How does the ARM architecture differ from x86? [closed]

Asked 11 years, 10 months ago Modified 2 years, 6 months ago

Viewed 288k times

265

Closed. This question does not meet Stack Overflow guidelines. It is not currently accepting answers.

This question does not appear to be about a specific programming problem, a software algorithm, or software tools primarily used by programmers. If you believe the question would be on-topic on another Stack Exchange site, you can leave a comment to explain where the question may be able to be answered.

Closed 9 years ago.

Improve this question

Is the x86 Architecture specially designed to work with a keyboard while ARM expects to be mobile? What are the key differences between the two?

x86 arm cpu-architecture

Share

Improve this question

edited May 28, 2022 at 3:04

Follow

Peter Cordes
361k ● 49 ● 699 ● 958

asked Feb 10, 2013 at 3:39

user1922878
2,853 ● 3 ● 14 ● 7

45 Unless the x86 has a ps/2 port I don't know about, it's no more built for keyboards than a pair of dirty underwear :-) – paxdiablo Feb 10, 2013 at 3:43

10 I think **keyboard** is referring to a typical PC role as opposed to the physical device. – artless-noise-bye-due2AI Feb 11, 2013 at 14:28

35 The x86 was not designed; It evolved on an island, with a strange bird that ate everthing that tried to pray on it. It now looks stranger than a duck billed platypus, and would not do well if a ship-full of new animals came along. – ctrl-alt-delor Dec 13, 2014 at 20:00

7 @richard - sadly, this happens to be the most historically accurate description of x86 i've ever seen. It says quite a lot about the industry. – Leeor Jan 4, 2015 at 18:48

6 @Leeor Sorry I made a small mistake in my comment, I said that the bird ate predators of the x86, where as it did not eat them, it sat on them. It is also worthy of note that the soft feathers of the bird where so very very very tidy. – ctrl-alt-delor Jan 4, 2015 at 20:14

5 Answers

Sorted by: Highest score (default)

ARM is a RISC (Reduced Instruction Set Computing) architecture while x86 is a CISC (Complex Instruction Set Computing) one.

401

The core difference between those in this aspect is that ARM instructions operate only on registers with a few instructions for loading and storing data from/to memory while x86 can use memory or register operands with ALU instructions, sometimes getting the same work done in fewer instructions. Sometimes more because ARM has its own useful tricks like loading a pair of registers in one instruction, or using a shifted register as part of another operation. Up until ARMv8 / AArch64, ARM was a native 32 bit architecture, favoring four byte operations over others.

So ARM is a simpler architecture, leading to small silicon area and lots of power save features while x86 becomes a power beast in terms of both power consumption and production.

To answer your question "*Is the x86 Architecture specially designed to work with a keyboard while ARM expects to be mobile?*". x86 isn't specially designed to work with a keyboard just like ARM isn't designed specifically for mobile. However, again because of the core architectural choices, x86 also has instructions to work directly with a separate IO address space, while ARM does not. Instead, ARM uses memory-mapped IO for everything, including reading/writing PCI IO space. (Which is rarely needed with modern devices because it's slow on x86. e.g. modern USB controllers, so accessing USB-connected devices is as efficient as the USB controller makes it.)

If you need a document to quote, this is what Cortex-A Series Programmers Guide (4.0) tells about differences between RISC and CISC architectures:

> An ARM processor is a Reduced Instruction Set Computer (RISC) processor.
>
> Complex Instruction Set Computer (CISC) processors, like the x86, have a rich instruction set capable of doing complex things with a single instruction. Such processors often have significant amounts of internal logic that decode machine instructions to sequences of internal operations (microcode).
>
> RISC architectures, in contrast, have a smaller number of more general purpose instructions, that might be executed with significantly fewer transistors, making the silicon cheaper and more power efficient. Like other RISC architectures, ARM cores have a large number of general-purpose registers and many instructions execute in a single cycle. It has simple addressing modes, where all load/store addresses can be determined from register contents and instruction fields.

ARM company also provides a paper titled Architectures, Processors, and Devices Development Article describing how those terms apply to their business.

An example comparing instruction set architecture:

For example if you would need some sort of bytewise memory comparison block in your application (generated by compiler, skipping details), this is how it might look like on x86, if optimizing for code-size over speed. (`rep movsb` / `rep stosb` are fast-ish on modern CPUs, the conditional-rep comparison instructions aren't.)

```
repe cmpsb          /* repeat while equal compare
string bytewise */
```

while on ARM shortest form might look like (without error checking or optimization for comparing multiple bytes at once etc.)

```
top:
ldrb r2, [r0, #1]! /* load a byte from address in
r0 into r2, increment r0 after */
ldrb r3, [r1, #1]! /* load a byte from address in
r1 into r3, increment r1 after */
subs r2, r3, r2    /* subtract r2 from r3 and put
result into r2      */
beq  top           /* branch(/jump) if result is
zero                 */
```

which should give you a hint on how RISC and CISC instruction sets differ in complexity. Interestingly, x86 does not have write-back addressing modes (that load and increment the pointer) *except* via its "string" instructions like `lodsd`.

 edited May 28, 2022 at 2:55

Follow

Peter Cordes
361k ● 49 ● 699 ● 958

answered Feb 10, 2013 at 7:05

auselen
28.1k ● 8 ● 76 ● 117

12 ARMv8-A has a 64-bit architecture called AArch64. – remmy Nov 23, 2013 at 20:58

15 Although the x86 has some very powerful instructions, the arm can still beat it in a fight (if both have same clock speed). This is partly because the arm has a good set of registers, where as the x86 spends 1/2 of its time moving data in and out of its limited set of registers (this is less true of x86-64, is it has more registers). And partly because the Arm's simplicity leaves room for a bigger cache, and has all instructions conditional (making cache misses fewer). And arm's move multiple instruction (the only non RISC instruction), allows it to move data quickly. – ctrl-alt-delor Jan 4, 2015 at 20:24

6 I could write ARM code faster, Though bigger, by using more registers. If I look at this implementation the x86 takes 5+9×N clocks, the ARM takes 4×N clocks (both figures are for no cache misses). The x86 scores better for instruction bytes on this example: x86 = 2 bytes, arm = 16 bytes. ARM scores much better on this metric in more realistic tests, e.g on exiting loop r2 will have information on if strings are equal / which is bigger, so will condition codes. The arm can run other instructions before checking condition codes. Arm does not have to branch when checking condition codes.
– ctrl-alt-delor Jan 4, 2015 at 20:56

3 @JeremyFelix It looks like this stackoverflow.com/questions/13106297/… There are different pipes for different type of instructions, even there are duplicated ones. CPU divides instructions into micro

instructions and those can run in parallel among pipeline. – auselen Dec 17, 2015 at 12:31

4 You say “while x86 can operate on directly memory as well.” however for the x86 (pre x86-64), it have so few registers that there was no “as well”, you had to store everything in memory; about ½ of instructions in a program where just to move things about. Whereas in ARM very few instructions are needed to move data about. – ctrl-alt-delor Feb 17, 2016 at 9:48

115

Neither has anything specific to keyboard or mobile, other than the fact that for years ARM has had a pretty substantial advantage in terms of power consumption, which made it attractive for all sorts of battery operated devices.

As far as the actual differences: ARM has more registers, supported predication for most instructions long before Intel added it, and has long incorporated all sorts of techniques (call them "tricks", if you prefer) to save power almost everywhere it could.

There's also a considerable difference in how the two encode instructions. Intel uses a fairly complex variable-length encoding in which an instruction can occupy anywhere from 1 up to 15 byte. This allows programs to be quite small, but makes instruction decoding relatively difficult (as in: decoding instructions fast in parallel is more like a complete nightmare).

ARM has two different instruction encoding modes: ARM and THUMB. In ARM mode, you get access to all instructions, and the encoding is extremely simple and fast to decode. Unfortunately, ARM mode code tends to be fairly large, so it's fairly common for a program to occupy around twice as much memory as Intel code would. Thumb mode attempts to mitigate that. It still uses quite a regular instruction encoding, but reduces most instructions from 32 bits to 16 bits, such as by reducing the number of registers, eliminating predication from most instructions, and reducing the range of branches. At least in my experience, this still doesn't usually give *quite* as dense of coding as x86 code can get, but it's fairly close, and decoding is still fairly simple and straightforward. Lower code density means you generally need at least a little more memory and (generally more seriously) a larger cache to get equivalent performance.

At one time Intel put a lot more emphasis on speed than power consumption. They started emphasizing power consumption primarily on the context of laptops. For laptops their typical power goal was on the order of 6 watts for a fairly small laptop. More recently (*much* more recently) they've started to target mobile devices (phones, tablets, etc.) For this market, they're looking at a couple of watts or so at most. They seem to be doing pretty well at that, though their approach has been substantially different from ARM's, emphasizing fabrication technology where ARM has mostly emphasized micro-architecture (not surprising, considering that ARM sells designs, and leaves fabrication to others).

Depending on the situation, a CPU's energy consumption is often more important than its power consumption though. At least as I'm using the terms, power consumption refers to power usage on a (more or less) instantaneous basis. Energy consumption, however, normalizes for speed, so if (for example) CPU A consumes 1 watt for 2 seconds to do a job, and CPU B consumes 2 watts for 1 second to do the same job, both CPUs consume the same total amount of energy (two watt seconds) to do that job--but with CPU B, you get results twice as fast.

ARM processors tend to do very well in terms of power consumption. So if you need something that needs a processor's "presence" almost constantly, but isn't really doing much work, they can work out pretty well. For example, if you're doing video conferencing, you gather a few milliseconds of data, compress it, send it, receive data from others, decompress it, play it back, and repeat. Even a really fast processor can't spend much time sleeping, so for tasks like this, ARM does really well.

Intel's processors (especially their Atom processors, which are actually intended for low power applications) are extremely competitive in terms of energy consumption. While they're running close to their full speed, they will consume more power than most ARM processors--but they also finish work quickly, so they can go back to sleep sooner. As a result, they can combine good battery life with good performance.

So, when comparing the two, you have to be careful about what you measure, to be sure that it reflects what you honestly care about. ARM does very well at power consumption, but depending on the situation you may easily care more about energy consumption than instantaneous power consumption.

Share Improve this answer Follow

edited Jun 23, 2020 at 8:26

answered Feb 10, 2013 at 3:50

Jerry Coffin

489k ● 83 ● 647 ● 1.1k

2 that is why ? RISC needs more RAM, whereas CISC has an emphasis on smaller code size and uses less RAM overall than RISC – Waqar Naeem Sep 21, 2019 at 6:19

1 Thumb mode (variable length allowing short encodings) isn't a *difference*; that's how x86 always works (but moreso, with instruction length varying from 1 to 15 bytes, and much harder to decode than Thumb2). ARM mode (fixed width encoding with 3-operand non-destructive instructions) is the difference from x86! – Peter Cordes Jun 23, 2020 at 6:33

1 *Having a lot faster processor isn't a big help* - video conferencing might be a better example: low latency means you can't just do a burst of decoding into a decent sized buffer and and go back into a deep or medium level sleep state. "Race to sleep" is a key concept in energy consumption for a fixed amount of computation, given that modern CPUs can save significant power when fully idle (clock stopped, or even powering down parts of the core. Or in deeper sleeps, also caches after write-back.) ... and that's

the point you make in the next paragraph, of course. >.< – Peter Cordes Jun 23, 2020 at 6:38

@PeterCordes: Thumb Mode encoding isn't much like x86 encoding. Although it's not *quite* as regular as ARM encoding, it's still pretty much fixed format.Density increase is largely from eliminating bits that are simply rarely used in ARM encoding. For example, virtually all ARM instructions are conditional, but conditions are only used a fairly small percentage of the time (so most non-branch THUMB instructions are unconditional). – Jerry Coffin Jun 23, 2020 at 6:42

@PeterCordes: You're right: video conferencing is a better example--I've edited that in. Thank you. – Jerry Coffin Jun 23, 2020 at 6:46

43

Additional to Jerry Coffin's first paragraph. Ie, ARM design gives lower power consumption.

The company `ARM`, only licenses the CPU technology. They don't make physical chips. This allows other companies to add various peripheral technologies, typically called **SOC** or system-on-chip. Whether the device is a tablet, a cell phone, or an in-car entertainment system. This allows chip vendors to tailor the rest of the chip to a particular application. This has additional benefits,

1. Lower board cost
2. Lower power (note1)
3. Easier manufacture

4. Smaller form factor

`ARM` supports SOC vendors with AMBA, allowing SOC implementers to purchase off the shelf 3rd party modules; like an Ethernet, memory and interrupt controllers. Some other CPU platforms support this, like MIPS, but MIPS is not as power conscious.

All of these are beneficial to a handheld/battery operated design. Some are just good all around. As well, `ARM` has a history of battery operated devices; Apple Newton, Psion Organizers. The PDA software infra-structure was leveraged by some companies to create **smart phone** type devices. Although, more success was had by those who re-invented the GUI for use with a **smart phone**.

The rise of `Open source` tool sets and `operating systems` also facilitated the various `SOC` chips. A closed organization would have issues trying to support all the various devices available for the ARM. The two most popular cellular platforms, Andriod and OSx/IOS, are based up Linux and FreeBSD, Mach and NetBSD os's. `Open Source` helps `SOC` vendors provide software support for their chip sets.

Hopefully, why **x86** is used for the *keyboard* is self-evident. It has the software, and more importantly people trained to use that software. Netwinder is one `ARM` system that was originally designed for the *keyboard*. Also, manufacturer's are currently looking at ARM64 for the server market. Power/heat is a concern at 24/7 data centers.

So I would say that the **ecosystem** that grows around these chips is as important as features like low power consumption. `ARM` has been striving for low power, higher performance computing for some time (mid to late 1980's) and they have a lot of people on board.

Note1: Multiple chips need bus drivers to inter-communicate at known voltages and drive. Also, typically separate chips need support capacitors and other power components which can be shared in an **SOC** system.

Share Improve this answer

Follow

edited May 23, 2017 at 11:54

Community Bot

1 ● 1

answered Feb 11, 2013 at 14:54

artless-noise-bye-due2AI

22.4k ● 6 ● 73 ● 109

35

The ARM is like an Italian sports car:

- Well balanced, well tuned, engine. Gives good acceleration, and top speed.
- Excellent chases, brakes and suspension. Can stop quickly, can corner without slowing down.

The x86 is like an American muscle car:

- Big engine, big fuel pump. Gives excellent top speed, and acceleration, but uses a lot of fuel.

- Dreadful brakes, you need to put an appointment in your diary, if you want to slowdown.
- Terrible steering, you have to slow down to corner.

In summary: the x86 is based on a design from 1974 and is good in a straight line (but uses a lot of fuel). The arm uses little fuel, does not slowdown for corners (branches).

Metaphor over, here are some real differences.

- Arm has more registers.
- Arm has few special purpose registers, x86 is all special purpose registers (so less moving stuff around).
- Arm has few memory access commands, only load/store register.
- Arm is internally Harvard architecture my design.
- Arm is simple and fast.
- Arm instructions are architecturally single cycle (except load/store multiple).
- Arm instructions often do more than one thing (in a single cycle).
- Where more that one Arm instruction is needed, such as the x86's looping store & auto-increment, the Arm still does it in less clock cycles.
- Arm has more conditional instructions.

- Arm's branch predictor is trivially simple (if unconditional or backwards then assume branch, else assume not-branch), and performs better that the very very very complex one in the x86 (there is not enough space here to explain it, not that I could).
- Arm has a simple consistent instruction set (you could compile by hand, and learn the instruction set quickly).

Share Improve this answer edited Dec 29, 2018 at 16:58

Follow

answered Aug 19, 2015 at 19:12

ctrl-alt-delor

7,725 ● 5 ● 44 ● 54

12 This analogy breaks at the fact that Italian sports cars break down at every instant they can get while ARM CPUs don't, and that while it could be easily done, you can't actually *buy* a single ARM CPU that can do desktop CPU speeds, let alone socketed ones and mainboards to put them in. :)
– anon Jan 13, 2016 at 21:48

2 Performance wise it competes directly with some of the biggest / faster Xeon processors (eg E5-2690 v3) but at lower power, cost. quora.com/… – ctrl-alt-delor Jan 14, 2016 at 21:39

2 For massively parallel workloads like databases and I/O servers, sure. For single-threaded performance, nobody's designed an ARM core anywhere near as big as x86. No reason they couldn't, just nobody has. The "x86 tax" on power and die area is not that large compared to the amount of silicon used for the out-of-order machinery in high-power

CPU cores. There are certainly warts in x86, but RISC has a code-density disadvantage (which doesn't usually matter much, but it still matters). This gets argued repeatedly on realworldtech.com forums. – Peter Cordes Feb 24, 2016 at 5:19

2 @richard: There's a lot of stuff you don't "need", but that increases code density. The trick is balancing decode complexity against code size / number of instructions. Increasing the width of an out-of-order core is extremely expensive in power consumption, so packing more work into each instruction is valuable. A small increase in decode complexity is much cheaper. Modern x86 CPUs already manage to decode x86 quickly. (Not quite quickly enough to keep a 4-wide OOO core fed from the decoders instead of uop-cache or loop buffer, and of course at a high power cost.) – Peter Cordes Feb 26, 2016 at 21:53

4 @Evi1M4chine, it also breaks at the fact that an Italian sports car is hugely expensive, while an America muscle car is relatively cheap. And the muscle car is what it is because it is simple, while something like a Ferrari is very very complicated. Quite the opposite of CISC vs. RISC – Lorenzo Dematté Jul 28, 2016 at 15:29

18

The ARM architecture was originally designed for Acorn personal computers (See Acorn Archimedes, circa 1987, and RiscPC), which were just as much keyboard-based personal computers as were x86 based IBM PC models. Only later ARM implementations were primarily targeted at the mobile and embedded market segment.

Originally, simple RISC CPUs of roughly equivalent performance could be designed by much smaller

engineering teams (see Berkeley RISC) than those working on the x86 development at Intel.

But, nowadays, the fastest ARM chips have very complex multi-issue out-of-order instruction dispatch units designed by large engineering teams, and x86 cores may have something like a RISC core fed by an instruction translation unit.

So, any current differences between the two architectures are more related to the specific market needs of the product niches that the development teams are targeting. (Random opinion: ARM probably makes more in license fees from embedded applications that tend to be far more power and cost constrained. And Intel needs to maintain a performance edge in PCs and servers for their profit margins. Thus you see differing implementation optimizations.)

edited Mar 8, 2017 at 18:45

answered Sep 23, 2013 at 5:37

hotpaw2
70.6k ● 15 ● 92 ● 155

1 There are still massive architectural differences. However intel have done a wonderful job and invested a shed load of money, to make there poorly architected CPU run very well (one wonders what could have been done, if all this effort was put into a well architected CPU). – ctrl-alt-delor Dec 29, 2018 at 16:41