ARM V8 FAMILY PROCESSORS

Silvia González Rodríguez
Miguel Enrique Játiva Jiménez
Miguel Sánchez de la Rosa

Computer Architecture

December 2019

Contents

List of Figures

List of Tables

1 Historic Context

ARM, previously Advanced RISC Machine, originally Acorn RISC Machine, is a family of reduced instruction set computing (RISC) architectures for computer processors, configured for various environments. 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) and systems on modules (SoM) 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 desings into their own products.[1]

The British computer manufacturer Acorn Computers first developed the Acorn RISC Machine architecture (ARM) in the 1980s to use in its personal computers. Its first ARM-based products were coprocessor modules for the BBC Micro series of computers. After the successful BBC Micro computer, Acorn Computers considered how to move on from the relatively simple MOS Technology 6502 processor to address business markets like the one that was soon dominated by the IBM PC, launched in 1981. The Acorn Business Computer (ABC) plan required that a number of second processors be made to work with the BBC Micro platform, but processors such as the Motorola 68000 and National Semiconductor 32016 were considered unsuitable, and the 6502 was not powerful enough for a graphics-based user interface.[1]

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 (DMA) hardware. The first samples of ARM silicon worked properly when first received and tested on 26 April 1985.[1]

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.[1]

In the late 1980s, Apple Computer and VLSI Technology started working with Acorn on newer versions of the ARM core. In 1990, Acorn spun off the design team

into a new company named Advanced RISC Machines Ltd., which became ARM Ltd when its parent company, Arm Holdings plc, floated on the London Stock Exchange and NASDAQ in 1998. 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.[1]

In 1994, Acorn used the ARM610 as the main central processing unit (CPU) in their RiscPC computers. DEC licensed the ARMv4 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 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 throughout these changes; ARM2 had 30,000 transistors, while ARM6 grew only to 35,000.[1]

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".[1]

2 General characteristics

The ARM Cortex-A72 is a microarchitecture implementing the ARMv8-A 64-bit instruction set designed by ARM Holdings' Austin design centre. The Cortex-A72 is a 3-wide decode out-of-order superscalar pipeline. It is available as SIP core to licensees, and its design makes it suitable for integration with other SIP cores (e.g. GPU, display controller, DSP, image processor, etc.) into one die constituting a system on a chip (SoC). The Cortex-A72 was announced in 2015 to serve as the successor of the Cortex-A57, and was designed to use 20% less power or offer 90% greater performance. It has one to four cores in a single processor device with L1 and L2 cache subsystems.[2] The following figure 1 shows an example block diagram of a Cortex-A72 processor configuration with four cores.

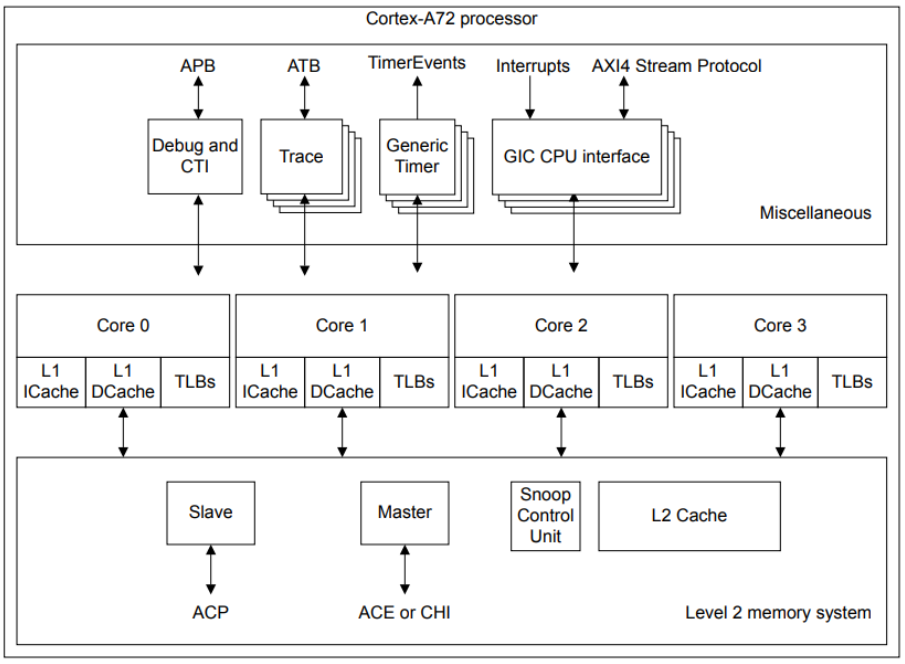

Figure 1: Example Cortex-A72 processor configuration

The Cortex-A72 boasts up to a 50 percent energy reduction when compared with the Cortex-A15 and a 20 percent saving compared with the A57, at the same clock speeds. Milliwatts per core have dropped from the A57, to around 700mW at 2.5GHz.[3] The design takes up 10 percent less area than the A57, which will also help save on costs. We can see this in figure 2.

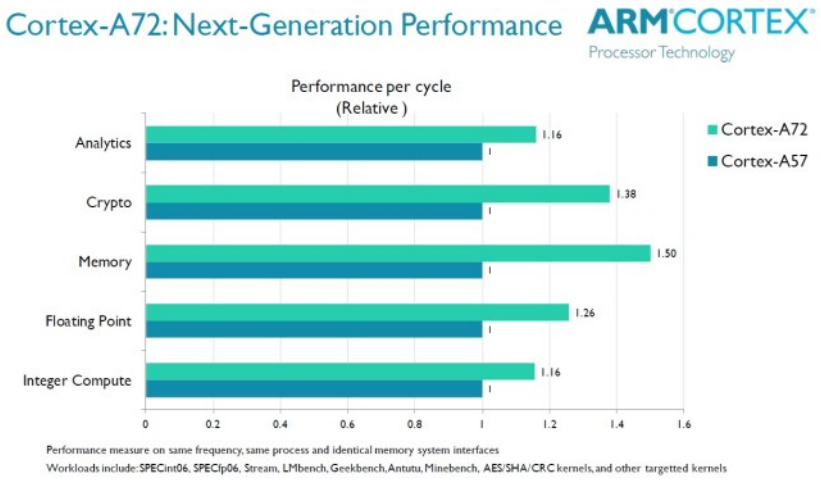

Figure 2: Cortex-A72 performance

As well as substantial energy savings, ARM reckons that the A72 will be able to sustain 2.5GHz clocks on the new 16nm process, whilst keeping within the limited smartphone power budget as seen in figure 3. It's the additional power efficiency

and resulting lower heat profile that will really help the A72 achieve higher clock speeds than a 16nm A57.[3]

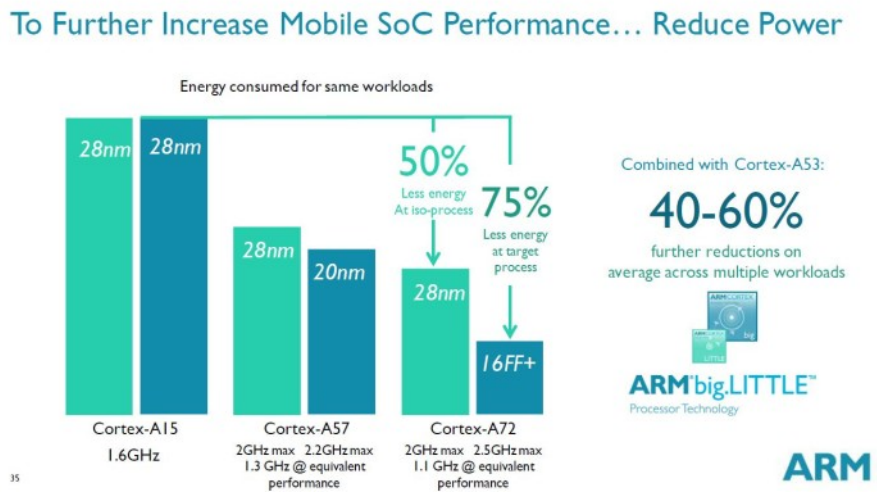

Figure 3: Cortex-A72 power reduction

ARM is looking to differentiate its high performing designs from their lower energy counterparts. The A53 and A57 are quite different in their design and intended applications, so switching the more powerful cores over to the A7x naming scheme should help avoid any confusion in the future.[3]

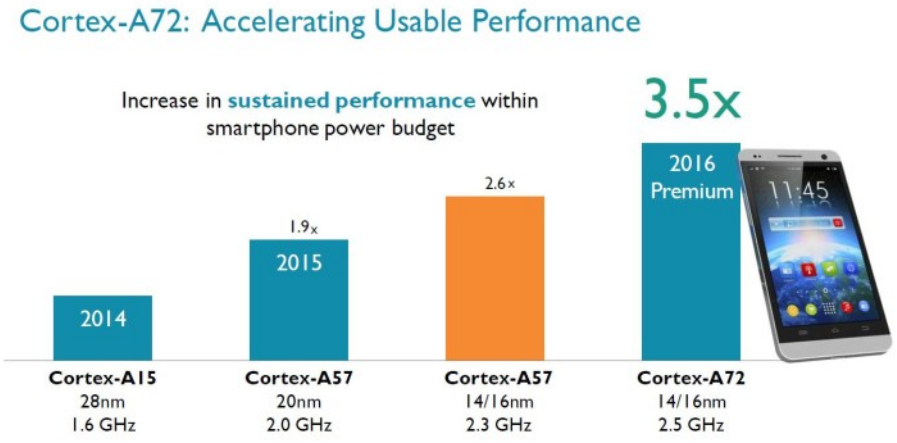

Figure 4: Cortex-A72 usable performance

The key point to take-away is that ARM has focused heavily on improving power and area efficiency with the A72, which is always welcome in mobile products. This also has the added benefit of allowing the chip to run cooler and to be clocked slightly higher than its predecessor.[3]

3 Architectural characteristics

The Cortex-A72 processor implements the ARMv8-A architecture. This includes the A64 instruction set.

A64 is a 32-bit fixed-length instruction set that machines in AArch64 state execute. This instruction set was introduced with the ARMv8 ISA.

The A64 instruction set was introduced with the ARMv8 ISA in order to introduce 64-bit capabilities, primarily targeting the LP64 and LPP64 64-bit data models. It is fixed-length with instructions being 32 bits in size and follow almost identical semantics as in AArch32. This was done in order to maximize the hardware reuse. With the exception of branches and comparisons, predication was also eliminted. Compared to the 32-bit instruction set which offered 14 general-purpose registers, the A64 IS has 31 GPRs. Most instructions are capable of having a 32-bit or 64-bit operand and the address is assumed to be 64-bit in size.[4]

3.1 Registers

A64 has 31 general-purpose registers, all of which are 64-bit wide. Depending on the instruction, the 32nd register is used to obtain a zero or the stack pointer. The almost doubling of the GPR count was done in order to improve both the performance and power of the implementations (designed to reduce cache accesses). Banking was eliminated altogether. Additionally, the stack pointer is no longer regarded as a GPR and the program counter is no longer treated as a normal directly accessible register.

Compared to A32, smaller registers no longer packed into the larger ones but are instead mapped one-to-one to the low-order bits of the bigger register. Registers W0 through W30 are 32 bit and register X0 through X30 are 64 bits.[4]

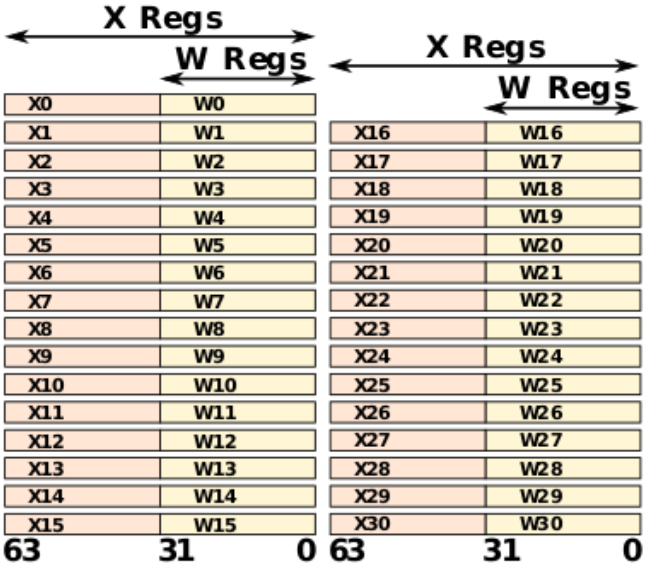

Figure 5: Arm A64 registers

3.2 Instructions

Most instructions are capable of supporting both 32-bit or 64-bit operands. A64 primarily targeted the LP64 and LPP64 data models, meaning long, long long integers, and pointers are 64-bit wide while integers are 32 bit. A64 has a limited set of instructions supporting predication and arbitrary length load/store are no longer supported.[4]

The A64 instruction set overloads instruction mnemonics. That is, it distinguishes between the different forms of an instruction, based on the operand register names that are used. For example, the following ADD instructions all have different forms, but you only have to remember one instruction and the assembler automatically chooses the correct encoding, based on the operands used.[5]

```
ADD W0, W1, W2              // add 32-bit registers
ADD X0, X1, X2              // add 64-bit registers
ADD X0, X1, W2, SXTW        // add sign extended 32-bit register to 64-bit
                            // extended register
ADD X0, X1, #42             // add immediate to 64-bit register
ADD V0.8H, V1.8H, V2.8H     // NEON 16-bit add, in each of 8 lanes
```

Figure 6: Arm A64 ADD Instructions

3.3 Advanced SIMD

A64 Advanced SIMD (NEON) shares the same register file as the A32 floating-point register file but is extended to 128-bit. A64 has 32 128-bit wide vector registers (this is double the amount from A32). A64 advanced SIMD added support for double-precision floating point as well as IEE 754 compliance. Additional instructions were also added to support the 2008 version of the standard (e.g., conversion, min/max).[4]

Media registers are referred to Vn. The 64-bit (Dn) and 32-bit (Sn) forms can also be accessed.

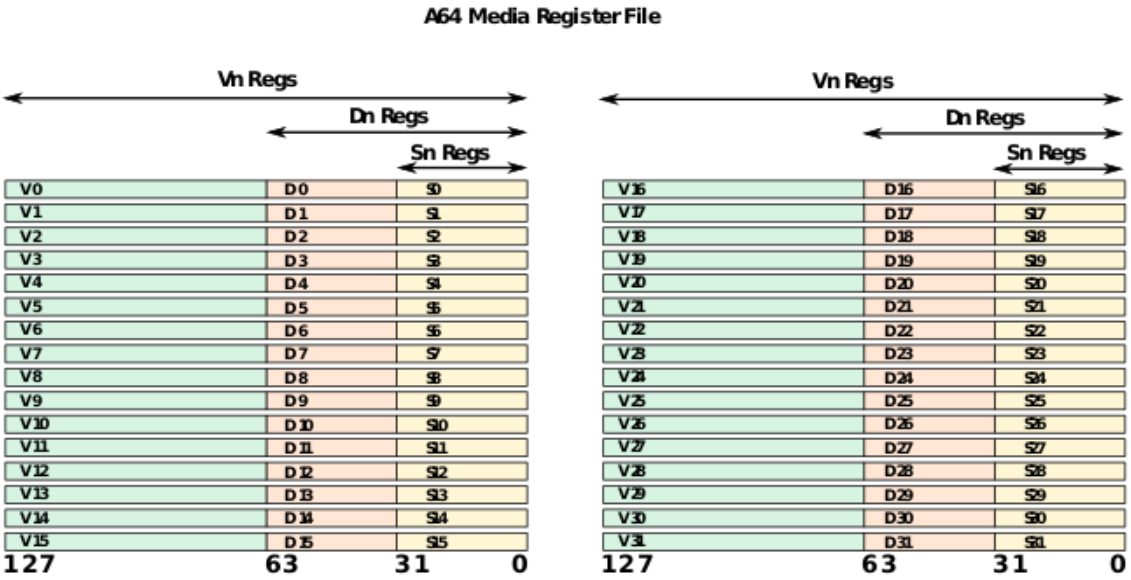

Figure 7: Arm A64 Media Register File

4 Internal organization/structure (datapath)

The internal structure is the one pictured in Figure 1. And the details of the blocks are described in section 2.1.1 of source [2].

Fetch, decode and dispatch units

Instruction Fetch

Up to 3 instructions are fetched from the L1 instruction memory. The cache has 48KB of size in a 3-way associativity configuration with 64-bits-long cache lines; and optionally, parity bits for error-checking in the data and tag. In order to facilitate address prediction and translation, the architecture provides this features:

- For fast virtual address translation, a Translation Lookaside Buffer (TLB) is with support for pages of sizes 4KB, 64KB, and 1MB.
- 2-level dynamic Branch Target Buffer (BTB). A static predictor is also provided.

Instruction decode

Types of instructions able to be decoded by the CPU (including)[2]:

- A32: 32-bit wide, aligned on 4-byte boundaries. Both A-profile and R-profile are supported (application and real-time applications respectively) It is the most supported ISA. [6]
- T32: mixed 32 and 16-bit length instruction set, aiming at lost memory footprint and cost. It's the only ISA supported by type M instructions (specific

to microcontrollers)[8]

- A64. Instructions semantics similar to A32 and T32, with 31 64-bit general purpose registers; independent from Program Counter and Stack Pointer, and hard-coded zero register. [7]

  This unit performs register renaming to facilitate out-of-order execution; thus removing both WAW and WAR hazards.

Instruction dispatch

The dispatch unit controls when the instructions that are decoded are ready to be issued, and afterwards, when the the results are retired. [2]

Functional units for execution and memory access

This set of functional units is composed by the units for operating with integers, floating point values, and memory accesses. [2]

Integer execute

- 2 ALU pipelines.
- Integer multiply-accumulate and ALU pipelines.
- Iterative integer dividing unit.
- Unit for resolution and prediction of branches.
- Result forwarding routes.

Load/Store unit

This unit is responsible of accessing L1 memory cache and service memory coherency requests with the second cache level (L2).

L1 is a 32KB 2-way associative cache with 64-bits-long lines, and can optionally implement Error Correction Code (ECC)[1] for every 32 bits. Apart from the aforementioned TLB (see Instruction Fetch), this unit also uses automatic pre-fetching targeting the L1 data cache and L2. [2]

Level-2 of cache

Services the both the data and instruction L1 in case of cache miss, and keeps coherence. For that, it keeps a duplicate of the tag fields of each L1 cache. Its size is either 12KB, 1MB, 2MB, or 4MB; with a 16-way associativity with optional ECC data per 64 bits; and a 1024-entry TLB **for each processor**. This cache, as explained in the section 7.2 of [2] has a *dirty* bit per line of cache (That is, per 64 bits). For a vision of the whole cache parameters, see table 1

[1]The difference between using parity bits in memories and using ECC memory is that, while parity bits helps **detecting** errors; ECC memory can both **detect** and **repair** those errors

Advanced SIMD and floating-point

It gives support to ARMv8 advanced SIMD (Simple Input, multiple data) and floating-point operations. Optionally, it is up to the implementation to include an optional cryptography engine, albeit with contractual rights. [2]

Interrupts and debug

GIC CPU interface

This unit delivers interrupts to the processor. Examples of interruptions: device signals (device ready, read/write completed, error; battery and power), clock IRQ (for scheduling), exceptions. [2]

Generic Timer

Provides with the ability to trigger interrupts and trigger events. [2]

Debug and trace

- ARMv8 debug architecture with slave architecture with debug registers. It provides the programmer with registers for discerning the interrupts **source**, **behavior** to apply, and **routing** to other processors.
- Performance Monitor Unit: creates statistic data from **runtime** status of the CPU and the memory
- Embedded Trace Macrocells: performance of real-time instruction flow tracing.

  [2]

5 Cache memory

Table 1: Cache parameters for Cortex-A72

Cache level	Sizes available	Associativity	Cache line length
Level 1 - Data	32KB **per core**	2-way set assoc.	64 bits
Level 1 - Instruction	48KBs	3-way set assoc.	64 bits
Level 2	12KB, 1MB, 2MB, or 4MB	16-way associativity	64 bits

In table 1 we have a unified version for all the information regarding cache that was defined previously.

Regarding writing policy, write requests to main memory use as policy **Write-back-write-allocate**. This means that updating MM only takes place when cache is *dirty* and prompt to be discarded, and not on MM directly; and upon a miss in cache, the block is brought to the upper levels and allocated there for it to be read or written, and not directly on main memory. [2] 6.4.1

No access time or bandwidth was found for the cache memories, however, in some implementations, there's information regarding memory controller, therefore cache memory should be faster to access and transfer data from/to. In that specific implementation, the maximum bandwidth for secondary memory is $11.92GiB/s$; and the most powerful technology to use in this CPU *LPDDR3-1600*, 1600 indicating its frequency in MHz.[9]

6 I/O support

The I/O of this architecture is provided by interfaces with other units or components, being mainly these:

- Memory interface: provides the following advantages:
  - Hardware cache coherency (faster than software).
  - Transaction ordering for memory.
  - Management of virtual memory management.
- (Optional) Generic Interrupt Controller, already described in section 4.
- Debug interface, also defined in 4; the same as the trace interface.
- Performance Monitor Unit: provides useful information about the processor when debugging or profiling code. By using six counter can be used to keep track of available events. Those events can be codified and stored in registers of the architecture itself or the APB architecture. [2] 11.2.1

  The block diagram is as follows in figure 8:

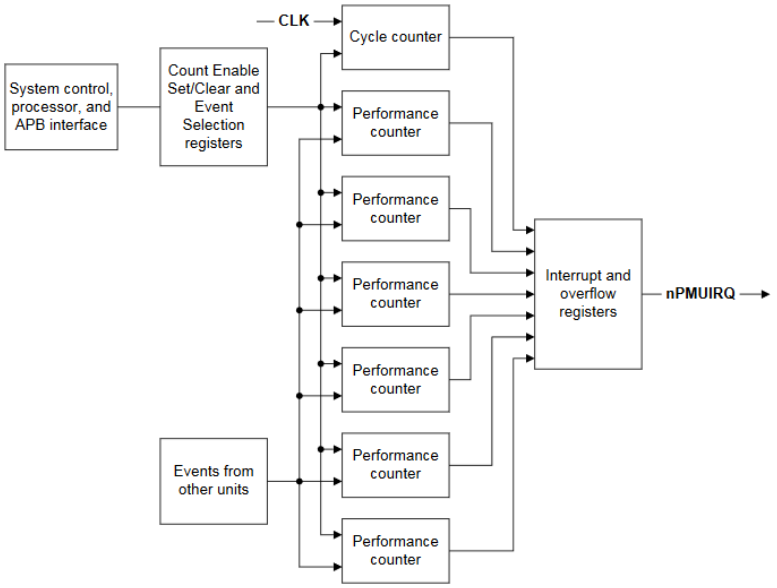

Figure 8: PMU block diagram

7 Pipeline characteristics

Number of stages

The Cortex-A72 processor has a variable-length pipeline, the minimum length of which is 15 stages. This is a result of the different functional units included in the CPU, and so the maximum pipeline length can vary in other ARMv8-based processors. [10] The A72's pipeline is, for the most part, the same as that of its predecessor, the Cortex-A57.

Names

To simplify the the pipeline model, its stages can be divided into 5 groups and named according to the functional units that perform them: [10]

- **Fetch**, during cycles 1-5.
- **Decode**, during cycles 6-12.
- **Issue**, in cycle 13.
- **Execute**, during the following cycles, depending on the required execution units; integer and branch operations take the shortest, at only one cycle.[2]
- **Retire**, or commit, in the last cycle.

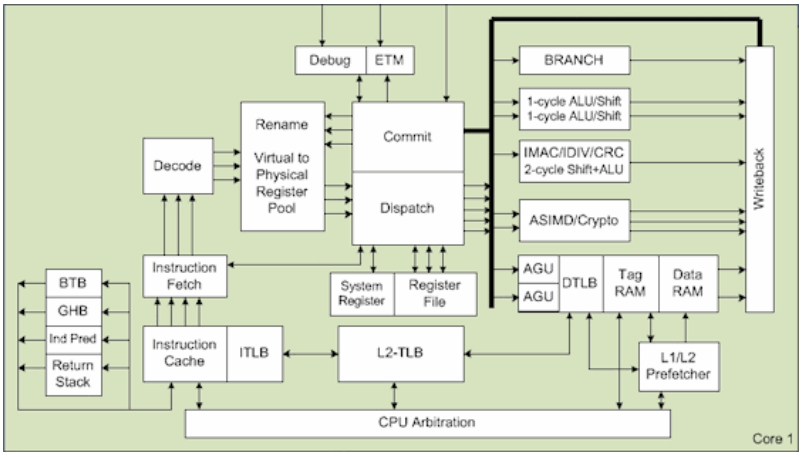

Figure 9: The structure of a Cortex-A72 processor core

[2]This stage is the only significant difference between the Cortex-A72's pipeline and the A57's, whose functional units exhibit deeper pipelines.

8 Steps of instruction execution

Fetch

Instruction execution begins as up to three instructions are fetched from the L1 instruction cache. For branch instructions, prediction also takes place during this stage, making use of the BTB and static predictor this architecture provides.

Decode

In this next stage, the previous instructions are decoded into micro-operations to be fed into the execution units, register renaming is performed, removing any potential WAW or WAR hazards, and up to five of the decoded micro-ops are dispatched to the issue queues; this stage could therefore be interpreted as three separate ones (decode, rename and dispatch), with the first taking four cycles, and the latter two taking two cycles each.

Performing register renaming is key in order to facilitate out-of-order execution in the following stages. It should also be noted that, prior to the renaming, the instructions operate on a virtual register pool, which helps to prevent the number of registers specified at the architectural level from becoming a bottleneck. [11]

Issue

Once they have entered the issue queue, the micro-operations are scheduled onto their respective execution units, up to eight at a time, corresponding to the eight execution units of this processor. From this point on, instruction execution is performed out-of-order.

Execute

This stage is different for each instruction, due to the different execution units that perform this step. Here it is worth noting that some of the more common operations (basic integer operations, vector operations, and load/stores, in this case) have two available execution units, in order to avoid structural hazards.

Retire

Finally, once all the micro-ops of a given instruction have finished executing, its result is written to a 128-entry re-order buffer, and then sent back to the dispatch unit to update the register files and commit the instructions in order.

9 Performance studies

The main contributors to the increased performance of the Cortex-A72 compared to the previous A57, from an architectural standpoint, are the new branch predictor, the increased decode/dispatch operation bandwidth, and modified execution units.

These changes contribute to more efficient use of the predictor units, as well as up to 20% more accurate predictions, a more continuous influx of micro-operations to the execution units, lower latencies for floating-point and SIMD operations, and up to 30% faster fetching of data from cache thanks to the combined L1/L2 prefetcher, among other improvements which could be seen in figures 2, 3, and 4 earlier. [3]

The performance of the Cortex-A72 can be studied further using a Raspberry Pi 4, as its system on chip, the Broadcom BCM2711, implements it as its only CPU. This makes it easier to isolate the results compared to other SoCs which include a second, less powerful CPU, following ARM's big.LITTLE design, often used in smartphones for the increased power efficiency that it can provide in that use case. Taking this into account, here are some benchmark results for the aforementioned system:

Linpack

A classic benchmark program for testing floating point performance, the results in figure 10 of the Linpack benchmark show the increase in performance, of nearly four times the amount of instructions per second, of the A72 compared to the Cortex-A53, used in the Raspberry Pi 3, despite only running 100 MHz faster: [12]

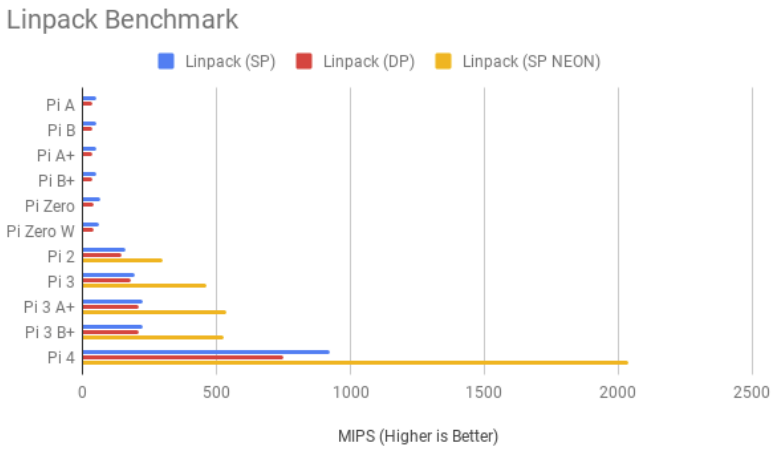

Figure 10: Linpack benchmark results for different models of Raspberry Pi

The previous figure also shows the progression in performance from the ARMv6-based ARM11 CPU of the first Raspberry Pi, through the ARMv7-based Cortex-A7 of the Pi 2.

SysBench

This benchmark software is based around multi-threaded prime number computation. In this case, as shown in figure 11, the A72 is faster than the A53 of the Raspberry Pi 3, but the margin is not as large as with Linpack, this time showing a performance improvement of about 36% when testing for prime numbers below 20000: [13]

Figure 11: Sysbench results for the Pi 4, Pi 3 and ASUS TinkerBoard

The ASUS TinkerBoard also shown in the graph, for reference, uses the ARMv7-based Cortex-A17 processor included in the Rockchip RK3288 SoC, running at 1.8 GHz, 400 MHz faster than the Pi 4's A72. The A17 was the last ARM-designed CPU using the ARMv7 architecture, and as seen here, can be superseded at lower clock speeds by the newer processors.

References

[1] Wikipedia. *ARM architecture* - Last access: 11/12/2019. Available at https://en.wikipedia.org/wiki/ARM_architecture

[2] *ARM® Cortex®-A72 MPCore Processor Technical Reference Manual* - Last access: 13/12/2019. Available at https://static.docs.arm.com/100095/0003/cortex_a72_mpcore_trm_100095_0003_05_en.pdf

[3] Robert Triggs. *A closer look at the ARM Cortex-A72* - Last access: 13/12/2019. Available at https://www.androidauthority.com/closer-look-arm-cortex-a72-603699/

[4] *The A64 instruction set* - Last access: 13/12/2019. Available at https://static.docs.arm.com/100898/0100/the_a64_Instruction_set_100898_0100.pdf

[5] *A64 - ARM* - Last access: 13/12/2019. Available at https://en.wikichip.org/wiki/arm/a64

[6] *Arm Developer - A32 instruction set* - last access: 12/12/2019. Available at https://developer.arm.com/architectures/instruction-sets/base-isas/a32

[7] *Arm Developer - A64 instruction set* - last access: 12/12/2019. Available at https://developer.arm.com/architectures/instruction-sets/base-isas/a64

[8] *Arm Developer - T32 instruction set* - last access: 12/12/2019. Available at https://developer.arm.com/architectures/instruction-sets/base-isas/t32

[9] *Arm Developer - T32 instruction set* - last access: 13/12/2019. Available at https://en.wikichip.org/wiki/mediatek/helio/mt6797

[10] Chris Shore. *ARMv8-A CPU Architecture Overview* - last access: 15/12/2019. Available at https://armkeil.blob.core.windows.net/developer/Files/pdf/graphics-and-multimedia/ARM_CPU_Architecture.pdf

[11] Shaked Flur, et al. *Modelling the ARMv8 Architecture, Operationally: Concurrency and ISA*. ACM SIGPLAN Notices, Vol. 51, No. 1. ACM, 2016.

[12] Gareth Halfacree. *Benchmarking the Raspberry Pi 4* - last access: 16/12/2019. available at https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-4-73e5afbcd54b

[13] *JEGX. Raspberry Pi 4 vs Raspberry Pi 3: CPU and GPU Benchmarks (Updated with TinkerBoard CPU test)* - last access: 16/12/2019. available at https://www.geeks3d.com/20190930/raspberry-pi-4-vs-raspberry-pi-3-cpu-and-gpu-benchmarks/