Twitter Thread by foone

foone
@Foone

you know how Atari called the Jaguar as a "64bit system", because it had two 32bit CPUs? (or because it had a 64bit bus, but let's not worry about that)
we should apply that logic to the current generation of consoles.
So the Switch is a 256bit console. It's the Nintendo 256.

The Xbox Series X? terrible name.
Better call it the "Xbox 512 Blu-ray".
like the Jaguar CD, the Sega CD... the good old days when consoles had their media type in the name.

Boringly the PS5 is also a Playstation 512 Blu-ray, because both of them are using the same AMD Zen 2 CPU.

The Switch one is weeeird. It's built on an Nvidia Tegra X1, which is an 8 core 64-bit CPU, so that should be 512, right?

Well... no. Four of them can't be used. And later documentation doesn't even mention them.

it's some weird system where the system-on-a-chip has four Cortex-A57 cores and four Cortex-A53 cores but only the A57s are usable?

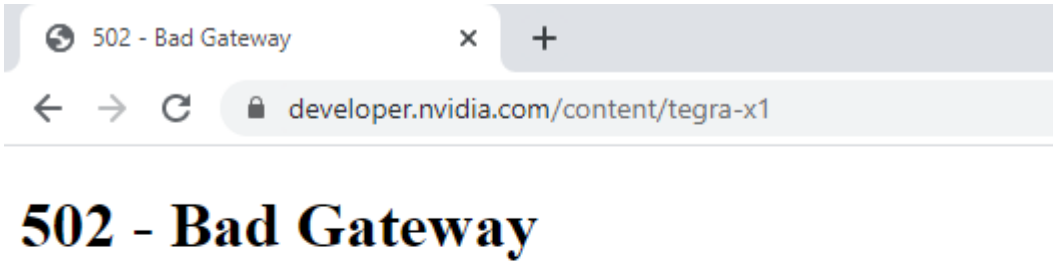
and it's not like "oh, games use the A57s and the A53s are used for low-powered things like the OS", because games actually only run on three of the A57s, with one reserved for the OS!
So you could instead argue the switch is a 192-bit system, by that logic.

the implication seems to be that Nvidia designed the system-on-chip to be 8-core, with 4 main cores and 4 secondary cores, as effectively two processors.
Then they discovered the secondary cores don't work properly, but shipped it anyway.

The later TM660M model of the chip just says it has 4 cores, the A57s... but it's possible the A53s are still on there too, just as dead silicon

I tried to look up the official Tegra X1 page and... huh.

NVIDIA DID YOU BREAK YOUR CPU SO BAD THAT ONLY HALF OF IT WORKS, THEN HOST YOUR WEBSITE ON IT?



anyway I did find one mention that the chip can't power both sets of cores at once.
so presumably this setup was originally designed for things like phones, where sometimes you want MAX PERFORMANCE but most of the time you just want power efficiency?

so even if it was functionally designed as 8 separate cores, it can only be used as the 4 performance cores or the 4 efficiency cores.

so it's like a turbo switch

(no nintendo pun intended.)

it's just a broken turbo switch that's stuck on

this is ARM's big.LITTLE system.
fun fact: some of the implementations of this technology are operating system transparent: the code running on it doesn't even need to know it's running on 8 cores instead of 4.

and the CPU can just transparently switch between running on the fast CPUs and the low CPUs, based on things like temperature and battery power. The OS doesn't need to know.

Then there's a version where the OS knows there's different cores (or at least that it can switch performance modes) and can toggle them as needed, per-core.
So you can have an 8 core system (4 fast 4 slow) and use any 4 of them at once, depending on demand.

then some CPUs have all cores functional at once, but the OS knows that some are faster than others, and therefore can intelligently assign threads to fast or slow cores as needed.
The Apple A-series chips, for example, work this way.

ARM has developed a successor to big.LITTLE, called DynamIQ.
big.LITTLE works on clusters, where each cluster is a grouping of identical cores. DynamIQ makes it so that clusters can be heterogenous core collections, as well as supporting more of them

so you can have 8 cores in a cluster and up to 32 clusters.
I can't wait for the Xbox Aleph null which deploys this configuration.
Eight 64bit ARM A57 clusters, 32 of them, for a 16384bit CPU

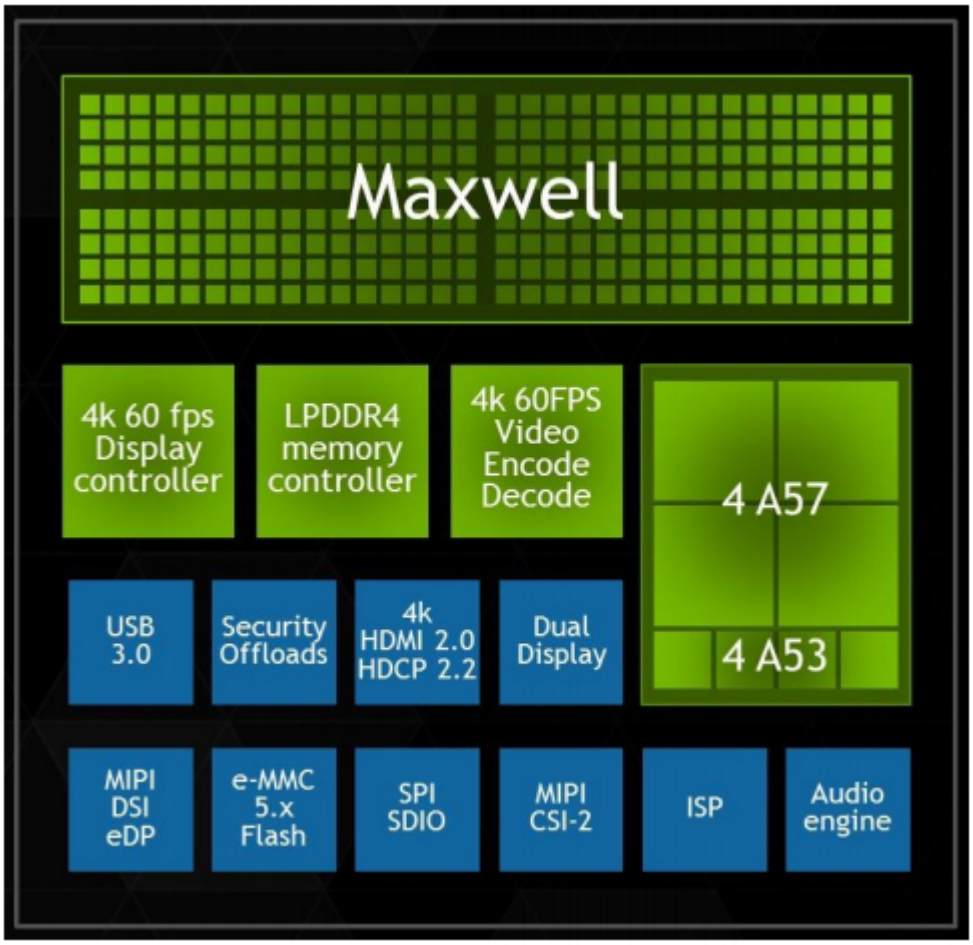
the die is 8 inches across, has a yield of 3 per fortnight, and requires active liquid helium cooling

BTW, if you accept the "external bus width = consoles bit width" thing to explain why the Atari Jaguar is 64bit despite having 32bit CPUs, it means the IBM PC/XT isn't a 16bit architecture despite using a 16bit processor.
It's an 8-bit system!

Because the IBM PC & IBM XT were built on the Intel 8088 version of the Intel 8086 chip. Both are 16-bit internally, but the 8088 uses an external 8-bit bus to make it cheaper to interface with.

on the other hand, if you're going by the "add all the bits together" method, the IBM PC is 24bit, as long as you have the keyboard plugged in.
(the 8051 in the keyboard is 8-bit)

anyway now that the tegra x1 page loads you can see that they still show the A53s on the architecture image. They're just tiny and hidden under the A57s:



the idea for making them more efficient is that they're effectively the same CPUs, functionally, but implemented differently.
Modern CPUs have a lot of transistors devoted to being clever so they can run faster.

and the power use of a given processor increases as you have more toggling transistors and more leaking transistors.
So by having a smaller, simpler implementation of a functionality-identical processor, you can save on power.

like processors do things like out-of-order-execution, where they figure out which of the instructions they're about to execute can be executed at the same time, and re-order them to allow simultaneous execution on different functional units.

like a CPU might have separate silicon for adding integers and multiplying floating point numbers, and if the instructions it needs to run are:
1+1
2+2
3.1*2.3
5.9*0.5

a naïve CPU will spend 4 cycles on this task:
add the integer, add the other integer, multiply the floating point, multiply the other floating point.
The other part of the CPU is just dark when one part is being used.

and a smarter, more pipelined CPU, might realize that 4 cycles is longer than is needed:
You can do the second step and the third step at the same time.
so it adds 1+1, then on the next cycle it does the 2+2 at the same time as the 3.1*2.3, then does the last multiplication next

but an even smarter CPU can figure out that none of these operations depend on each other, so you can do them in any order you want. So you can do it in two cycles

and this sort of cleverness is a lot of how modern CPUs are so fast: they figure out tricks like this to improve execution speed.
But it's not free: You need more silicon to figure out when you can do this, when you can't, and to actually do the re-ordering.

and on desktop all that means is that your CPU uses a bit more power and was a bit more complicated to design (so it might have more faults...) but on mobile that means your CPU is fast but uses more battery

and the fun thing about how CPU power usage is based on two things:
1. transistors toggling
2. total transistors (because of leakage)

you can't just downclock your CPU for efficiency savings

running your CPU at a lower clock speed means you save power on transistor-toggling energy usage, but the transistor leakage remains the same. it's based on how many transistors you have powered, not how many you have in active use.

that's why systems like this are designed with a big-fast core and a small-slow core.
You can completely de-power the big-fast core when you're in battery-savings mode, and then there's no leakage from it.

BTW, what I mean about "depend on each other": sometimes you can't reorder operations.
Like if your instructions were:
1. set X to 1
2. add 5 to X
3. multiply X by 2

You don't want your CPU to decide that it can put step 3 before step 2, and instead of (1+5)*2 you get (1*2)+5, because those are different numbers.

so the CPU has to have a way of figure out when the instructions depend on each other and when they don't.
And that functionality means more transistors devoted to doing it.

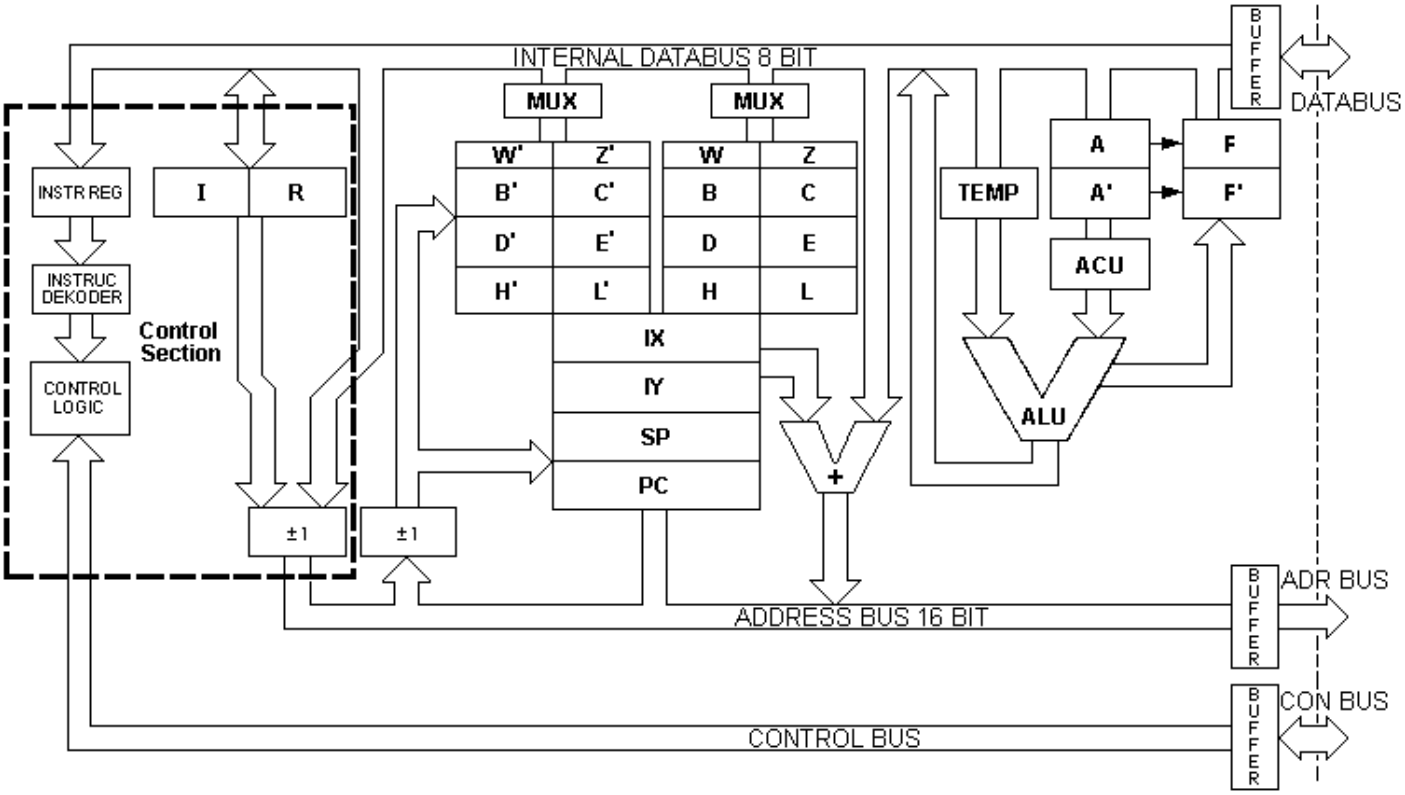
Whereas a simple architecture that just does all instructions in-order might be a lot slower in practice, it's much smaller in terms of transistors needed, so it's more efficient.

you can also change things like cache size (a bigger cache can improve average memory access time a lot, but that's a bunch of transistors to implement it), since a smaller cache will fallback to just pulling from main RAM (or higher cache levels) more often.

another that can be varied is execution units.
Like, CPUs spend a lot of time doing simple integer math, right?

So performance CPUs tend not to just have a big "integer math unit" which does the adding and subtracting and such

like on this diagram of the relatively simple Z80 CPU, that's the ALU: the "Arithmetic logic unit", which is a black box that does adding/subtracting.



instead, CPU cores can just have several of them, and therefore run them in parallel.
Like the Intel NetBurst architecture (used in the Pentium 4) has two separate ALUs, and they're clocked twice as fast as the main CPU.

so if your code is doing a lot of number crunching, you get much faster execution by having two ALUs (and two double-speed ALUs, at that)

But functionally, a single-ALU design works the same... just slower.

so maybe your fast-version of the chip has 2 or 4 ALUs, and the slow-version just has one.

another fun thing processors do to speed up execution is
branch prediction.
So you've got some code like:
1: add 1 to X
2: if X is over 10, abort
3: go back to 1

now the CPU does pipelining (where it has several instructions in process at once) and out of order execution, but a conditional operation puts a big wedge in that functionality.

The CPU doesn't know if it's going to do the abort step, so it doesn't know what happens after step 2