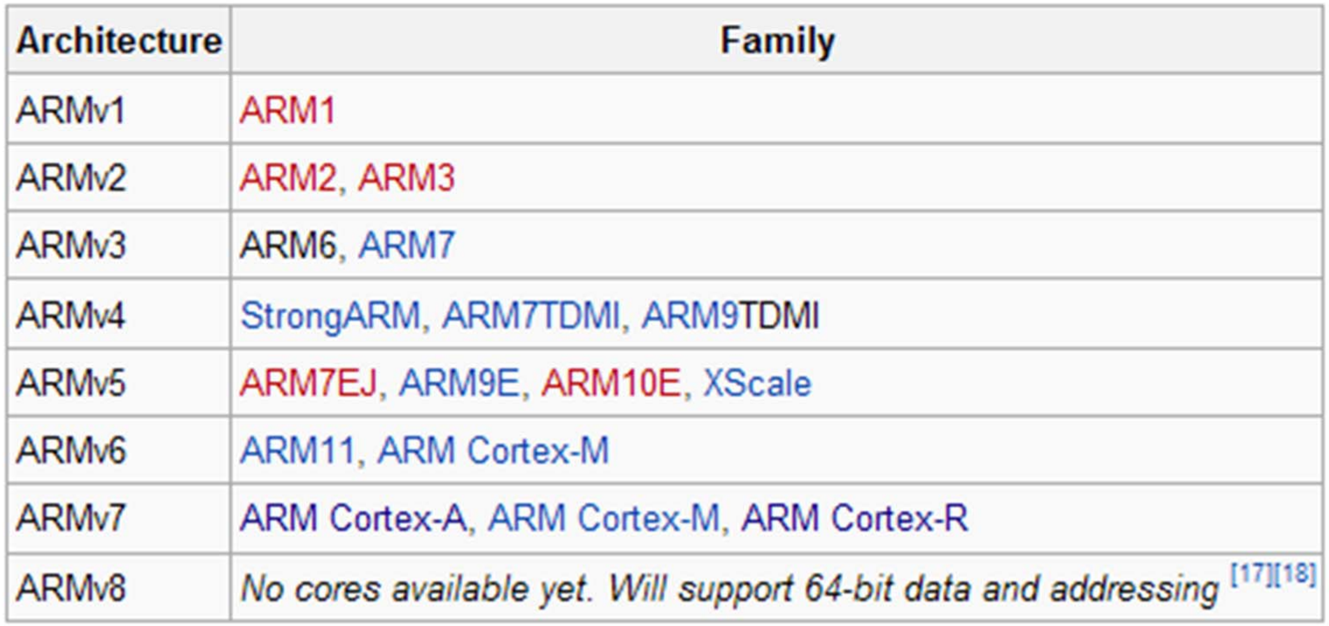
Table 2.7 Revision history.

Revision	Example core implementation	ISA enhancement
ARMv1	ARM1	First ARM processor 26-bit addressing
ARMv2	ARM2	32-bit multiplier 32-bit coprocessor support
ARMv2a	ARM3	On-chip cache Atomic swap instruction Coprocessor 15 for cache management
ARMv3	ARM6 and ARM7DI	32-bit addressing Separate *cpsr* and *spsr* New modes—*undefined instruction* and *abort* MMU support—virtual memory
ARMv3M	ARM7M	Signed and unsigned long multiply instructions
ARMv4	StrongARM	Load-store instructions for signed and unsigned halfwords/bytes New mode—*system* Reserve SWI space for architecturally defined operations 26-bit addressing mode no longer supported
ARMv4T	ARM7TDMI and ARM9T	Thumb
ARMv5TE	ARM9E and ARM10E	Superset of the ARMv4T Extra instructions added for changing state between ARM and Thumb Enhanced multiply instructions Extra DSP-type instructions Faster multiply accumulate
ARMv5TEJ	ARM7EJ and ARM926EJ	Java acceleration
ARMv6	ARM11	Improved multiprocessor instructions Unaligned and mixed endian data handling New multimedia instructions

ARMv5 to ARMv8


CRYPTO
CRYPTO
VFPv3/v4
NEON™ Adv SIMD
Key feature ARMv7-A compatibility
Thumb®-2
TrustZone®
SIMD
VFPv2
Jazelle®
A32+T32 ISAs including: • Scalar FP (SP and DP) • Adv SIMD (SP Float)
AArch32
A64 ISA including: • Scalar FP (SP and DP) • Adv SIMD (SP+DP Float)
AArch64
ARMv5
ARMv6
ARMv7-A/R
ARMv8-A

ARM Architecture and Family



Architecture	Family
ARMv1	ARM1
ARMv2	ARM2, ARM3
ARMv3	ARM6, ARM7
ARMv4	StrongARM, ARM7TDMI, ARM9TDMI
ARMv5	ARM7EJ, ARM9E, ARM10E, XScale
ARMv6	ARM11, ARM Cortex-M
ARMv7	ARM Cortex-A, ARM Cortex-M, ARM Cortex-R
ARMv8	*No cores available yet. Will support 64-bit data and addressing* [17][18]

ARM Processor Families

❒ ARM families

- ARM7, ARM9, ARM10, ARM11, and Coretex cores
- The postfix numbers indicate different core designs
- ARM8 was developed but was soon superseded

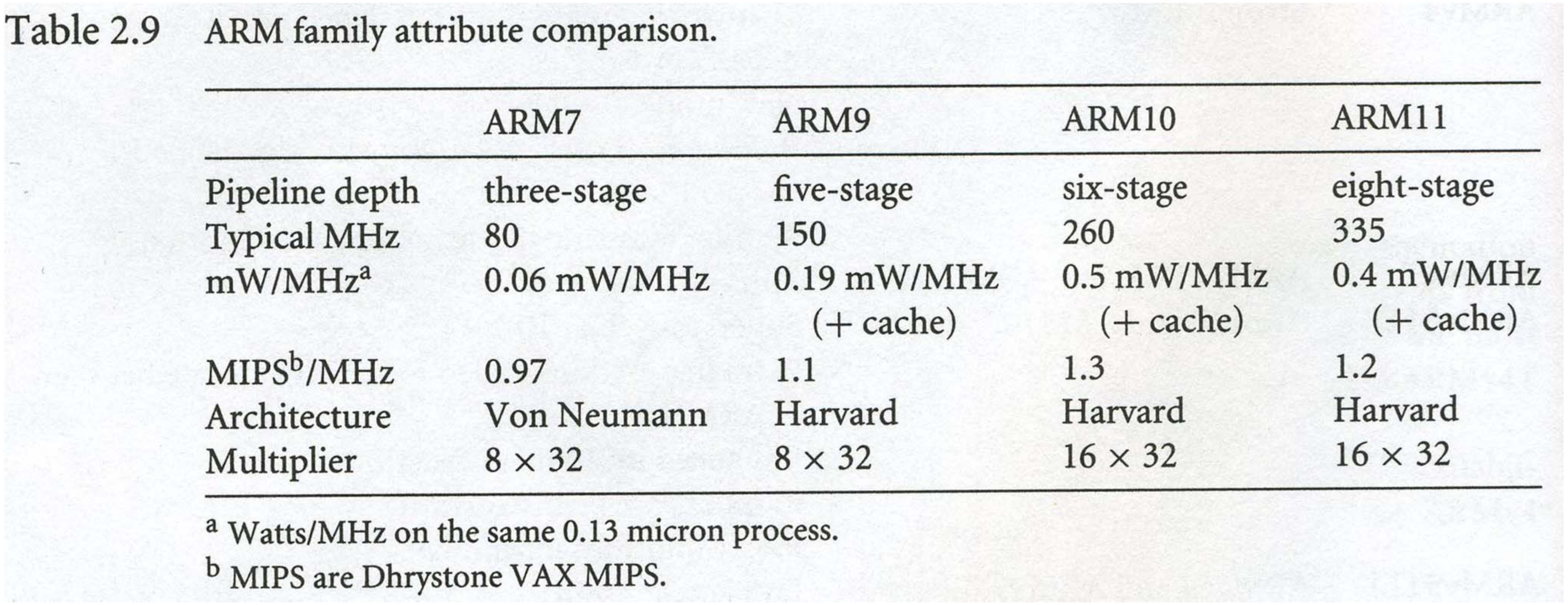


Table 2.9 ARM family attribute comparison.

	ARM7	ARM9	ARM10	ARM11
Pipeline depth	three-stage	five-stage	six-stage	eight-stage
Typical MHz	80	150	260	335
mW/MHz[a]	0.06 mW/MHz	0.19 mW/MHz (+ cache)	0.5 mW/MHz (+ cache)	0.4 mW/MHz (+ cache)
MIPS[b]/MHz	0.97	1.1	1.3	1.2
Architecture	Von Neumann	Harvard	Harvard	Harvard
Multiplier	8 × 32	8 × 32	16 × 32	16 × 32

[a] Watts/MHz on the same 0.13 micron process.

[b] MIPS are Dhrystone VAX MIPS.

ARM Processor Variants

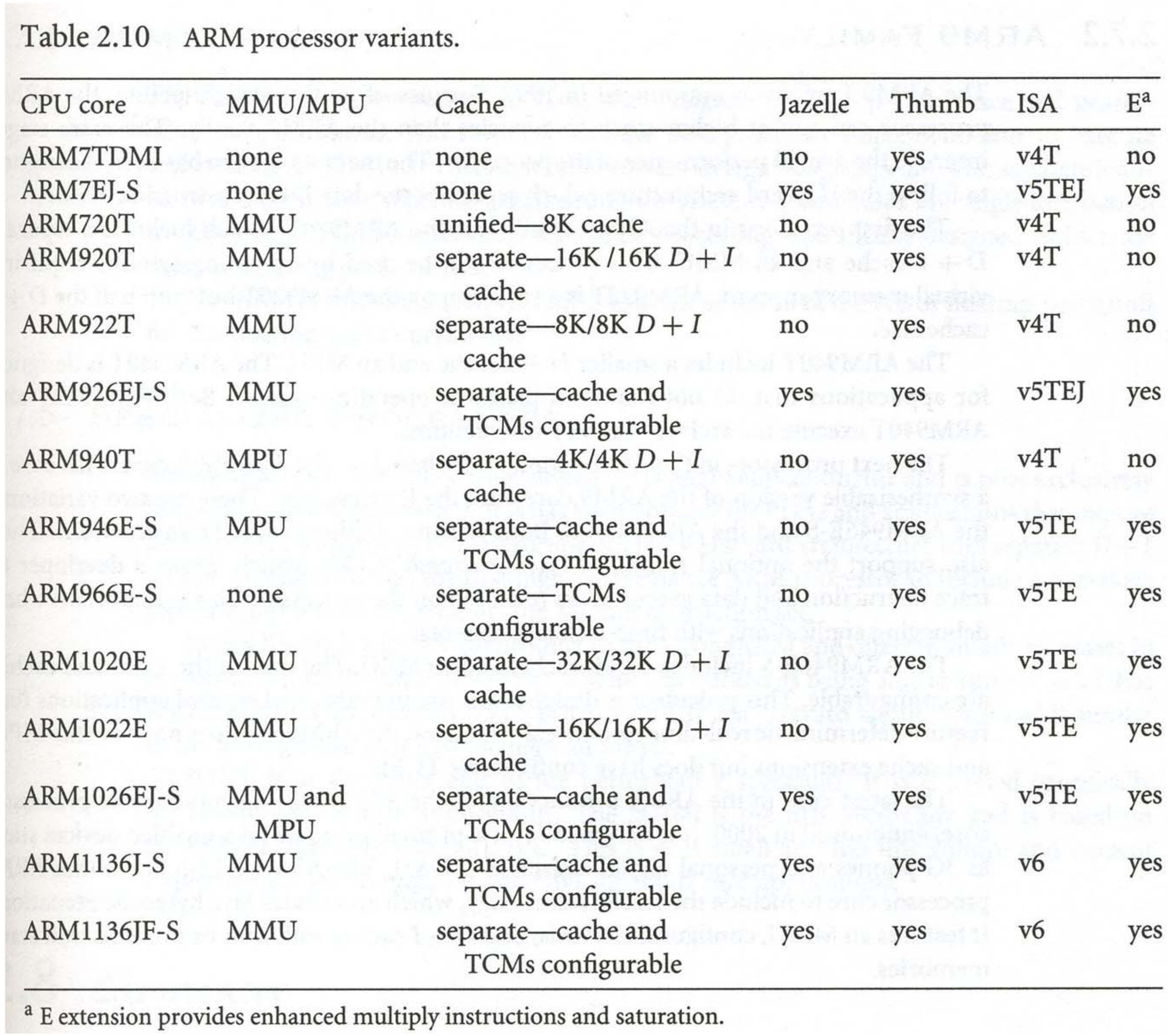


Table 2.10 ARM processor variants.

CPU core	MMU/MPU	Cache	Jazelle	Thumb	ISA	E[a]
ARM7TDMI	none	none	no	yes	v4T	no
ARM7EJ-S	none	none	yes	yes	v5TEJ	yes
ARM720T	MMU	unified—8K cache	no	yes	v4T	no
ARM920T	MMU	separate—16K /16K *D* + *I* cache	no	yes	v4T	no
ARM922T	MMU	separate—8K/8K *D* + *I* cache	no	yes	v4T	no
ARM926EJ-S	MMU	separate—cache and TCMs configurable	yes	yes	v5TEJ	yes
ARM940T	MPU	separate—4K/4K *D* + *I* cache	no	yes	v4T	no
ARM946E-S	MPU	separate—cache and TCMs configurable	no	yes	v5TE	yes
ARM966E-S	none	separate—TCMs configurable	no	yes	v5TE	yes
ARM1020E	MMU	separate—32K/32K *D* + *I* cache	no	yes	v5TE	yes
ARM1022E	MMU	separate—16K/16K *D* + *I* cache	no	yes	v5TE	yes
ARM1026EJ-S	MMU and MPU	separate—cache and TCMs configurable	yes	yes	v5TE	yes
ARM1136J-S	MMU	separate—cache and TCMs configurable	yes	yes	v6	yes
ARM1136JF-S	MMU	separate—cache and TCMs configurable	yes	yes	v6	yes

[a] E extension provides enhanced multiply instructions and saturation.

ARM7 Family

- The ARM7TDMI was the first of a new range of processors introduced in 1995
  - Licensed by many of the top semiconductor companies around the world

- Characteristics
  - <u>Good performance-to-power ratio</u>
  - <u>The first core that introduced the Thumb instruction set</u>

ARM9 Family

- ❒ ARM9 family was announced in 1997
  - ▪ The memory system has been redesigned to follow the Harvard architecture which separates the data D and instruction I buses
- ❒ ARM920T was the first processor in the ARM9 family
  - ▪ A separate 16K/16K D + I cache and an MMU
- ❒ ARM946E-S and ARM966E-S execute v5TE instructions and support ETM (embedded trace macrocell)
- ❒ ARM926EJ-S was designed for small portable Java-enabled devices such as 3G phones
  - ▪ The first to include the Jazelle technology

ARM10 Family

- The ARM10 announced in 1999 was designed for performance
  - 6-stage pipeline and optional VFP (vector floating point) unit which adds a seventh stage to the ARM10 pipeline
  - VFP increases floating-point performance and is compliant with the IEEE 754.1985 floating-point standard

ARM11 Family

- The ARM1136J-S announced was designed for high performance and power efficient applications
  - The first processor implementation to execute architecture ARMv6 instructions
  - 8-stage pipeline with separate load-store and arithmetic pipelines
  - Single instruction multiple data (SIMD) extensions for media processing, specifically designed to increase video processing performance

Cortex Series

- **Three "profiles" are defined**
  - **"Application" profile: Cortex-A series** 핸드폰
    - Provide an entire range of solutions for devices hosting a rich OS platform and user applications
  - **"Real-time" profile: Cortex-R series** 자동차
    - Designed for high performance, dependability and error-resistance with highly deterministic behavior
  - **"Microcontroller" profile: Cortex-M series** iot
    - Optimized for cost and power sensitive MCU and mixed-signal devices
- **Profiles are allowed to subset the architecture**
  - **For example, the ARMv6-M profile (used by the Cortex-M0) is a subset of the ARMv7-M profile (it supports fewer instructions)**

Specialized Processors

- StrongARM was originally co-developed by Digital Semiconductor
  - Now exclusively licensed by Intel Corporation
  - Popular for PDAs
  - Harvard architecture with separate D + I caches
  - 5-stage pipeline
  - No support for the Thumb instructions set

Specialized Processors

- ❒ Intel’s Xscale is a follow-on product to the StrongARM
  - Dramatic increase in performance
  - Runs up to 1 GHz
  - Harvard architecture and MMU

- ❒ SC100 is at the other end of the performance spectrum
  - Designed specifically for low-power security applications
  - The SC100 is the first SecureCore and is based on an ARM7TDMI with an MPU
  - Small and has low voltage and current requirements
  - Attractive for smart card applications

Thumb Instruction Set

bit 제한 때문에 프로그램 제한 많다.`

- **A compact 16-bit encoding for a subset of the ARM instruction set**
  - **The purpose is to improve compiled code-density**

- **Processors since the ARM7TDMI have featured Thumb instruction set, which have their own state**
  - **The "T" in "TDMI" indicates the Thumb feature**

- **The space-saving comes from making some of the instruction operands implicit and limiting the number of possibilities compared to the ARM instructions executed in the ARM instruction set state**

Thumb 2 Instruction Set

❒ Thumb-2 technology made its debut in the ARM1156 core, announced in 2003
  - Thumb-2 extends the limited 16-bit instruction set of Thumb with additional 32-bit instructions to give the instruction set more breadth, thus producing a variable-length instruction set
  - A stated aim for Thumb-2 is to achieve code density similar to Thumb with performance similar to the ARM instruction set on 32-bit memory

❒ Thumb-2 extends both the ARM and Thumb instruction set with yet more instructions, including bit-field manipulation, table branches, and conditional execution

SIMD

- SIMD (Single Instruction, Multiple Data) is a technique employed to achieve data level parallelism, as in a vector or array processor

- Example: the same value is being added to a large number of data points
  - It would be changing the brightness of an image
  - Each pixel of an image consists of three values for the brightness of the red, green and blue portions of the color
  - To change the brightness, the R G and B values are read from memory, a value is added (or subtracted) from it, and the resulting value is written back out to memory

The ARM DSP Extensions and SIMD

- The ARM DSP instruction set extensions increase the DSP processing
  - Optimized for a broad range of software applications including servo motor control, Voice over IP (VOIP) and video & audio codecs

- Features
  - Single-cycle 16x16 and 32x16 MAC implementations
  - New instructions to load and store pairs of registers, with enhanced addressing modes
  - New CLZ instruction improves normalization in arithmetic operations and improves divide performance

NEON

❐ The Advanced SIMD extension

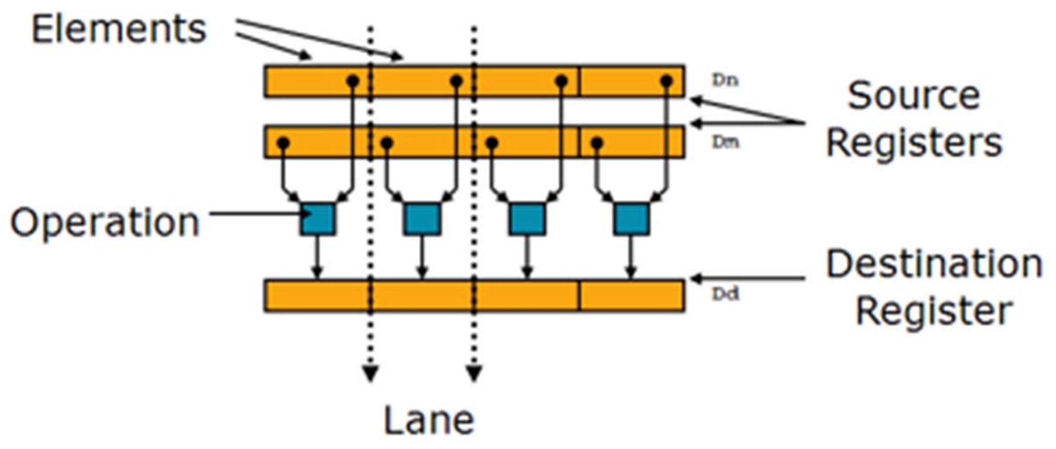
- A combined 64- and 128-bit single instruction multiple data (SIMD) instruction set that provides standardized acceleration for media and signal processing applications
- At least 3x the performance of ARMv5 and at least 2x the performance of ARMv6 SIMD

❐ NEON is included in all Cortex-A8 devices but is optional in Cortex-A9 devices

NEON

❒ NEON instructions perform "Packed SIMD" processing:

- Registers are considered as vectors of elements of the same data type
- Data types can be: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single precision floating point
- Instructions perform the same operation in all lanes

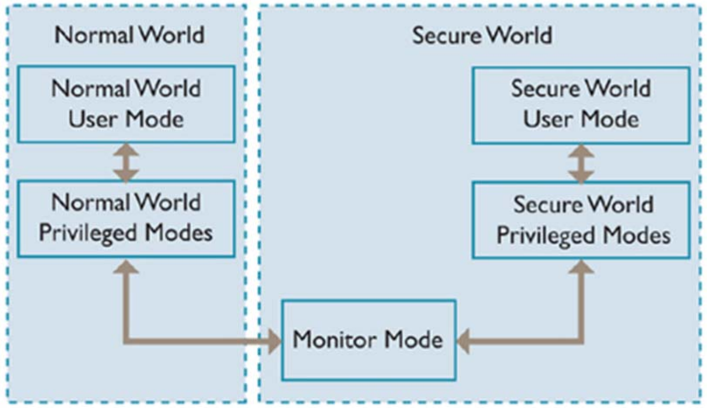


TrustZone

- The Security Extensions
  - Provide two virtual processors backed by hardware based access control
  - Enable the application core to switch between two states, referred to as worlds, in order to prevent information from leaking from the more trusted world to the less trusted world
  - Each world can operate independently of the other while using the same core
- Typical applications are to run a rich operating system in the less trusted world, and smaller security-specialized code in the more trusted world
  - The specific implementation details of TrustZone are proprietary and have not been publicly disclosed for review

TrustZone

- ❐ Modes in an ARM for the Security Extensions



- ❐ The entry to monitor can be triggered by software executing a dedicated Secure Monitor Call (SMC) instruction, or by a subset of the hardware exceptions
  - The IRQ, FIQ, external Data Abort, and external Prefetch Abort exceptions can all be configured to cause the processor to switch into monitor mode