So, most mobile devices, iPhones and Androids run on ARM. Qualcomm, HUAWEI Kirin, Samsung Exynos and Apple A13/A14 Bionic are all ARM processors.

But on a computer, it’s not like that - x86 dominates there under the wing of Intel and AMD. That is why on the phone we cannot run Word from the computer.

x86 - this is the name given to the last digits of the family of classic Intel processors of the 70-80s.

How are they different?

There are two key differences.

The first is a set of instructions, that is, the language that the processor understands.

x86 processors use a complex instruction set called CISC - Complex Instruction Set Computing.

ARM processors, on the contrary, use a simplified instruction set - RISC - Reduced Instruction Set Computing.

By the way, ARM stands for Advanced RISC Machines

Sets of instructions are also called architecture or ISA - Instruction Set Architecture.

The second difference is the microarchitecture. What it is?

The language the processors speak also determines how they are designed. Because to execute each instruction on the processor, you need to place your own logical block. Accordingly, different instructions - different processor design. And design is microarchitecture.

x86 - CISC

ARM-RISC

So, remember. We say x86 - we mean the CISC architecture, ARM is RISC.

But how did it happen that the processors began to speak different languages?

It all started in the 1960s. At first, programmers worked with machine code, that is, they actually wrote zeros and ones. This quickly got everyone and Assembler appeared. A low-level programming language that allowed you to write simple commands such as add, copy, and so on. But programming in Assembler was also not easy. Because it was necessary literally “by the handle” to gradually describe to the processor each of its actions.

Therefore, if you were having dinner with the processor, and asked to pass it to you with salt, it would look like this:

Hey processor, look at the center of the table.

See the salt? Take her.

Now look at me.

Give me the salt. - Yeah thanks!

Now take the salt from me again.

Put it where you got it from

Thank you very much! Get on with your business.

Ahem... Processor, do you see the pepper?

Etc....

At some point, it all got the programmers sick. And they thought, Hey, why don't we just write a "Pass me the salt" instruction? So they did. A set of such complex instructions was called CISC.

This approach has become a real lifesaver for both developers and businesses. The client wanted a new instruction - no problem, if there was money - we will do it. But the clients had money.

Disadvantages of CISC

But was this approach optimal? From a developer's point of view, yes. But the microarchitecture suffered.

Imagine you have bought an apartment and now you need to furnish it with furniture. There is not enough space, every square meter counts. And just imagine, if a CISC processor furnished your living room with furniture, on the one hand, it would take care of the comfort of each potential visitor and would allocate its own personal place for him.

On the other hand, he would not spare the budget. A sofa for one person, a pouffe for another, a couch for another, a Game of Thrones throne for your Daenerys. In this case, the area of the room would end very quickly. To accommodate everyone, you would have to increase the budget and expand the hall. This is not rational. But most importantly, the CISC architecture has existed for a very long time, and the instructions that were written in the 60s are no longer relevant at all. Therefore, some of the furniture, or rather the execution units, simply will not be used. But many of them remain there. Therefore, RISC appeared …

Benefits of RISC

On the one hand, writing in Assembler for RISC processors is not very convenient. If we compare head-on code written for CISC and RISC processors, the advantage of the former is obvious.

This is how the code of the same operation looks like for x86 and ARM.

x86

MOV AX, 15; AH=00, AL=0Fh

AAA; AH=01, AL=05

RET

ARM

MOV-R3 #10

AND R2, R0, #0xF

CMP R2, R3

IT LT

BLT elsebranch

AD R2. #6

ADD R1. #one

elsebranch:

END

But it was like that before. Nobody writes in assembler for a long time. Now compilers do all this for programmers, so there are no difficulties with writing code for RISC processors. But there are advantages.

But these are all technical differences. There are also organizational differences. Have you ever wondered why there are so many manufacturers of processors for smartphones, and in the world of x86 PCs only AMD and Intel? It's simple - ARM is a licensing company, not a manufacturing company.

Even Apple had a hand in the development of ARM. Together with Acorn Computers and VLSI Technology. Apple has joined the alliance because of their upcoming device, the Newton. A device whose main function was text recognition.

Even you can start making your own processors by purchasing a license. But no one can produce x86 processors except for the blue and red companies. What does that mean? That's right, less competition, slower development. How did it happen?

OK. Let's say ARM does a great job with smartphones and tablets, but what about computers and servers, where the whole clearing is historically divided? And why Apple even rushed there with its Apple Silicon.

What about computers?

Ampere recently introduced its 80-core ARM processor. According to the manufacturer, in tests, the Ampere processor performs 4% better than AMD's fastest EPYC processor and consumes 14% less power.

Ampere is digging into the Cloud and Workstation segments and showing great numbers there. The world's fastest supercomputer today runs on ARM ISA. On the other hand, Intel is still trying to get into the low power segment and for this it releases a new interesting processor based on the lakefield microarchitecture.

So far, laptops and processors from Intel have one indisputable advantage - (cooling and) unity of architecture. While Qualcomm, Samsung, MediaTek exist on the ARM processor market, a monopoly is created in the x86 world and it is much easier for developers to make software and games for “adult” processors.

And Apple is the company that is able to motivate a sufficient number of developers to cut under their ARM. But the essence of this transition is rather not in the confrontation between CISC and RISC. As both approaches converge, the focus shifts to the microarchitecture that Apple makes for its mobile devices. And apparently their microarchitecture is cool. And they would like to use it in their computers.

And if Intel licensed x86 for money to other people, then probably Apple simply adapted their current microarchitecture for x86. But since they can't do that, they decided to just move to ARM. The problem for us with microarchitecture is that it's a trade secret. And we don't know anything about her.