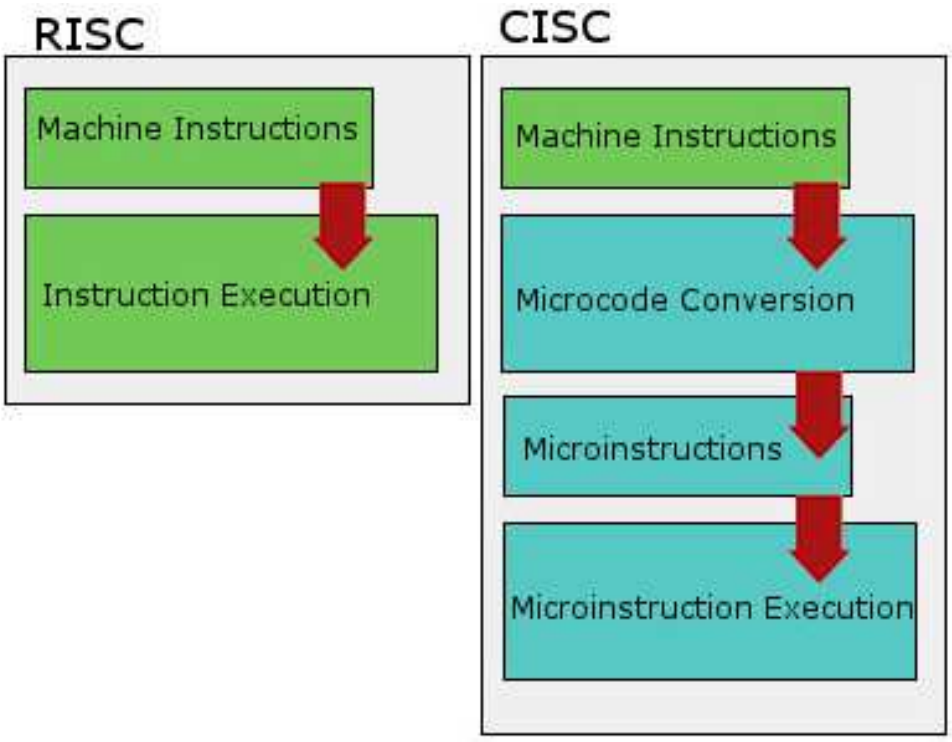
ARM Architecture and RISC Applications

Nikola Zlatanov*

ARM, originally Acorn RISC Machine, later Advanced RISC Machine, is a family of **Reduced Instruction Set Computing** (RISC) **architectures** for **computer processors**, configured for various environments. British company **ARM Holdings** develops the architecture and licenses it to other companies, who design their own products that implement one of those architectures—including **systems-on-chips** (SoC) that incorporate memory, interfaces, radios, etc. It also designs **cores** that implement this instruction set and licenses these designs to a number of companies that incorporate those core designs into their own products.

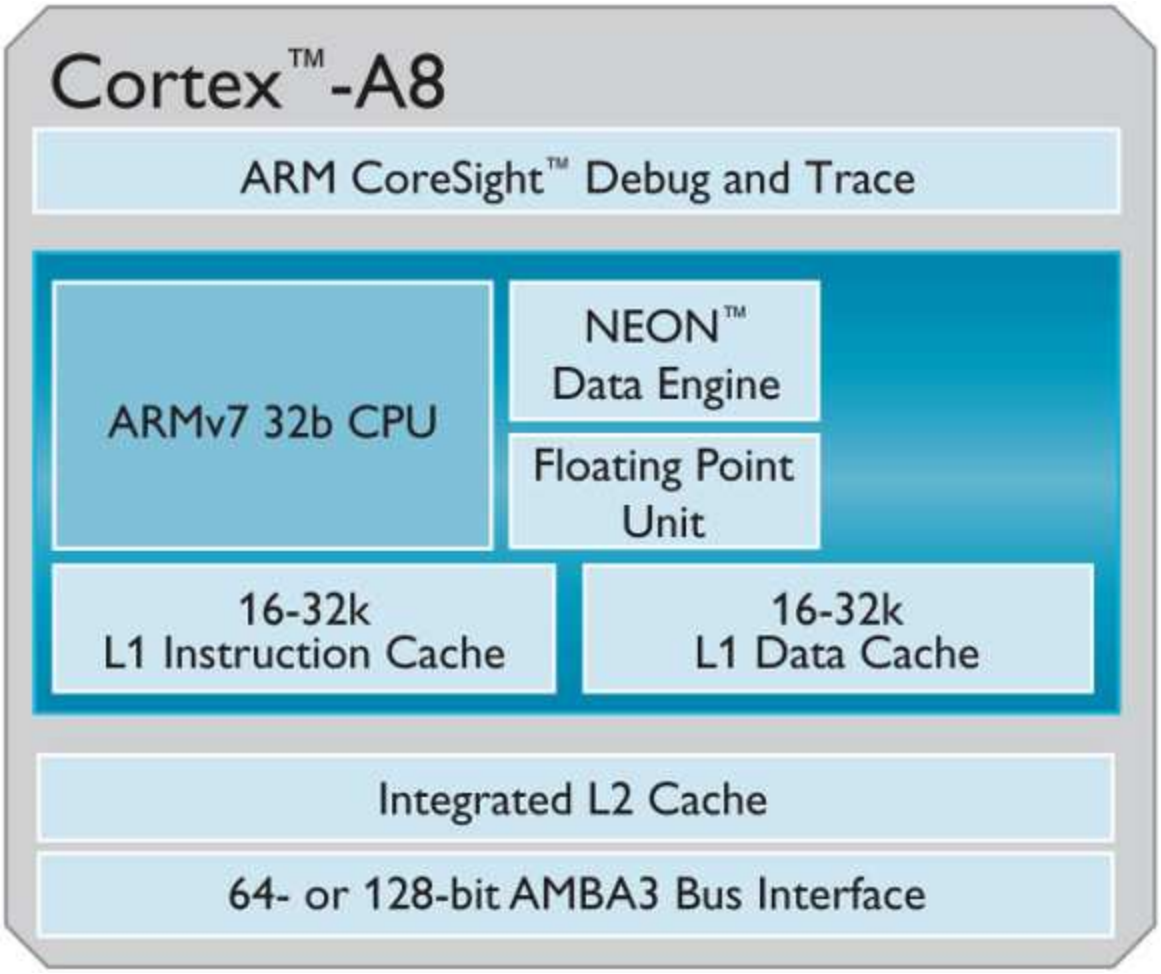
If you've paid any attention to smartphones and tablets you've likely heard of the term "ARM" used to refer to the hardware inside. It's thrown around left and right, often as a point of differentiation from laptops and desktops, which use Intel x86. The Key To ARM Is RISC.



RISC is, in its broadest form, a design philosophy for processors. It stems from a belief that a processor with a relatively simple instruction set will be more efficient than one which is more complex. The term originally came into use back in the 1980s with a research project called Berkeley RISC that investigated the possibilities of this approach to design and then created processors based on it.

All ARM processors are considered RISC designs, but this doesn't mean much because RISC itself is simply an approach to design rather than a technological standard or processor architecture. Still, a basic understanding of RISC properly frames ARM.

ARM's Basics



ARM refers to itself as an architecture, which can cause a misunderstanding when compared to Intel. Intel gives every new chip design its own unique code and talks about each as a new architecture – even when there's often many similarities and they all use the same instruction set (x86). ARM, on the other hand, treats its designs as an unbroken family. Updates are still a part of the ARM architecture. They're just given a new version number.

The trait that's most relevant to consumers is not the micro-architecture (the physical design of the chip) but instead the instruction set. The instruction set is the basic set of capabilities and features a processor makes available to software. It determines what arithmetic can be used, how cache should be allocated and the order in which instructions should be executed. Software designed for one instruction set can't be used on another unless it's revised.

Micro-architectures and instruction sets can't be separated because the architecture is a physical expression of the instruction set. This is why ARM-based processors tend to be small, efficient and relatively slow. The simple instruction set requires a small, simple design with fewer transistors. Transistors consume power and increase die size (which increases production cost), so having as few as possible is ideal when selecting a processor for a smartphone or tablet.

ARM's Business Is Different

Talking about ARM processors as a whole can be difficult because there are so many available and their performance varies. It's counter-intuitive. How can Apple have ARM processors that are quicker than the competition when it's using the same architecture?

This happens because of how ARM Holdings, the company that is responsible for ARM, does business. ARM Holdings is only a design company. They manage the instruction set and design new versions of the core architecture and then license it to other companies. Those companies can then improve it and pair it with whatever hardware seems appropriate.

It helps to understand that ARM's core architecture is only a processor. It doesn't handle wireless connectivity. It doesn't handle graphics. It doesn't handle USB or other forms of wired connectivity. All of that is the responsibility of other hardware licensees pair the architecture.

That's why there are so many variants of ARM on the market and why they perform differently. Apple has an entire in-house engineering staff that works on its ARM processors. Other companies, like Qualcomm and Texas Instruments, act as middle-men. They take the ARM architecture, pair it with a variety of hardware, and then re-sell it as a "system-on-a-chip" for smartphones and tablets.

What Does ARM Mean For Consumers

To a consumer, ARM can be thought of as an ecosystem. Software designed for ARM will only work on ARM. Windows RT apps, for example, don't work on a PC with Windows 8. Modifications must be made to a program to jump from ARM to x86.

Operating systems that work on one ARM device should work on others. This is why there are so many Android modifications and why Android can potentially be loaded onto tablets from HP and BlackBerry. Apple does mess with the ecosystem a bit, however, because the source code of iOS is not available. Attempting to port iOS to other ARM devices is almost impossible without it.

ARM also means lower power draw and lower performance relative to x86. This is not set in stone, however, because both architectures are changing over time. Intel is working hard to create versions of its processors with extremely low power draw. And ARM Holdings is working hard to improve the performance of its designs.

Will ARM Be In Your PC

There have been a few attempts to sell traditional devices with ARM processors that work like traditional PCs. Motorola sold a keyboard dock for the Atrix smartphone and advertised it as a laptop replacement. ASUS sells a line of Android tablets with keyboard docks. And Samsung now sells a Chromebook that runs on ARM.

Such attempts foreshadow a storm of potential surrounding ARM, yet predicting the storm's path and intensity is impossible. Current ARM architectures are significantly behind the performance Intel's slowest processors (never mind its mainstream line of Core processors). Nvidia says that it is working on a processor using ARM's architecture that will compete with Intel, but it's not clear how this is being accomplished or when a finished product might be released.

For now the potential threat of ARM remains a dark cloud on the horizon of the potential PC space. It looks threatening, but a concrete threat has yet to manifest. Is ARM a fearsome storm or simply a few shadowed clouds that will eventually disperse? That remains to be seen.

Acorn RISC Machine: ARM2

The official Acorn RISC Machine project started in October 1983. They chose **VLSI Technology** as the silicon partner, as they were a source of ROMs and custom chips for Acorn. Wilson and Furber led the design. They implemented it with a similar efficiency ethos as the 6502. A key design goal was achieving low-latency input/output (interrupt) handling like the 6502. The 6502's memory access architecture had let developers produce fast machines without costly **direct memory access** hardware.

The first samples of ARM silicon worked properly when first received and tested on 26 April 1985.

The first ARM application was as a second processor for the BBC Micro, where it helped in developing simulation software to finish development of the support chips (VIDC, IOC, MEMC), and sped

up the **CAD software** used in ARM2 development. Wilson subsequently rewrote BBC BASIC in ARM assembly language. The in-depth knowledge gained from designing the instruction set enabled the code to be very dense, making ARM BBC BASIC an extremely good test for any ARM emulator. The original aim of a principally ARM-based computer was achieved in 1987 with the release of the **Acorn Archimedes**. In 1992, Acorn once more won the **Queen's Award for Technology** for the ARM.

The ARM2 featured a 32-bit **data bus**, **26-bit** address space and 27 32-bit **registers**. Eight bits from the **program counter** register were available for other purposes; the top six bits (available because of the 26-bit address space) served as status flags, and the bottom two bits (available because the program counter was always **word-aligned**) were used for setting modes. The address bus was extended to 32 bits in the ARM6, but program code still had to lie within the first 64 MB of memory in 26-bit compatibility mode, due to the reserved bits for the status flags. The ARM2 had a **transistor count** of just 30,000, compared to Motorola's six-year-older 68000 model with around 40,000. Much of this simplicity came from the lack of **microcode** (which represents about one-quarter to one-third of the 68000) and from (like most CPUs of the day) not including any **cache**. This simplicity enabled low power consumption, yet better performance than the **Intel 80286**. A successor, ARM3, was produced with a 4 KB cache, which further improved performance.

The new Apple-ARM work would eventually evolve into the ARM6, first released in early 1992. Apple used the ARM6-based ARM610 as the basis for their Apple Newton PDA. In 1994, Acorn used the ARM610 as the main **central processing unit** (CPU) in their **RiscPC** computers. **DEC** licensed the ARM6 architecture and produced the StrongARM. At 233 **MHz**, this CPU drew only one watt (newer versions draw far less). This work was later passed to Intel as a part of a lawsuit settlement, and Intel took the opportunity to supplement their **i960** line with the StrongARM. Intel later developed its own high performance implementation named XScale, which it has since sold to **Marvell**. Transistor count of the ARM core remained essentially the same size throughout these changes; ARM2 had 30,000 transistors, while ARM6 grew only to 35,000.

Market share

In 2005, about 98% of all mobile phones sold used at least one ARM processor. In 2010, producers of chips based on ARM architectures reported shipments of 6.1 billion **ARM-based processors**, representing 95% of **smartphones**, 35% of **digital televisions** and **set-top boxes** and 10% of **mobile computers**. In 2011, the 32-bit ARM architecture was the most widely used architecture in mobile devices and the most popular 32-bit one in embedded systems. In 2013, 10 billion were produced and "ARM-based chips are found in nearly 60 percent of the world's mobile devices".

Core licence

ARM Holdings' primary business is selling **IP cores**, which licensees use to create **microcontrollers** (MCUs), **CPUs**, and **systems-on-chips** based on those cores. The **original design manufacturer** combines the ARM core with other parts to produce a complete device, typically one that can be built in existing **semiconductor fabs** at low cost and still deliver substantial performance. The most successful implementation has been the **ARM7TDMI** with hundreds of millions sold. **Atmel** has been a precursor design center in the ARM7TDMI-based embedded system.

The ARM architectures used in smartphones, PDAs and other **mobile devices** range from ARMv5 to ARMv6, used in low-end devices, to **ARMv7-A** used in current high-end devices. ARMv7 includes a hardware **floating-point unit** (FPU), with improved speed compared to software-based floating-point.

In 2009, some manufacturers introduced netbooks based on ARM architecture CPUs, in direct competition with netbooks based on **Intel Atom**. According to analyst firm IHS iSuppli, by 2015, ARM ICs may be in 23% of all laptops.

ARM Holdings offers a variety of licensing terms, varying in cost and deliverables. ARM Holdings provides to all licensees an integratable hardware description of the ARM core as well as complete

software development toolset (**compiler**, **debugger**, **software development kit**) and the right to sell manufactured **silicon** containing the ARM CPU.

SoC packages integrating ARM's core designs include Nvidia Tegra's first three generations, CSR plc's Quatro family, ST-Ericsson's Nova and NovaThor, Silicon Labs's Precision32 MCU, Texas Instruments' OMAP products, Samsung's Hummingbird and **Exynos** products, Apple's **A4**, **A5**, and **A5X**, and Freescale's i.MX.

Fabless licensees, who wish to integrate an ARM core into their own chip design, are usually only interested in acquiring a ready-to-manufacture verified **IP core**. For these customers, ARM Holdings delivers a **gate netlist** description of the chosen ARM core, along with an abstracted simulation model and test programs to aid design integration and verification. More ambitious customers, including integrated device manufacturers (IDM) and foundry operators, choose to acquire the processor IP in **synthesizable RTL** (**Verilog**) form. With the synthesizable RTL, the customer has the ability to perform architectural level optimisations and extensions. This allows the designer to achieve exotic design goals not otherwise possible with an unmodified netlist (**high clock speed**, very low power consumption, instruction set extensions, etc.). While ARM Holdings does not grant the licensee the right to resell the ARM architecture itself, licensees may freely sell manufactured product such as chip devices, evaluation boards and complete systems. **Merchant foundries** can be a special case; not only are they allowed to sell finished silicon containing ARM cores, they generally hold the right to re-manufacture ARM cores for other customers.

ARM Holdings prices its IP based on perceived value. Lower performing ARM cores typically have lower licence costs than higher performing cores. In implementation terms, a synthesizable core costs more than a hard macro (blackbox) core. Complicating price matters, a merchant foundry that holds an ARM license, such as Samsung or Fujitsu, can offer fab customers reduced licensing costs. In exchange for acquiring the ARM core through the foundry's in-house design services, the customer can reduce or eliminate payment of ARM's upfront license fee.

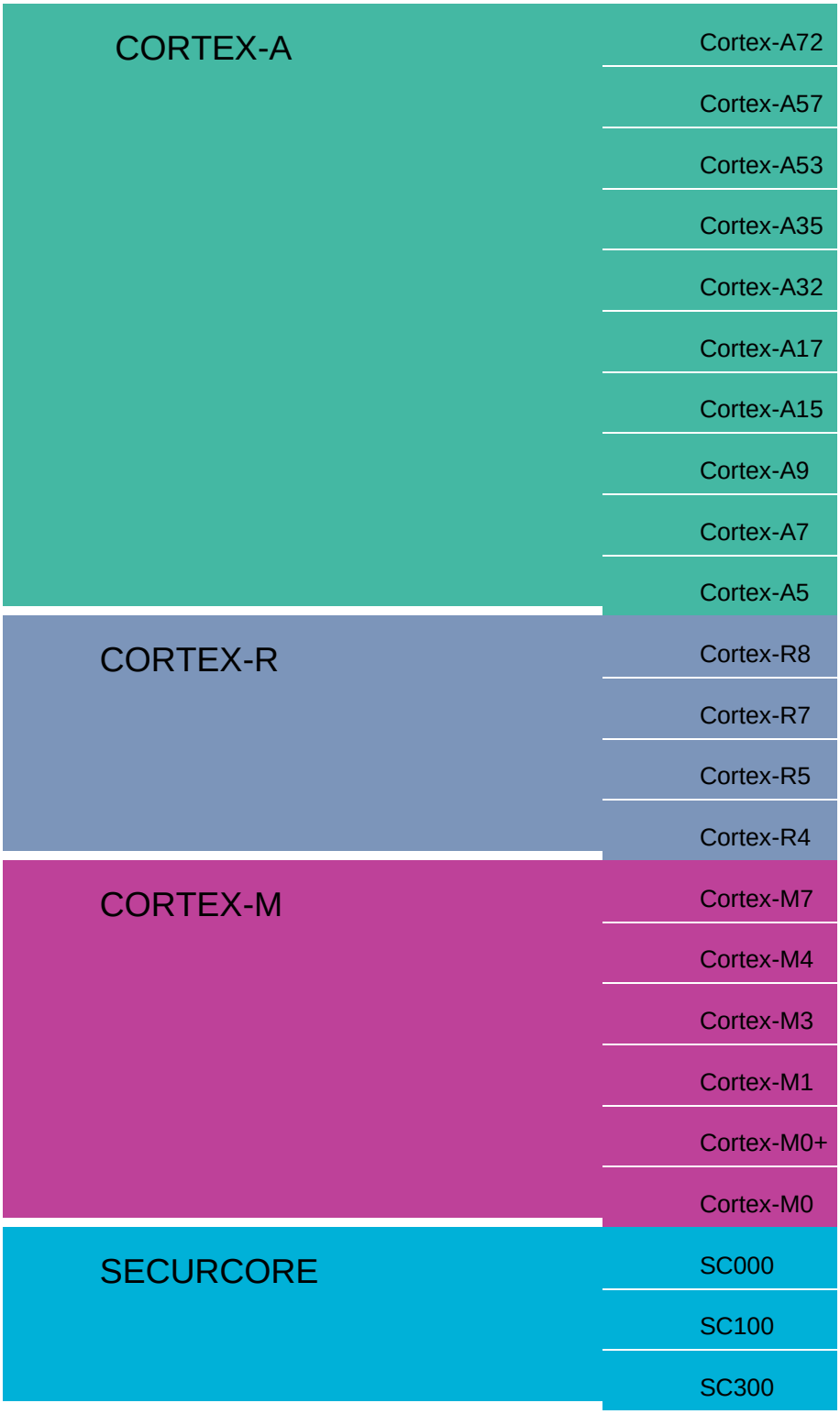
Compared to dedicated semiconductor foundries (such as **TSMC** and **UMC**) without in-house design services, Fujitsu/Samsung charge two- to three-times more per manufactured wafer. For low to mid volume applications, a design service foundry offers lower overall pricing (through subsidization of the license fee). For high volume mass-produced parts, the long term cost reduction achievable through lower wafer pricing reduces the impact of ARM's NRE (Non-Recurring Engineering) costs, making the dedicated foundry a better choice.

Companies that have designed chips with ARM cores include Amazon.com's Annapurna Labs subsidiary, Analog Devices**,** Apple**,** AppliedMicro**,** Atmel**,** Broadcom**,** Cypress Semiconductor**,** Freescale Semiconductor (now NXP Semiconductors), Nvidia**,** NXP**,** Qualcomm**,** Renesas**,** Samsung Electronics**,** ST Microelectronics and Texas Instruments**.**

Architectural license

Companies can also obtain an ARM architectural license for designing their own CPU cores using the ARM instruction sets. These cores must comply fully with the ARM architecture. Companies that have designed cores that implement an ARM architecture include Apple, AppliedMicro, Broadcom, Nvidia, Qualcomm, and Samsung Electronics.

Cores



ARM Cortex Application Processors

Cortex-A Series - High performance processors for feature rich Operating Systems

All of our Cortex-A processors deliver exceptional 32-bit performance for high-end computing, with the new Cortex-A72 processor and both the Cortex-A57 and Cortex-A53 processors delivering combined 32-bit and 64-bit performance, enabling next-generation mobile, networking and server products. The new

Cortex-A35 is the ARM's most efficient 32/64-bit ARMv8-A processor, targeting the next billion entry-level smartphone users. The processors are available in single-core and multi-core varieties, delivering up to four processing units with the ability to integrate **NEON™** multimedia processing blocks and advanced **Floating Point** execution units.

Applications include:

- Smartphones
- Netbooks
- eReaders
- Digital TV
- Home Gateways
- Servers and Networking

ARM Cortex Real-time Embedded Processors

Cortex-R Series - Exceptional performance for real-time applications

Cortex Real-time Embedded processors have been developed for deeply embedded real-time applications where the need for low power and fast and deterministic interrupt control are balanced with exceptional performance and strong compatibility with existing platforms.

Applications include:

- Automotive braking systems
- Powertrain solutions
- Mass storage controller
- Networking & Printing
- Modems and communications

ARM Cortex Embedded Processors

Cortex-M Series - Cost-sensitive solutions for deterministic microcontroller applications

Cortex-M series processors have been developed primarily for the microcontroller domain where the need for fast, highly deterministic, interrupt management is coupled with the desire for extremely low gate count and lowest possible power consumption.

Applications include:

- Microcontrollers
- Mixed signal devices
- Smart sensors
- Automotive body electronics and airbags

ARM Specialist Processors

SecurCore™ - Processors for high security applications

FPGA Cores - Processors for FPGA

ARM Specialist Processors are designed to meet the demanding needs of specific markets. SecurCore processors are utilized within the security markets for mobile SIMs and identification applications and integrate numerous technologies to detect and avoid security attacks while delivering outstanding performance.

ARM also develops processors for FPGA fabrics, enabling users to rapidly reach market while maintaining compatibility with traditional ARM devices. Additionally the fabric independent nature of

these processors enables developers to choose the target device which is right for their application rather than be locked to a specific vendor.

Although most **datapaths** and **CPU registers** in the early ARM processors were 32-bit, **addressable memory was limited to 26 bits**.

ARMv3 included a compatibility mode to support the **26-bit addresses** of earlier versions of the architecture. This compatibility mode optional in ARMv4, and removed entirely in ARMv5.

A list of vendors who implement ARM cores in their design (application specific standard products (ASSP), microprocessor and microcontrollers) is provided by ARM Holdings.

List of applications of ARM cores

ARM cores are used in a number of products, particularly PDAs and smartphones. Some computing examples are Microsoft's first generation Surface and Surface 2**,** Apple's iPads, and Asus's Eee Pad Transformer tablet computers. Others include Apple's iPhone smartphone **and** iPod portable media player, Canon PowerShot digital cameras**,** Nintendo DS handheld game consoles and TomTom turn-by-turn navigation systems.

In 2005, ARM Holdings took part in the development of Manchester University's computer, SpiNNaker, which used ARM cores to simulate the human brain.

ARM chips are also used in Raspberry Pi, BeagleBoard, BeagleBone, PandaBoard and **other** single-board computers, because they are very small, inexpensive and consume very little power.

32-bit architecture

The 32-bit ARM architecture, such as ARMv7-A, is the most widely used architecture in mobile devices.

Since 1995, the **ARM Architecture Reference Manual** has been the primary source of documentation on the ARM processor architecture and instruction set, distinguishing interfaces that all ARM processors are required to support (such as instruction semantics) from implementation details that may vary. The architecture has evolved over time, and version seven of the architecture, ARMv7, defines three architecture "profiles":

A-profile, the "Application" profile, implemented by 32-bit cores in the **Cortex-A** series and by some non-ARM cores;

R-profile, the "Real-time" profile, implemented by cores in the **Cortex-R** series

M-profile, the "Microcontroller" profile, implemented by most cores in the **Cortex-M** series.

Although the architecture profiles were first defined for ARMv7, ARM subsequently defined the ARMv6-M architecture (used by the Cortex **M0**/**M0+**/**M1**) as a subset of the ARMv7-M profile with fewer instructions.

CPU modes

Except in the M-profile, the 32-bit ARM architecture specifies several CPU modes, depending on the implemented architecture features. At any moment in time, the CPU can be in only one mode, but it can switch modes due to external events (interrupts) or programmatically.

User mode: The only non-privileged mode.

FIQ mode: A privileged mode that is entered whenever the processor accepts an **FIQ interrupt**.

IRQ mode: A privileged mode that is entered whenever the processor accepts an IRQ interrupt.

Supervisor (svc) mode: A privileged mode entered whenever the CPU is reset or when an SVC instruction is executed.

Abort mode: A privileged mode that is entered whenever a prefetch abort or data abort exception occurs.

Undefined mode: A privileged mode that is entered whenever an undefined instruction exception occurs.

System mode (ARMv4 and above): The only privileged mode that is not entered by an exception. It can only be entered by executing an instruction that explicitly writes to the mode bits of the CPSR.

Monitor mode (ARMv6 and ARMv7 Security Extensions, ARMv8 EL3): A monitor mode is introduced to support TrustZone extension in ARM cores.

Hyp mode (ARMv7 Virtualization Extensions, ARMv8 EL2): A hypervisor mode that supports **Popek and Goldberg virtualization requirements** for the non-secure operation of the CPU.

Instruction set

The original (and subsequent) ARM implementation was hardwired without **microcode**, like the much simpler **8-bit 6502** processor used in prior Acorn microcomputers.

The 32-bit ARM architecture (and the 64-bit architecture for the most part) includes the following RISC features:

Load/store architecture.

No support for **unaligned memory accesses** in the original version of the architecture. ARMv6 and later, except some microcontroller versions, support unaligned accesses for half-word and single-word load/store instructions with some limitations, such as no guaranteed **atomicity**.

Uniform 16× 32-bit **register file** (including the program counter, stack pointer and the link register).

Fixed instruction width of 32 bits to ease decoding and **pipelining**, at the cost of decreased **code density**. Later, the **Thumb instruction set** added 16-bit instructions and increased code density.

Mostly single clock-cycle execution.

To compensate for the simpler design, compared with processors like the Intel 80286 and **Motorola 68020**, some additional design features were used:

Conditional execution of most instructions reduces branch overhead and compensates for the lack of a **branch predictor**.

Arithmetic instructions alter **condition codes** only when desired.

32-bit **barrel shifter** can be used without performance penalty with most arithmetic instructions and address calculations.

Has powerful indexed **addressing modes**.

A **link register** supports fast leaf function calls.

A simple, but fast, 2-priority-level **interrupt** subsystem has switched register banks.

Arithmetic instructions

ARM includes integer arithmetic operations for add, subtract, and multiply; some versions of the architecture also support divide operations.

ARM supports 32-bit x 32-bit multiplies with either a 32-bit result or 64-bit result, though Cortex-M0 / M0+ / M1 cores don't support 64-bit results. Some ARM cores also support 16-bit x 16-bit and 32-bit x 16-bit multiplies.

The divide instructions are only included in the following ARM architectures:

ARMv7-M and ARMv7E-M architectures always include divide instructions.

ARMv7-R architecture always includes divide instructions in the Thumb instruction set, but optionally in its 32-bit instruction set.

ARMv7-An architecture optionally includes the divide instructions. The instructions might not be implemented, or implemented only in the Thumb instruction set, or implemented in both the Thumb and ARM instruction sets, or implemented if the Virtualization Extensions are included.

Registers

Registers R0 through R7 are the same across all CPU modes; they are never banked. Registers R8 through R12 are the same across all CPU modes except FIQ mode. FIQ mode has its own distinct R8 through R12 registers.

R13 and R14 are banked across all privileged CPU modes except system mode. That is, each mode that can be entered because of an exception has its own R13 and R14. These registers generally contain the stack pointer and the return address from function calls, respectively.

Aliases:

R13 is also referred to as SP, the Stack Pointer.

R14 is also referred to as LR, the Link Register.

R15 is also referred to as PC, the Program Counter.

The Current Program Status Register (CPSR) has the following 32 bits.

M (bits 0–4) is the processor mode bits.

T (bit 5) is the Thumb state bit.

F (bit 6) is the FIQ disable bit.

I (bit 7) is the IRQ disable bit.

A (bit 8) is the imprecise data abort disable bit.

E (bit 9) is the data endianness bit.

IT (bits 10–15 and 25–26) is the if-then state bits.

GE (bits 16–19) is the greater-than-or-equal-to bits.

DNM (bits 20–23) is the do not modify bits.

J (bit 24) is the Java state bit.

Q (bit 27) is the sticky overflow bit.

V (bit 28) is the overflow bit.

C (bit 29) is the carry/borrow/extend bit.

Z (bit 30) is the zero bit.

N (bit 31) is the negative/less than bit.

Conditional execution

Almost every ARM instruction has a conditional execution feature called **predication**, which is implemented with a 4-bit condition code selector (the predicate). To allow for unconditional execution, one of the four-bit codes causes the instruction to be always executed. Most other CPU architectures only have condition codes on branch instructions.

Though the predicate takes up four of the 32 bits in an instruction code, and thus cuts down significantly on the encoding bits available for displacements in memory access instructions, it avoids branch instructions when generating code for small **if statements**. Apart from eliminating the branch instructions themselves, this preserves the fetch/decode/execute pipeline at the cost of only one cycle per skipped instruction.

The standard example of conditional execution is the subtraction-based **Euclidean algorithm**:

In the **C programming language**, the loop is:

```
while (i != j)  // We enter the loop when i<j or i>j, not when i==j
{
  if (i > j)   // When i>j we do that
    i -= j;
  else         // When i<j we do that (since i!=j is checked in while condition)
```

```
        j -= i;
    }
```

In ARM **assembly**, the loop is transformed into:

```
    do
    {
      if      (i > j) // When i>j we do that
         i -= j;
      else if (i < j) // When i<j we do that
         j -= i;
      else
         ;          // When i==j we do nothing
    }
    while (i != j);    // We do nothing into the loop and leave the loop when i==j
```

and coded as:

```
loop:   CMP    Ri, Rj       ; set condition "NE" if (i != j),
                            ;              "GT" if (i > j),
                            ;           or "LT" if (i < j)
        SUBGT  Ri, Ri, Rj   ; if "GT" (Greater Than), i = i-j;
        SUBLT  Rj, Rj, Ri   ; if "LT" (Less Than), j = j-i;
        BNE  loop           ; if "NE" (Not Equal), then loop
```

which avoids the branches around the then and else clauses. If Ri and Rj are equal then neither of the SUB instructions will be executed, eliminating the need for a conditional branch to implement the while check at the top of the loop, for example had SUBLE (less than or equal) been used.

One of the ways that Thumb code provides a more dense encoding is to remove the four bit selector from non-branch instructions.

Other features

Another feature of the **instruction set** is the ability to fold shifts and rotates into the "data processing" (arithmetic, logical, and register-register move) instructions, so that, for example, the C statement

```
a += (j << 2);
```

could be rendered as a single-word, single-cycle instruction:

```
ADD  Ra, Ra, Rj, LSL #2
```

This results in the typical ARM program being denser than expected with fewer memory accesses; thus the pipeline is used more efficiently.

The ARM processor also has features rarely seen in other RISC architectures, such as **PC**-relative addressing (indeed, on the 32-bit ARM the **PC** is one of its 16 registers) and pre- and post-increment addressing modes.

The ARM instruction set has increased over time. Some early ARM processors (before ARM7TDMI), for example, have no instruction to store a two-byte quantity.

Pipelines and other implementation issues

The ARM7 and earlier implementations have a three-stage **pipeline**; the stages being fetch, decode and execute. Higher-performance designs, such as the ARM9, have deeper pipelines: Cortex-A8 has thirteen stages. Additional implementation changes for higher performance include a faster **adder** and more extensive **branch prediction** logic. The difference between the ARM7DI and ARM7DMI cores, for example, was an improved multiplier; hence the added "M".

Coprocessors

The ARM architecture (pre-ARMv8) provides a non-intrusive way of extending the instruction set using "coprocessors" that can be addressed using MCR, MRC, MRRC, MCRR, and similar instructions. The coprocessor space is divided logically into 16 coprocessors with numbers from 0 to 15, coprocessor 15 (cp15) being reserved for some typical control functions like managing the caches and **MMU** operation on processors that have one.

In ARM-based machines, peripheral devices are usually attached to the processor by mapping their physical registers into ARM memory space, into the coprocessor space, or by connecting to another device (a bus) that in turn attaches to the processor. Coprocessor accesses have lower latency, so some peripherals—for example an XScale interrupt controller—are accessible in both ways: through memory and through coprocessors.

In other cases, chip designers only integrate hardware using the coprocessor mechanism. For example, an image processing engine might be a small ARM7TDMI core combined with a coprocessor that has specialised operations to support a specific set of HDTV transcoding primitives.

Debugging

All modern ARM processors include hardware debugging facilities, allowing software debuggers to perform operations such as halting, stepping, and breakpointing of code starting from reset. These facilities are built using **JTAG** support, though some newer cores optionally support ARM's own two-wire "SWD" protocol. In ARM7TDMI cores, the "D" represented JTAG debug support, and the "I" represented presence of an "EmbeddedICE" debug module. For ARM7 and ARM9 core generations, EmbeddedICE over JTAG was a de facto debug standard, though not architecturally guaranteed.

The ARMv7 architecture defines basic debug facilities at an architectural level. These include breakpoints, watchpoints and instruction execution in a "Debug Mode"; similar facilities were also available with EmbeddedICE. Both "halt mode" and "monitor" mode debugging are supported. The actual transport mechanism used to access the debug facilities is not architecturally specified, but implementations generally include JTAG support.

There is a separate ARM "CoreSight" debug architecture, which is not architecturally required by ARMv7 processors.

DSP enhancement instructions

To improve the ARM architecture for **digital signal processing** and multimedia applications, DSP instructions were added to the set. These are signified by an "E" in the name of the ARMv5TE and ARMv5TEJ architectures. E-variants also imply T, D, M and I.

The new instructions are common in **digital signal processor** architectures. They include variations on signed **multiply–accumulate**, saturated add and subtract, and count leading zeros.

SIMD extensions for multimedia

Introduced in ARMv6 architecture, this was a precursor to the advanced SIMD technology also known as **NEON**.

Jazelle DBX (Direct Bytecode eXecution) is a technique that allows Java Bytecode to be executed directly in the ARM architecture as a third execution state (and instruction set) alongside the existing ARM and Thumb-mode. Support for this state is signified by the "J" in the ARMv5TEJ architecture, and in ARM9EJ-S and ARM7EJ-S core names. Support for this state is required starting in ARMv6 (except for the ARMv7-M profile), though newer cores only include a trivial implementation that provides no hardware acceleration.

Thumb

To improve compiled code-density, processors since the ARM7TDMI (released in 1994) have featured the Thumb instruction set, which have their own state. (The "T" in "TDMI" indicates the Thumb feature.) When in this state, the processor executes the Thumb instruction set, a compact 16-bit encoding for a subset of the ARM instruction set. Most of the Thumb instructions are directly mapped to normal ARM instructions. 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.

In Thumb, the 16-bit opcodes have less functionality. For example, only branches can be conditional, and many opcodes are restricted to accessing only half of all of the CPU's general-purpose registers. The shorter opcodes give improved code density overall, even though some operations require extra instructions. In situations where the memory port or bus width is constrained to less than 32 bits, the shorter Thumb opcodes allow increased performance compared with 32-bit ARM code, as less program code may need to be loaded into the processor over the constrained memory bandwidth.

Embedded hardware, such as the **Game Boy Advance**, typically have a small amount of RAM accessible with a full 32-bit datapath; the majority is accessed via a 16-bit or narrower secondary datapath. In this situation, it usually makes sense to compile Thumb code and hand-optimize a few of the most CPU-intensive sections using full 32-bit ARM instructions, placing these wider instructions into the 32-bit bus accessible memory.

The first processor with a Thumb **instruction decoder** was the ARM7TDMI. All ARM9 and later families, including XScale, have included a Thumb instruction decoder.

Thumb-2

Thumb-2 technology was introduced 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 was to achieve code density similar to Thumb with performance similar to the ARM instruction set on 32-bit memory.

Thumb-2 extends the Thumb instruction set with bit-field manipulation, table branches and conditional execution. At the same time, the ARM instruction set was extended to maintain equivalent functionality in both instruction sets. A new "Unified Assembly Language" (UAL) supports generation of either Thumb or ARM instructions from the same source code; versions of Thumb seen on ARMv7 processors are essentially as capable as ARM code (including the ability to write interrupt handlers). This requires a bit of care, and use of a new "IT" (if-then) instruction, which permits up to four successive instructions to execute based on a tested condition, or on its inverse. When compiling into ARM code, this is ignored, but when compiling into Thumb it generates an actual instruction. For example:

```
; if (r0 == r1)
CMP r0, r1
ITE EQ       ; ARM: no code ... Thumb: IT instruction
; then r0 = r2;
MOVEQ r0, r2  ; ARM: conditional; Thumb: condition via ITE 'T' (then)
; else r0 = r3;
MOVNE r0, r3  ; ARM: conditional; Thumb: condition via ITE 'E' (else)
; recall that the Thumb MOV instruction has no bits to encode "EQ" or "NE"
```

All ARMv7 chips support the Thumb instruction set. All chips in the Cortex-A series, Cortex-R series, and ARM11 series support both "ARM instruction set state" and "Thumb instruction set state", while chips in the **Cortex-M** series support only the Thumb instruction set.

Thumb Execution Environment (ThumbEE)

ThumbEE (erroneously called Thumb-2EE in some ARM documentation), marketed as **Jazelle RCT** (Runtime Compilation Target), was announced in 2005, first appearing in the Cortex-A8 processor. ThumbEE is a fourth Instruction set state, making small changes to the Thumb-2 extended Thumb instruction set. These changes make the instruction set particularly suited to code generated at runtime (e.g. by **JIT compilation**) in managed Execution Environments. ThumbEE is a target for languages such as **Java**, **C#**, **Perl**, and **Python**, and allows **JIT compilers** to output smaller compiled code without impacting performance.

New features provided by ThumbEE include automatic null pointer checks on every load and store instruction, an instruction to perform an array bounds check, and special instructions that call a handler. In addition, because it utilizes Thumb-2 technology, ThumbEE provides access to registers r8-r15 (where the Jazelle/DBX Java VM state is held). Handlers are small sections of frequently called code, commonly used to implement high level languages, such as allocating memory for a new object. These changes come from repurposing a handful of opcodes, and knowing the core is in the new ThumbEE Instruction set state.

On 23 November 2011, ARM Holdings deprecated any use of the ThumbEE instruction set, and ARMv8 removes support for ThumbEE.

Floating-point (VFP)

VFP (Vector Floating Point) technology is an FPU (**Floating-Point Unit**) coprocessor extension to the ARM architecture (implemented differently in ARMv8 - coprocessors not defined there). It provides low-cost **single-precision** and **double-precision** floating-point computation fully compliant with the **ANSI/IEEE Std 754-1985 Standard for Binary Floating-Point Arithmetic**. VFP provides floating-point computation suitable for a wide spectrum of applications such as PDAs, smartphones, voice compression and decompression, three-dimensional graphics and digital audio, printers, set-top boxes, and automotive applications. The VFP architecture was intended to support execution of short "vector mode" instructions but these operated on each vector element sequentially and thus did not offer the performance of true **single instruction, multiple data** (SIMD) vector parallelism. This vector mode was therefore removed shortly after its introduction, to be replaced with the much more powerful NEON Advanced SIMD unit.

Some devices such as the ARM Cortex-A8 have a cut-down VFPLite module instead of a full VFP module, and require roughly ten times more clock cycles per float operation. Pre-ARMv8 architecture implemented floating-point/SIMD with the coprocessor interface. Other floating-point and/or SIMD units found in ARM-based processors using the coprocessor interface include **FPA**, FPE, **iwMMXt**, some of which were implemented in software by trapping but could have been implemented in hardware. They provide some of the same functionality as VFP but are not **opcode**-compatible with it.

VFPv1

Obsolete

VFPv2

An optional extension to the ARM instruction set in the ARMv5TE, ARMv5TEJ and ARMv6 architectures. VFPv2 has 16 64-bit FPU registers.

VFPv3 or VFPv3-D32

Implemented on most Cortex-A8 and A9 ARMv7 processors. It is backwards compatible with VFPv2, except that it cannot trap floating-point exceptions. VFPv3 has 32 64-bit FPU registers as standard, adds

VCVT instructions to convert between scalar, float and double, adds immediate mode to VMOV such that constants can be loaded into FPU registers.

VFPv3-D16

As above, but with only 16 64-bit FPU registers. Implemented on Cortex-R4 and R5 processors and the **Tegra 2** (Cortex-A9).

VFPv3-F16

Uncommon; it supports **IEEE754-2008 half-precision (16-bit) floating point**.

VFPv4 or VFPv4-D32

Implemented on the Cortex-A12 and A15 ARMv7 processors, Cortex-A7 optionally has VFPv4-D32 in the case of an FPU with NEON. VFPv4 has 32 64-bit FPU registers as standard, adds both half-precision extensions and **fused multiply-accumulate** instructions to the features of VFPv3.

VFPv4-D16

As above, but it has only 16 64-bit FPU registers. Implemented on Cortex-A5 and A7 processors (in case of an FPU without NEON).

VFPv5-D16-M

Implemented on Cortex-M7 when single and double-precision floating-point core option exists.

In **Debian** Linux and derivatives armhf (ARM hard float) refers to the ARMv7 architecture including the additional VFP3-D16 floating-point hardware extension (and Thumb-2) above.

Software packages and cross-compiler tools use the armhf vs. arm/armel suffixes to differentiate.

Advanced SIMD (NEON)

The Advanced SIMD extension (aka NEON or "MPE" Media Processing Engine) is a combined 64- and **128-bit** SIMD instruction set that provides standardized acceleration for media and signal processing applications. NEON is included in all Cortex-A8 devices but is optional in Cortex-A9 devices. NEON can execute MP3 audio decoding on CPUs running at 10 MHz and can run the **GSM adaptive multi-rate** (AMR) speech **codec** at no more than 13 MHz. It features a comprehensive instruction set, separate register files and independent execution hardware. NEON supports 8-, 16-, 32- and 64-bit integer and single-precision (32-bit) floating-point data and SIMD operations for handling audio and video processing as well as graphics and gaming processing. In NEON, the SIMD supports up to 16 operations at the same time. The NEON hardware shares the same floating-point registers as used in VFP. Devices such as the ARM Cortex-A8 and Cortex-A9 support 128-bit vectors but will execute with 64 bits at a time, whereas newer Cortex-A15 devices can execute 128 bits at a time.

ProjectNe10 is ARM's first open source project (from its inception). The Ne10 library is a set of common, useful functions written in both NEON and C (for compatibility). The library was created to allow developers to use NEON optimisations without learning NEON but it also serves as a set of highly optimised NEON intrinsic and assembly code examples for common DSP, arithmetic and image processing routines. The code is available on **GitHub**.

Security extensions (TrustZone)

The Security Extensions, marketed as TrustZone Technology, is in ARMv6KZ and later application profile architectures. It provides a low-cost alternative to adding another dedicated security core to an SoC, by providing two virtual processors backed by hardware based access control. This lets the application core switch between two states, referred to as worlds (to reduce confusion with other names for capability domains), in order to prevent information from leaking from the more trusted world to the less trusted world. This world switch is generally orthogonal to all other capabilities of the processor, thus each world can operate independently of the other while using the same core. Memory and peripherals are then made aware of the operating world of the core and may use this to provide access control to secrets and code on the device.

Typical applications of TrustZone Technology are to run a rich operating system in the less trusted world, and smaller security-specialized code in the more trusted world (named TrustZone Software, a TrustZone optimised version of the Trusted Foundations Software developed by **Trusted Logic Mobility**), allowing much tighter **digital rights management** for controlling the use of media on ARM-based devices, and preventing any unapproved use of the device. Trusted Foundations Software was acquired by Gemalto. Giesecke & Devrient developed a rival implementation named Mobicore. In April 2012 ARM Gemalto and Giesecke & Devrient combined their TrustZone portfolios into a joint venture Trustonic. Open Virtualization and T6 are open source implementations of the trusted world architecture for TrustZone.

In practice, since the specific implementation details of TrustZone are proprietary and have not been publicly disclosed for review, it is unclear what level of assurance is provided for a given **threat model**.

No-execute page protection

As of ARMv6, the ARM architecture supports **no-execute page protection**, which is referred to as XN, for eXecute Never.

Large Physical Address Extension

The Large Physical Address Extension, which extends the physical address size from 32 bits to 40 bits, was added to the ARMv7-A architecture in 2011.

ARMv8-R

The ARMv8-R sub-architecture, announced after the ARMv8-A, shares some features except that it is not 64-bit.

64/32-bit architecture

ARMv8-A

Announced in October 2011, ARMv8-A (often called ARMv8 although not all variants are 64-bit such as ARMv8-R) represents a fundamental change to the ARM architecture. It adds a 64-bit architecture, named "AArch64", and a new "A64" instruction set. AArch64 provides **user-space** compatibility with ARMv7-A ISA, the 32-bit architecture, therein referred to as "AArch32" and the old 32-bit instruction set, now named "A32". The Thumb instruction sets are referred to as "T32" and have no 64-bit counterpart. ARMv8-A allows 32-bit applications to be executed in a 64-bit OS, and a 32-bit OS to be under the control of a 64-bit **hypervisor**. ARM announced their Cortex-A53 and Cortex-A57 cores on 30 October 2012. Apple was the first to release an ARMv8-A compatible core (**Apple A7**) in a consumer product (**iPhone 5S**). **AppliedMicro**, using an **FPGA**, was the first to demo ARMv8-A.The first ARMv8-A **SoC** from **Samsung** is the Exynos 5433 in the Galaxy Note 4, which features two clusters of four Cortex-A57 and Cortex-A53 cores in a big.LITTLE configuration; but it will run only in AArch32 mode.

To both AArch32 and AArch64, ARMv8-A makes VFPv3/v4 and advanced SIMD (NEON) standard. It also adds cryptography instructions supporting **AES** and **SHA-1**/**SHA-256**.

ARMv8.1-A

In December 2014, ARMv8.1-A, an update with "incremental benefits over v8.0", was announced. The enhancements fall into two categories:

Changes to the instruction set

Changes to the exception model and memory translation

Expected "product introductions mid-2015" with server CPU makers likely to adopt and Apple "will likely jump to the new architecture". "The incremental updates in ARMv8.1-A revolve around memory addressing, security, virtualization and throughput. ARMv8-A code will run on v8.1 cores."

AArch64 features

New instruction set, A64

Has 31 general-purpose 64-bit registers.

Has dedicated SP or zero register.

The program counter (PC) is no longer accessible as a register.

Instructions are still 32 bits long and mostly the same as A32 (with LDM/STM instructions and most conditional execution dropped).

Has paired loads/stores (in place of LDM/STM).

No **predication** for most instructions (except branches).

Most instructions can take 32-bit or 64-bit arguments.

Addresses assumed to be 64-bit.

Advanced SIMD (NEON) enhanced

Has 32× 128-bit registers (up from 16), also accessible via VFPv4.

Supports double-precision floating point.

Fully IEEE 754 compliant.

AES encrypt/decrypt and SHA-1/SHA-2 hashing instructions also use these registers.

A new exception system

Fewer banked registers and modes.

Memory translation from 48-bit virtual addresses based on the existing Large Physical Address Extension (LPAE), which was designed to be easily extended to 64-bit

Operating system support

32-bit operating systems

Historical operating systems

The first 32-bit ARM-based personal computer, the Acorn Archimedes, ran an interim operating system called **Arthur**, which evolved into **RISC OS**, used on later ARM-based systems from Acorn and other vendors. Some Acorn machines also had a **Unix** port called **RISC iX**.

Embedded operating systems

The 32-bit ARM architecture is supported by a large number of embedded and real-time operating systems, including Android**,** Linux**,** FreeRTOS**,** VxWorks**,** Windows Embedded Compact**,** Windows 10 IoT Core**,** ChibiOS/RT**,** DRYOS**,** eCos**,** Integrity**,** Nucleus PLUS**,** NuttX**,** MicroC/OS-II**,** PikeOS**,** QNX**,** RIOT**,** RTEMS**,** RTXC Quadros**,** ThreadX**,** MQX**,** T-Kernel**,** OSE**,** OS-9**.**

Real time operating systems

SCIOPTA has a first proof-of-concept implementation running.

Mobile device operating systems

The 32-bit ARM architecture is the primary hardware environment for most mobile device operating systems such as Android**,** iOS**,** Windows Phone**,** Windows RT**,** Bada**,** BlackBerry OS**/**BlackBerry 10**,** Chrome OS**,** Firefox OS**,** MeeGo**,** Tizen**,** Ubuntu Touch**,** Sailfish**,** Symbian**,** and webOS**.**

Desktop/server operating systems

The 32-bit ARM architecture is supported by RISC OS and multiple **Unix-like** operating systems including **BSD** (**NetBSD**, **FreeBSD**, **OpenBSD**), **OpenSolaris** and various **Linux distributions** such as **Debian**, **Gentoo**, and **Ubuntu**.

64-bit operating systems

Mobile device operating systems

iOS supports ARMv8-A in **iOS 7** and later on 64-bit **Apple SoCs**.

Android supports ARMv8-A in **Android Lollipop** (5.0) and later.

Desktop/server operating systems

Support for ARMv8-A was merged into the **Linux kernel** version 3.7 in late 2012.

ARMv8-A is supported by a number of **Linux distributions**, such as **Debian**, **Fedora**, **open SUSE**. Support for ARMv8-A was merged into **FreeBSD** in late 2014.

Porting to 32- or 64-bit ARM operating systems

Windows applications recompiled for ARM and linked with Winelib – from the **Wine** project can run on 32-bit or 64-bit ARM in Linux (or FreeBSD or other compatible enough operating systems).

Intel's x86 binaries, e.g. when not specially compiled for ARM, work with Wine (on Linux, OS X and more); and have been demonstrated on ARM, with **QEMU**, but do not work at full speed or same capability as with Winelib.

References

[1] *Grisenthwaite, Richard (2011).* "ARMv8-A Technology Preview" *. Retrieved 31 October 2011.*

[2] *"Procedure Call Standard for the ARM Architecture" . ARM Holdings. 30 November 2013. Retrieved 27 May 2013.*

[3] "Some facts about the Acorn RISC Machine" Roger Wilson posting to comp.arch, 2 November 1988. Retrieved 25 May 2007.

[4] "ARM Cores Climb Into 3G Territory" by Mark Hachman, 2002.

[5] "The Two Percent Solution" by Jim Turley 2002.

[6] *"ARM Discloses Technical Details Of The Next Version Of The ARM Architecture" (Press release). ARM Holdings. 27 October 2011. Retrieved 20 September 2013.*

[7] *Tracy Robinson (12 February 2014), Celebrating 50 Billion shipped ARM-powered Chips**,** retrieved 31 January 2016*

[8] *"MCU Market on Migration Path to 32-bit and ARM-based Devices: 32-bit tops in sales; 16-bit leads in unit shipments". IC Insights. 25 April 2013. Retrieved 1 July 2014.*

[9] *Hachman, Mark (2002). "ARM Cores Climb into 3G Territory". ExtremeTech.*

[10] *Turley, Jim (2002). "The Two Percent Solution". www.embedded.com.*

[11] *ARM Holdings eager for PC and server expansion, 1 February 2011*

[12] *Kerry McGuire Balanza (11 May 2010), ARM from zero to billions in 25 short years**,** ARM Holdings, retrieved 8 November 2012*

[13] *Acorn Archimedes Promotion from 1987. 1987.*

[14] *Manners, David (29 April 1998). "ARM's way"**.** Electronics Weekly. Retrieved 26 October 2012.*

[15] *Chisnall, David (23 August 2010). "Understanding ARM Architectures". Retrieved 26 May 2013.*

[16] *Furber, Stephen B. (2000). ARM system-on-chip architecture. Boston: Addison-Wesley. ISBN 0-201-67519-6.*

[17] *Goodwins, Rupert (4 December 2010). "Intel's victims: Eight would-be giant killers". ZDNet. Retrieved 7 March 2012.*

[18] *Levy, Markus. "The History of The ARM Architecture: From Inception to IPO" . Retrieved 14 March 2013.*

[19] *Santanu Chattopadhyay (1 January 2010). Embedded System Design. PHI Learning Pvt. Ltd. p. 9. ISBN 978-81-203-4024-4. Retrieved 15 March 2013.*

[20] *Weber, Jonathan (28 November 1990). "Apple to Join Acorn, VLSI in Chip-Making Venture". Los Angeles Times (Los Angeles). Retrieved 6 February 2012. Apple has invested about $3 million (roughly 1.5 million pounds) for a 30% interest in the company, dubbed Advanced Risc Machines Ltd.*

[21] *DeMone, Paul (9 November 2000). "ARM's Race to Embedded World Domination". Real World Technologies. Retrieved 6 October 2015.*

[22] *"March of the Machines". technologyreview.com. MIT Technology Review. 2010-04-20. Retrieved 6 October 2015.*

[23] *Krazit, Tom (3 April 2006). "ARMed for the living room". CNet.com.*

[24] *Fitzpatrick, J. (2011). "An Interview with Steve Furber". Communications of the ACM 54 (5): 34. doi:10.1145/1941487.1941501.*

[25] *Tracy Robinson (12 February 2014). "Celebrating 50 Billion shipped ARM-powered Chips".*

[26] *Sarah Murry (3 March 2014). "ARM's Reach: 50 Billion Chip Milestone".*

[27] *Brown, Eric (2009). "ARM netbook ships with detachable tablet".*

[28] *McGrath, Dylan (18 July 2011). "IHS: ARM ICs to be in 23% of laptops in 2015". EE Times. Retrieved 20 July 2011.*

[29] *Peter Clarke (7 January 2016). "Amazon Now Sells Own ARM chips".*

[30] *Nolting, Stephan. "STORM CORE Processor System" . opencores.org. Retrieved 1 April 2014.*

[31] *"ARM Launches Cortex-A50 Series, the World's Most Energy-Efficient 64-bit Processors" (Press release). ARM Holdings. Retrieved 31 October 2012.*

[32] Cavium Thunder X ups the ARM core count to 48 on a single chip; SemiAccurate SemiAccurate.com; Jun 3, 2014

[33] Cavium at Supercomputing 2014 Yahoo Finance; November 17, 2014

[34] Cray to Evaluate ARM Chips in Its Supercomputers eWeek; November 17, 2014

[35] ARMv8 Architecture Technology Preview (Slides); ARM Holdings.

[36] *"ARM Cortex-R Architecture" . ARM Holdings. October 2013. Retrieved 1 February 2014.*

[37] *"ARMv8-M Architecture Simplifies Security for Smart Embedded - ARM". ARM. Retrieved November 10, 2015.*

[38] *Parrish, Kevin (14 July 2011). "One Million ARM Cores Linked to Simulate Brain". EE Times. Retrieved 2 August 2011.*

[39] *"Processor mode". ARM Holdings. Retrieved 26 March 2013.*

[40] *Brash, David (August 2010). "Extensions to the ARMv7-A Architecture" . ARM Ltd. Retrieved 6 June 2014.*

[41] "How does the ARM Compiler support unaligned accesses?". 2011. Retrieved 5 October 2013.

[42] "ARM DSP Instruction Set Extensions". Arm.com. Archived from the original on 14 April 2009. Retrieved 18 April 2009.

[43] "ARM Processor Instruction Set Architecture". Arm.com. Archived from the original on 15 April 2009. Retrieved 18 April 2009.

[44] "ARM aims son of Thumb at uCs, ASSPs, SoCs". Linuxdevices.com. Retrieved 18 April 2009.

[45] "ARM Information Center". Infocenter.arm.com. Retrieved 18 April 2009.

[46] "Arm strengthens Java compilers: New 16-Bit Thumb-2EE Instructions Conserve System Memory" by Tom R. Halfhill 2005.

[47] "ARM Compiler toolchain Using the Assembler — VFP coprocessor". Arm.com. Retrieved 20 August 2014.

[47] "VFP directives and vector notation". Arm.com. Retrieved 21 November 2011.

[48] "Differences between ARM Cortex-A8 and Cortex-A9". Shervin Emami. Retrieved 21 November 2011.

[49] "About the Cortex-A9 NEON MPE". Arm.com. Retrieved 21 November 2011.

[50] "Genode - An Exploration of ARM TrustZone Technology". Retrieved 10 July 2015.

[51] "ARM Announces Availability of Mobile Consumer DRM Software Solutions Based on ARM T". News.thomasnet.com. Retrieved 18 April 2009.

[53] "ARM, Gemalto and Giesecke & Devrient Form Joint Venture To". ARM Holdings. 3 April 2012. Retrieved 19 January 2013.

[54] "ARM TrustZone and ARM Hypervisor Open Source Software". Open Virtualization. Retrieved 14 June 2013.

[55] "T6: TrustZone Based Trusted Kernel". trustkernel. 8 July 2014. Retrieved 8 July 2014.

[56] "AppliedMicro Showcases World's First 64-bit ARM v8 Core" (Press release). AppliedMicro. 28 October 2011. Retrieved 11 February 2014.

[57] "Samsung's Exynos 5433 is an A57/A53 ARM SoC". AnandTech. Retrieved 17 September 2014.

[58] Brash, David (2 December 2014). "The ARMv8-A architecture and its ongoing development". Retrieved 23 January 2015.

[59] Shah, Agam (3 December 2014). "ARM technology in Apple's A7, A8 chips gets an upgrade". Retrieved 23 January 2015.

[60] "PikeOS Safe and Secure Virtualization". Retrieved 10 July 2013.

[61] "Safety Certified Real-Time Operating Systems - Supported CPUs".

[62] "ARM Platform Port". opensolaris.org. Retrieved 29 December 2012.

[63] Linus Torvalds (1 October 2012). "Re: [GIT PULL] arm64: Linux kernel port". Linux kernel mailing list. Retrieved 2 October 2012.

[64] Larabel, Michael (27 February 2013). "64-bit ARM Version Of Ubuntu/Debian Is Booting". Phoronix. Retrieved 17 August 2014.

[65] "64-bit ARM architecture project update". The FreeBSD *Foundation. 24 November 2014.*

* Mr. Nikola Zlatanov spent over 20 years working in the Capital Semiconductor Equipment Industry. His work at Gasonics, Novellus, Lam and KLA-Tencor involved progressing electrical engineering and management roles in disruptive technologies. Nikola received his Undergraduate degree in Electrical Engineering and Computer Systems from Technical University, Sofia, Bulgaria and completed a Graduate Program in Engineering Management at Santa Clara University. He is currently consulting for Fortune 500 companies as well as Startup ventures in Silicon Valley, California.