Computer Organization

Micro-Architecture

x86, Apple

Dr Jeff Drobman

website ➡ *drjeffsoftware.com/classroom.html*

email ➡ *jeffrey.drobman@csun.edu*

Index

Section

RISC vs CISC

RISC vs CISC

EXTREMETECH

The problem with using RISC versus CISC as a lens for comparing modern x86 versus ARM CPUs is that it takes three specific attributes that matter to the x86 versus ARM comparison — process node, microarchitecture, and ISA — crushes them down to one, and then declares ARM superior on the basis of ISA alone. "ISA-centric" versus "implementation-centric" is a better way of understanding the topic, provided one remembers that there's a Venn diagram of agreed-upon important factors between the two. Specifically:

The ISA-centric argument acknowledges that manufacturing geometry and microarchitecture are important and were historically responsible for x86's dominance of the PC, server, and HPC market. This view holds that when the advantages of manufacturing prowess and install base are controlled for or nullified, RISC — and by extension, ARM CPUs — will typically prove superior to x86 CPUs.

The implementation-centric argument acknowledges that ISA can and does matter, but that historically, microarchitecture and process geometry have mattered more. Intel is still recovering from some of the worst delays in the company's history. AMD is still working to improve Ryzen, especially in mobile. Historically, both x86 manufacturers have demonstrated an ability to compete effectively against RISC CPU manufacturers.

RISC vs CISC

Quora

Why are RISC processors considered faster than CISC processors?

Bob Colwell, former Ex-Intel Chief X86 Architect, Ex-DARPA/MTO Directo

Answered September 26, 2018 · Upvoted by Ed Bell, 35+ years experience supporting

Intel's x86's do NOT have a RISC engine "under the hood." They implement the x86 instruction set architecture via a decode/execution scheme relying on mapping the x86 instructions into machine operations, or sequences of machine operations for complex instructions, and those operations then find their way through the microarchitecture, obeying various rules about data dependencies and ultimately time-sequencing. The "micro-ops" that perform this feat are over 100 bits wide, carry all sorts of odd information, cannot be directly generated by a compiler, are not necessarily single cycle. But most of all, they are a microarchitecture artifice — RISC/CISC is about the instruction set architecture.

Microarchitectures are about pipelines, branch prediction, ld/st prediction, register renaming, speculation, misprediction recovery, and so on. All of these things are orthogonal to what instructions you put into your ISA.

RISC vs CISC

Micro-Ops

Quora

Why are RISC processors considered faster than CISC processors?

Bob Colwell, former Ex-Intel Chief X86 Architect, Ex-DARPA/MTO Directo

Answered September 26, 2018 · Upvoted by Ed Bell, 35+ years experience supporting

There can be real consequences to mentally blurring the lines between architecture and microarchitecture. I think that's how some of the not-so-good ideas from the early RISC work came into existence: register windows and branch shadows, for example. Microarchitecture is about performance of this chip that I'm designing right now. Architecture (adding new instructions, for example) is about what new baggage I'm going to inflict on designers of compatible future chips and those writing compilers for them.

The micro-op idea was not "RISC-inspired", "RISC-like", or related to RISC at all. It was our design team finding a way to break the complexity of a very elaborate instruction set away from the microarchitecture opportunities and constraints present in a competitive microprocessor.

RISC Goals

Quora

Jeff Drobman · 20h ago

Joe, I teach that RISC has these specific design principles: Load-Store, large GR set, which support single-cycle execution.

Upvote Reply ooo

Joe Zbiciak · 13h ago

Single cycle throughput and large orthogonal general purpose register file are ideals RISC CPUs aspired to. Single cycle *latency* for arithmetic was also a goal, but some operations simply take more than one cycle.

Every commercial RISC seems to tarnish the ideal in some way. MIPS with their bolt-on multiplier. ARM with load/store multiple and shifter on src2. SPARC register windows.

Jerry Coffin and I have gone back and forth on RISC in theory vs. practice. He's more or less convinced me that any list of properties you state for RISC is likely violated in some way by a processor that claims to be RISC.

Many of the early RISC properties, such as eliminate pipeline interlocks, expose delay slots, and let the compiler fix everything, have been abandoned. Even fixed length encoding... RISC-V has an extension for variable length encodings from 16 bits to 192 bits, IIRC.

CISC vs RISC Performance

- CISC → *CPI* = ~5-9 (typ)
- RISC → *CPI* = ~1.4 (typ) → 5X faster

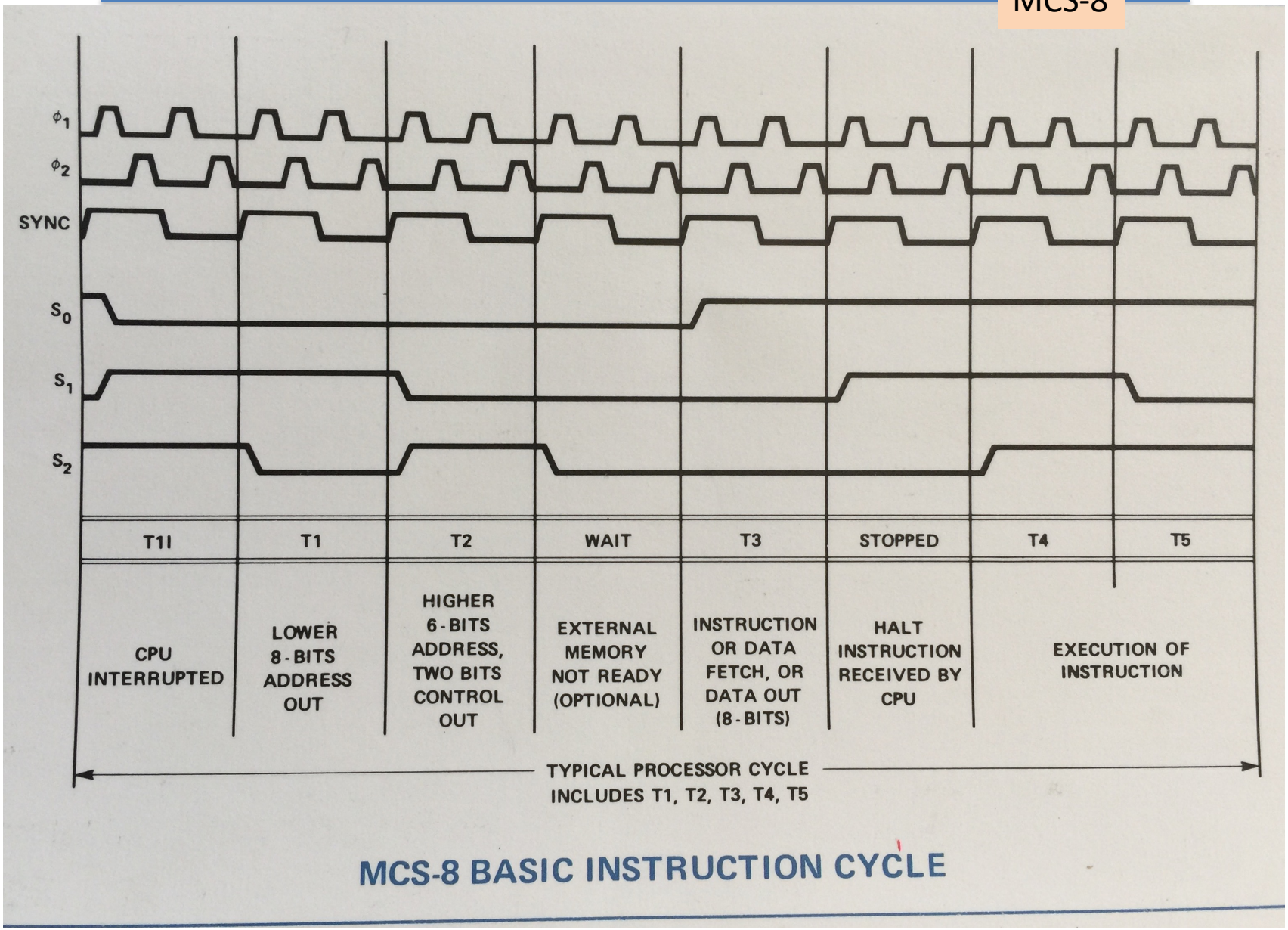
Single core, single pipeline
(no instruction level parallelism)

Single-cycle execution → +Delays for Load, Branch

- Pipeline architecture
- Memory access limited (Load-Store)

CISC Instruction Cycle

MCS-8

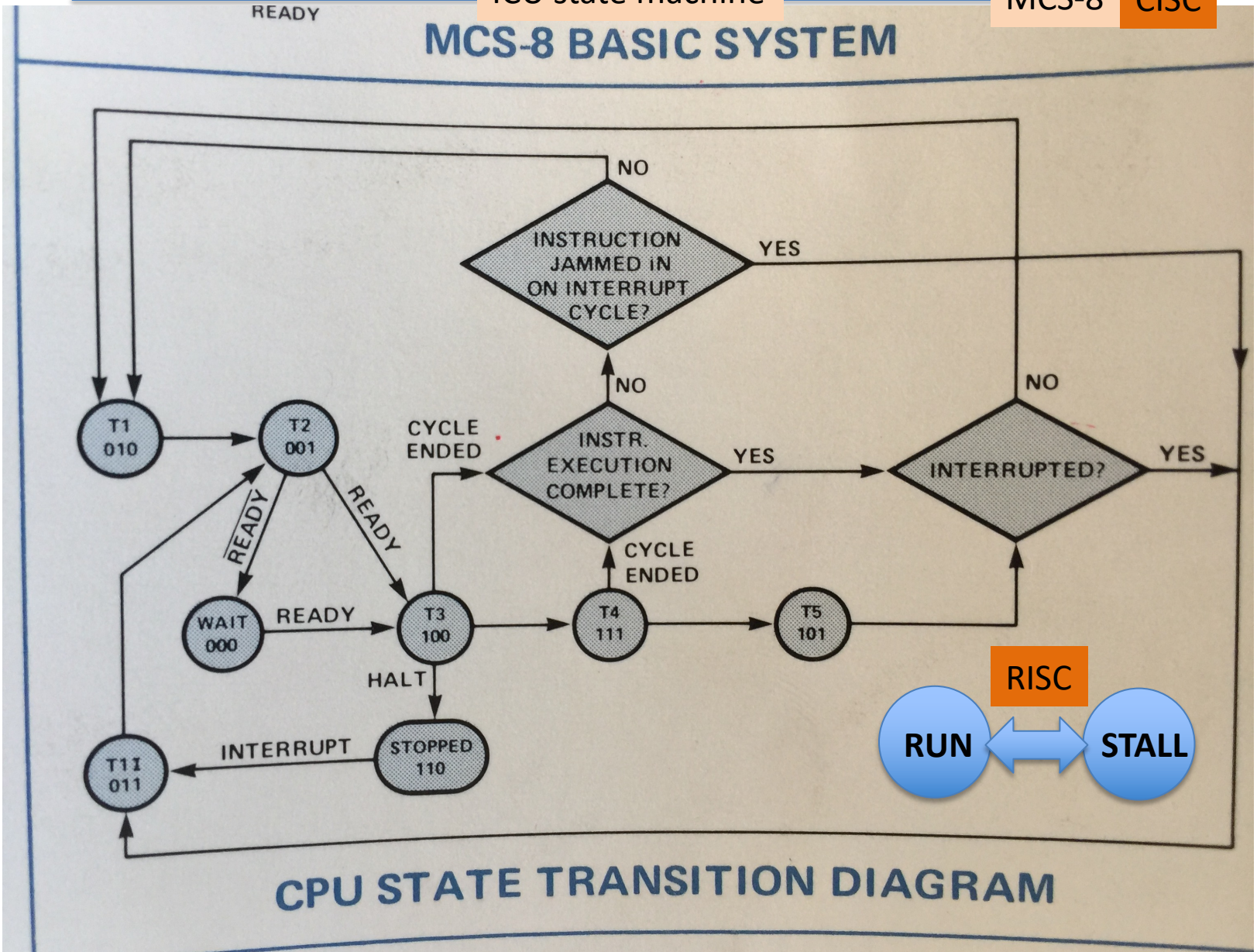


MCS-8 BASIC INSTRUCTION CYCLE

CISC State Diagram

ICU state machine

MCS-8 CISC

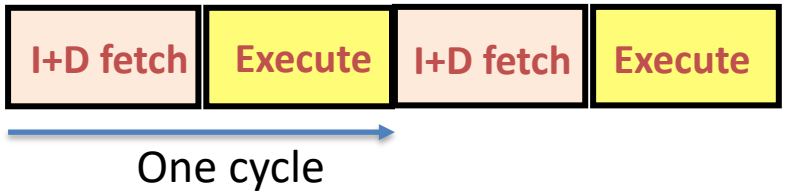


CISC/RISC Pipelines

Non pipelined

4-8 cycles per I

I8008/M6800

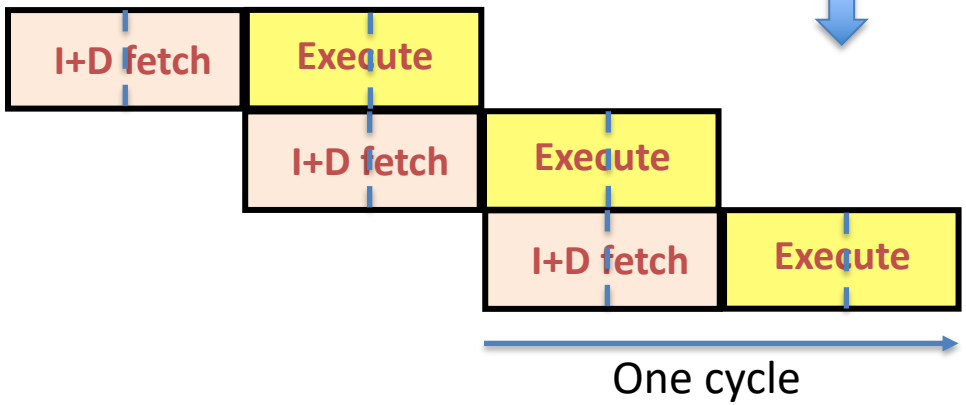


CISC Pipeline

2-4 cycles per I

i8088/M68000

2-stage



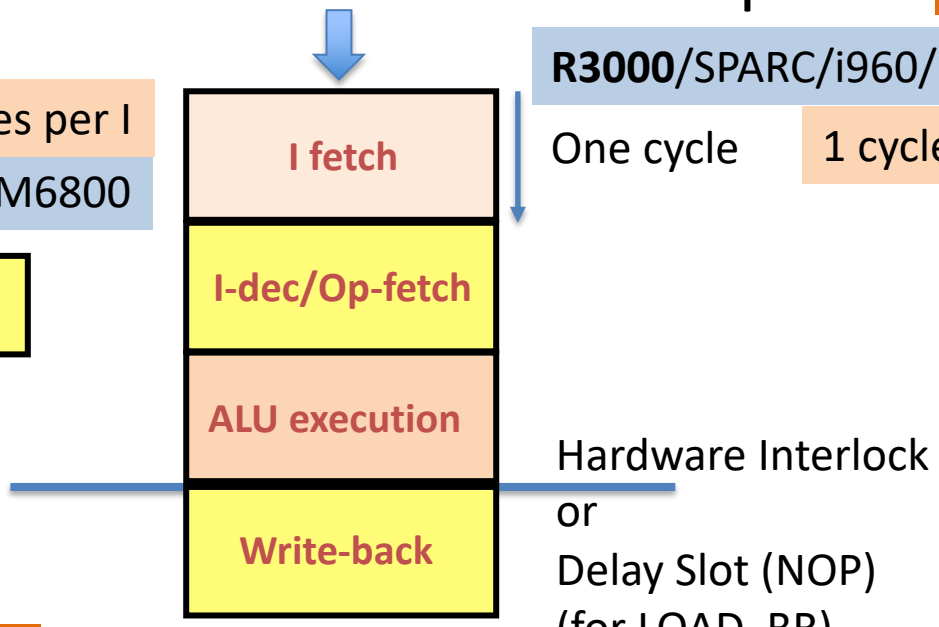
RISC Pipeline

4/5-stage

R3000/SPARC/i960/29K/PPC

One cycle

1 cycle per I

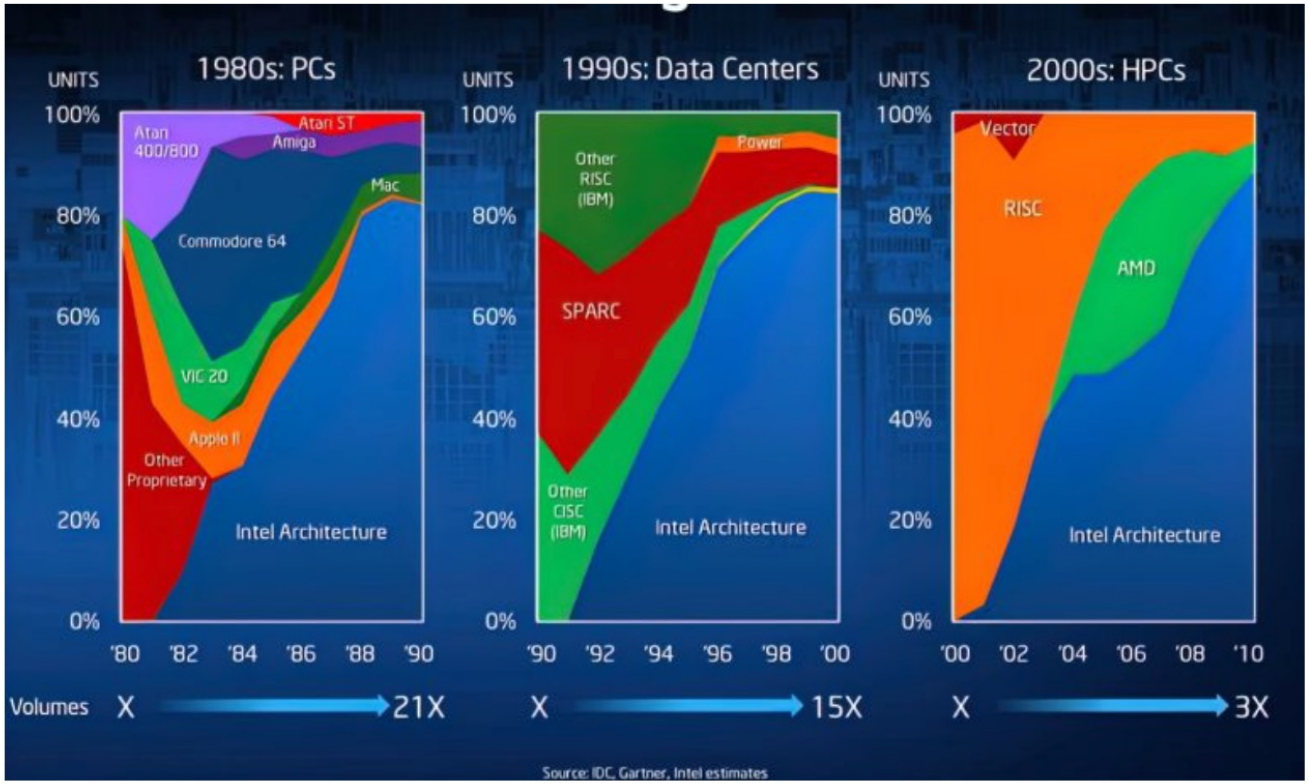


Hardware Interlock
or
Delay Slot (NOP)
(for LOAD, BR)

x86 vs RISC

EXTREMETECH

Early RISC CPU families like SPARC and HP's PA-RISC family also set performance records. During the late 1980s and early 1990s, it was common to hear people say that CISC-based architectures like x86 were the past, and perhaps good enough for home computing, but if you wanted to work with a *real* CPU, you bought a RISC chip. Data centers, workstations, and HPC is where RISC CPUs were most successful, as illustrated below:



This Intel image is useful but needs a bit of context. "Intel Architecture" appears to refer only to x86 CPUs — not chips like the 8080, which was popular in the early computer market. Similarly, Intel had a number of supercomputers in the "RISC" category in 2000 — it was x86 machines that gained market share, specifically.

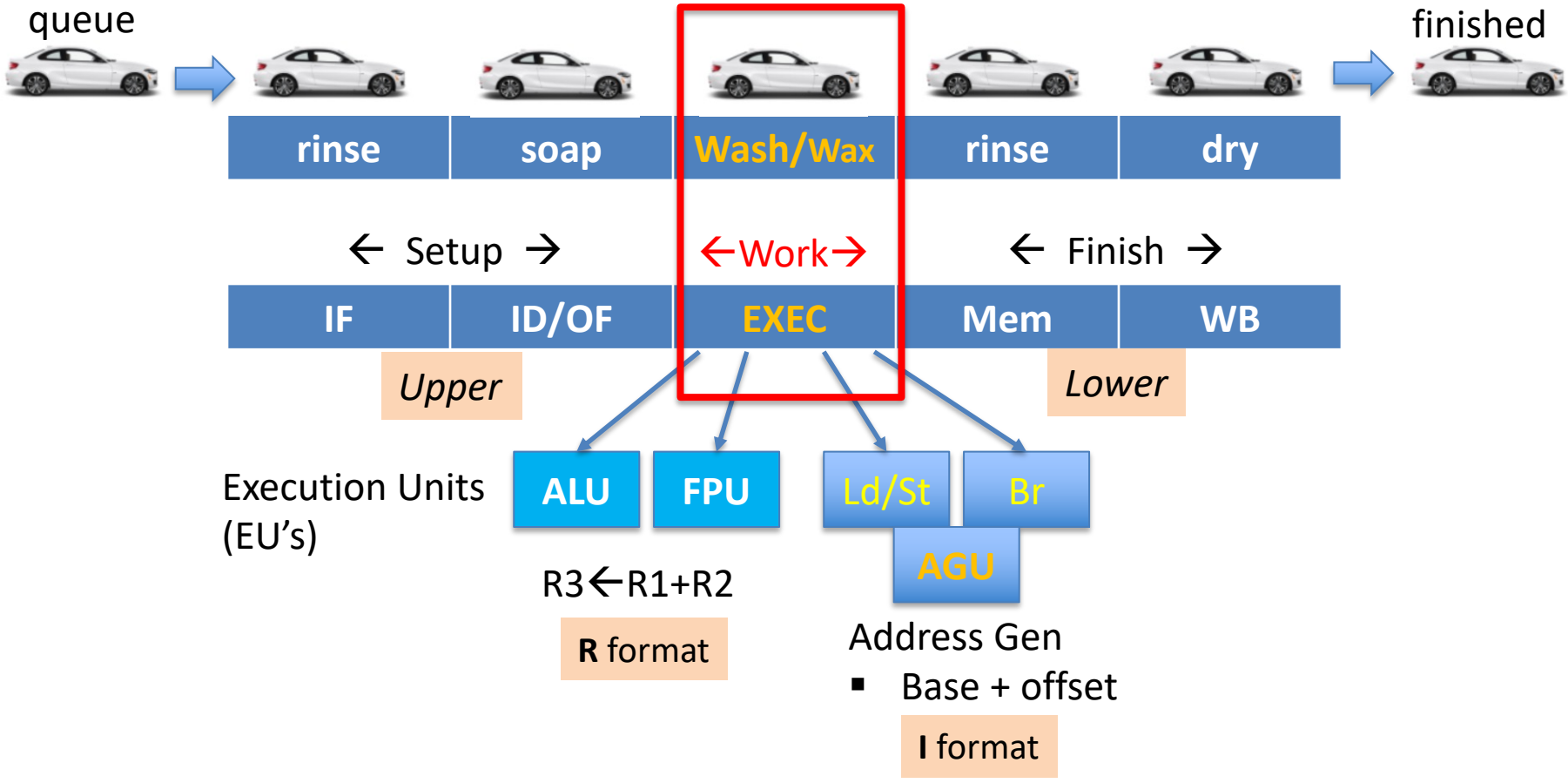
Section

Pipeline Details

MIPS RISC Pipeline

5 Stages

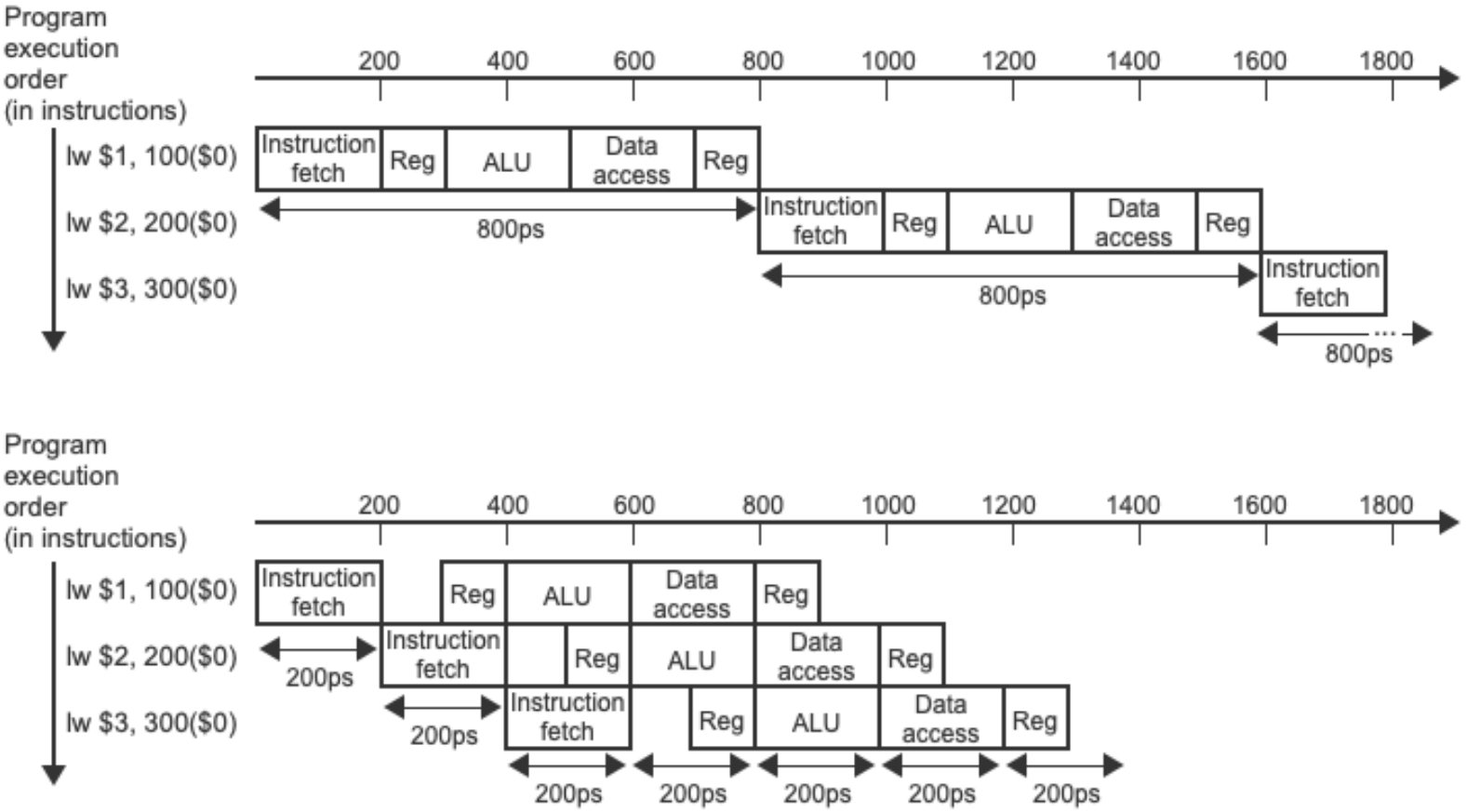
Each stage takes only 1/5 of instruction cycle: **clock F => 5x**



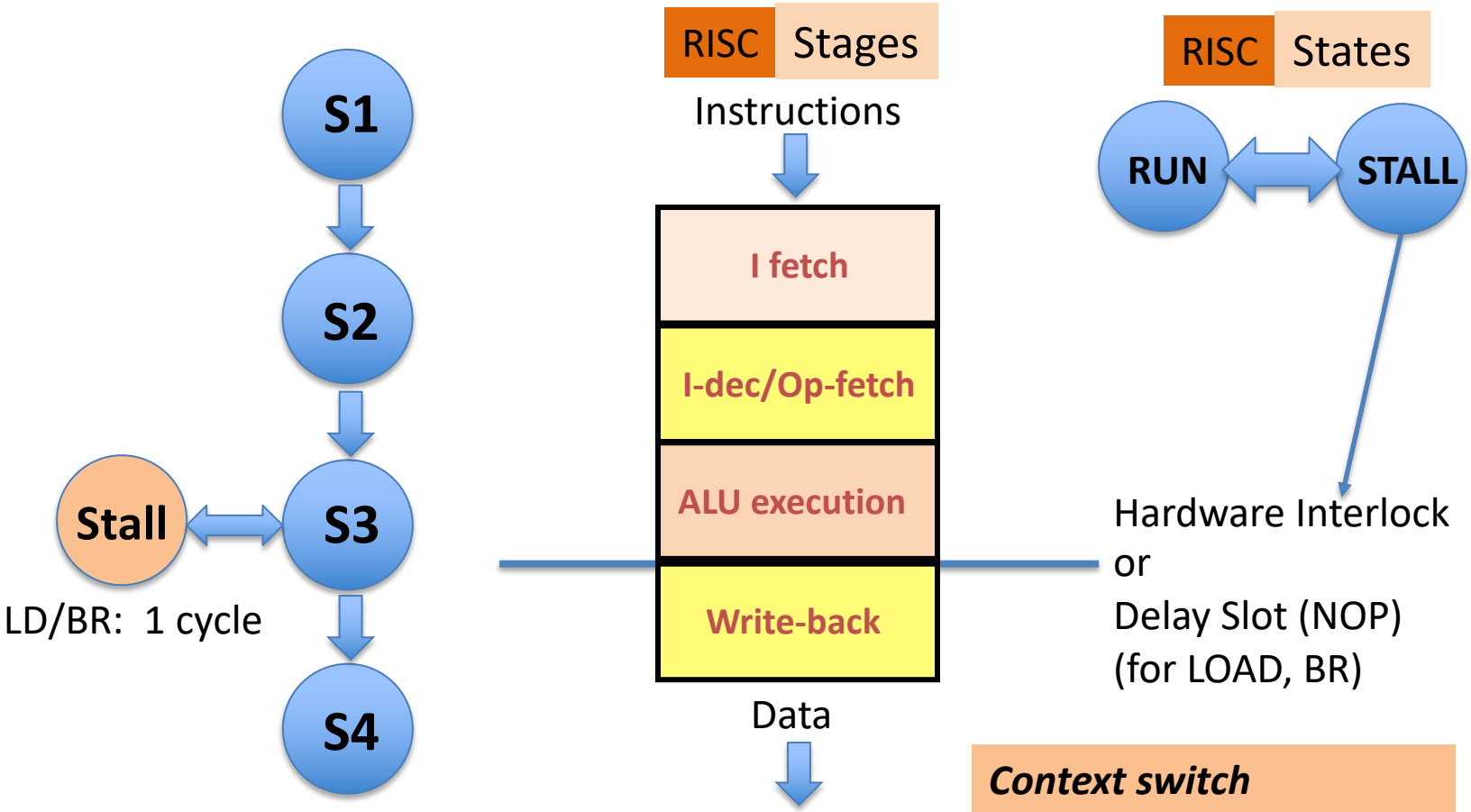
CISC/RISC Pipelines

PARTICIPATION ACTIVITY 4.5.3: Single-cycle, nonpipelined execution in top versus pipelined execution in bottom (COD Figure 4.27).

Pipelining: An implementation technique in which multiple instructions are overlapped in execution, much like an assembly line.



RISC Pipelines: Stages/States



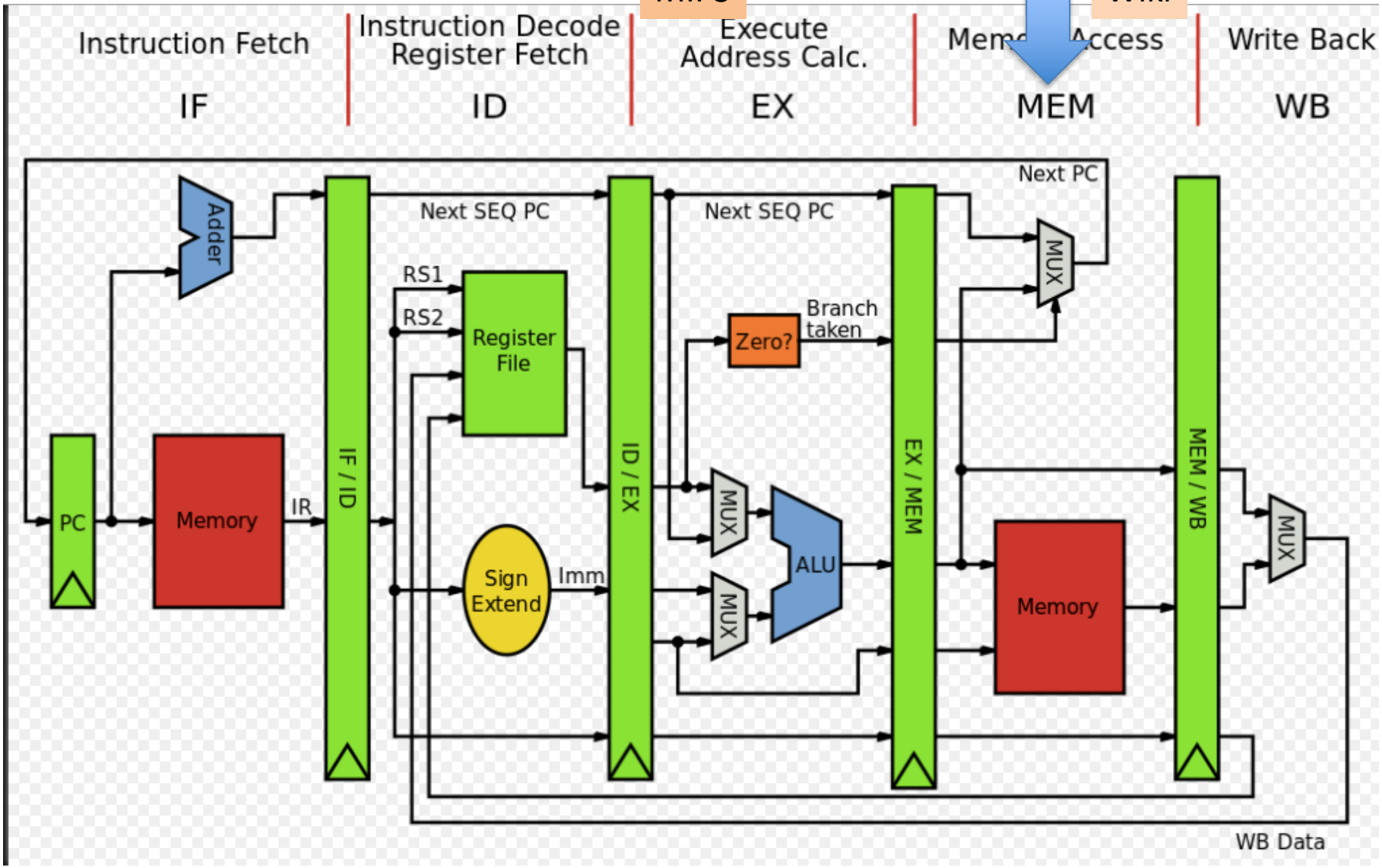
Instruction State Diagram

Context switch
Pipeline flush: stages 1-3

- Exceptions/Traps
- Interrupts
- System Calls

MIPS Pipelined Org

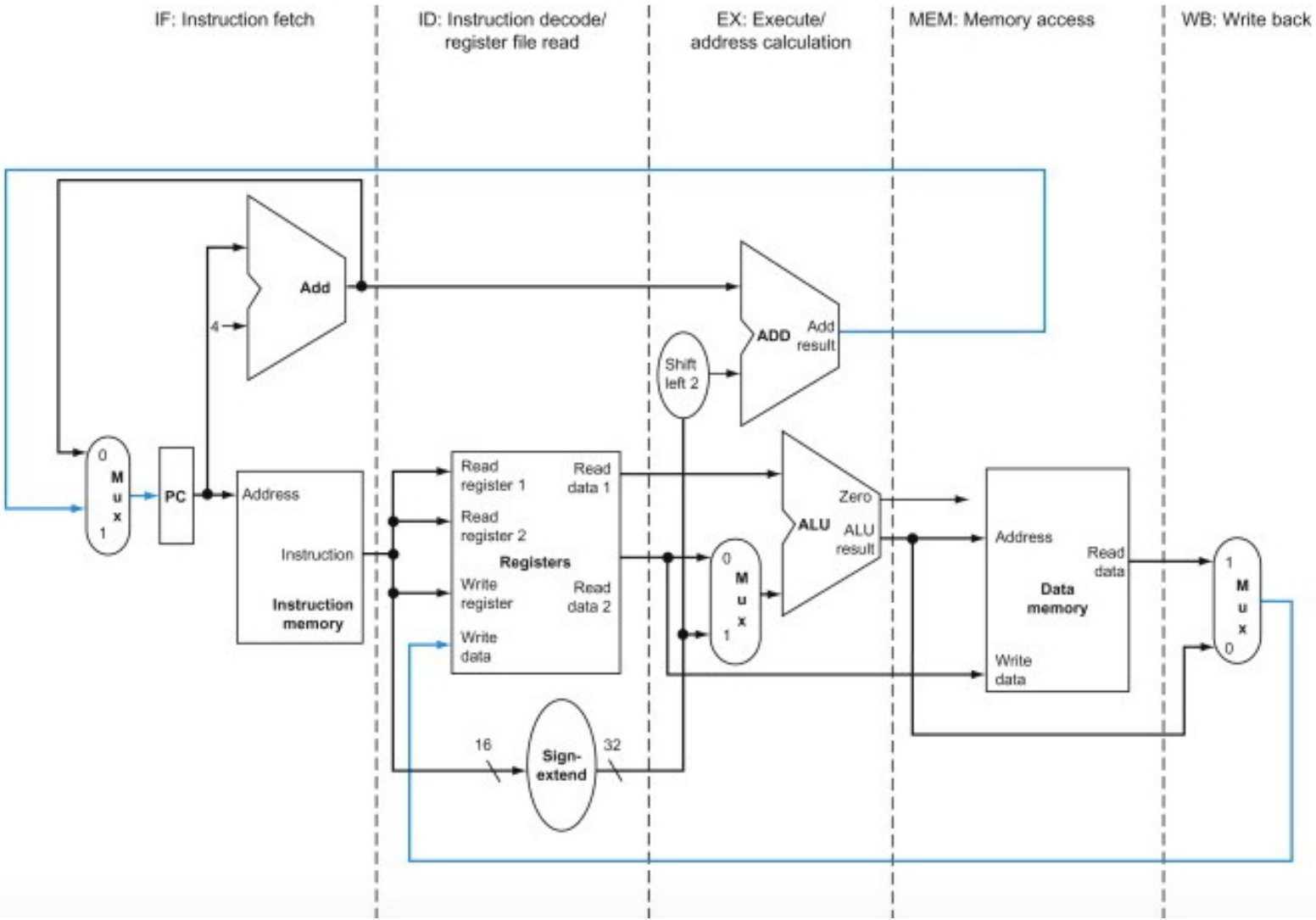
MIPS Wiki



MIPS, showing the five stages (instruction fetch, instruction decode, execute, memory access and write back).

MIPS Pipelined Org

Figure 4.6.1: The single-cycle datapath from COD Section 4.4 (A simple implementation scheme) (COD Figure 4



CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP222

RISC Pipelines

MIPS | Hennessy & Patterson 4.5

Figure 4.5.4: We need a stall even with forwarding when an R-format instruction following a load tries to

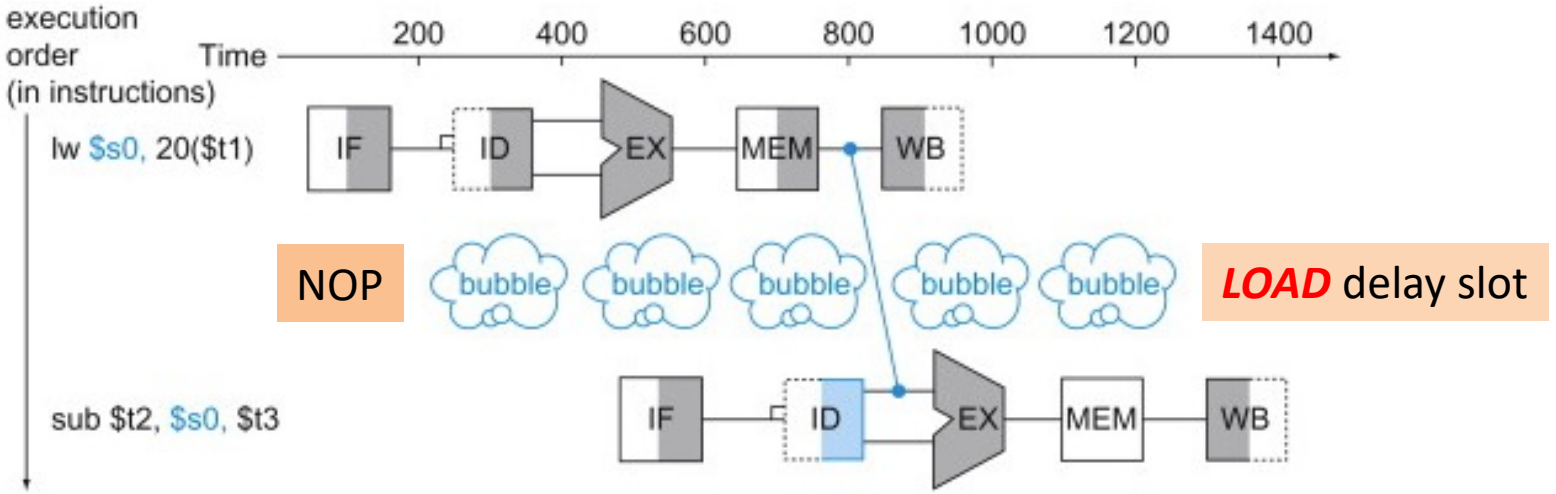
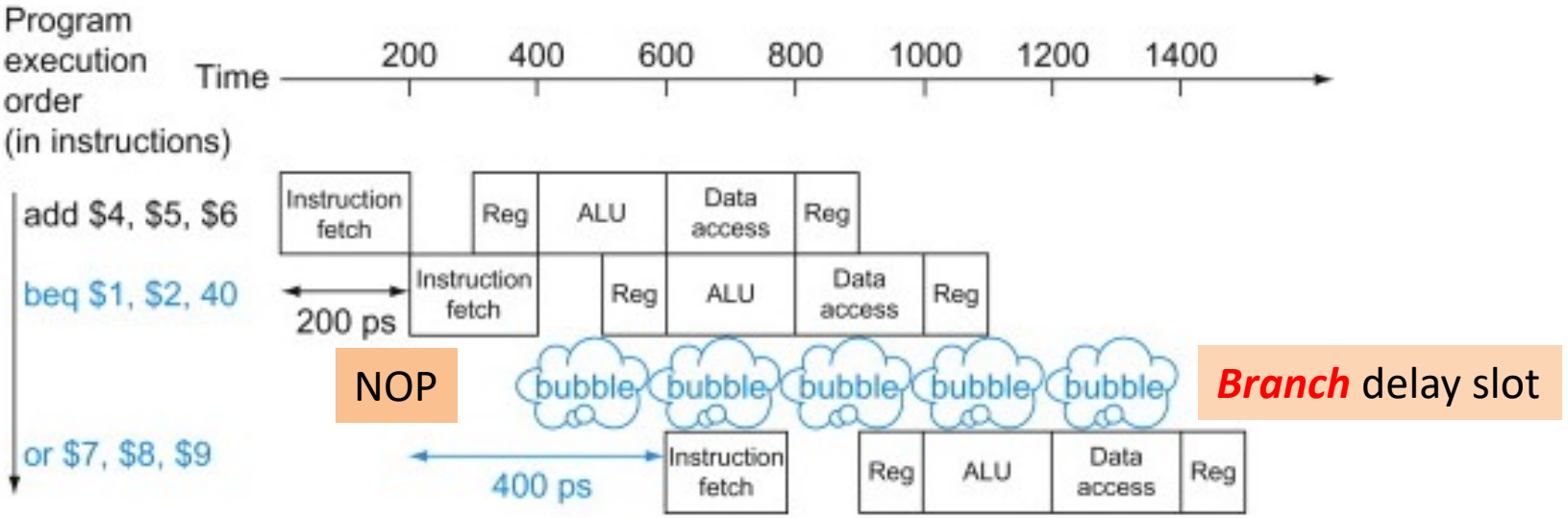


Figure 4.5.5: Pipeline showing stalling on every conditional branch as solution to control hazards



RISC Pipelines

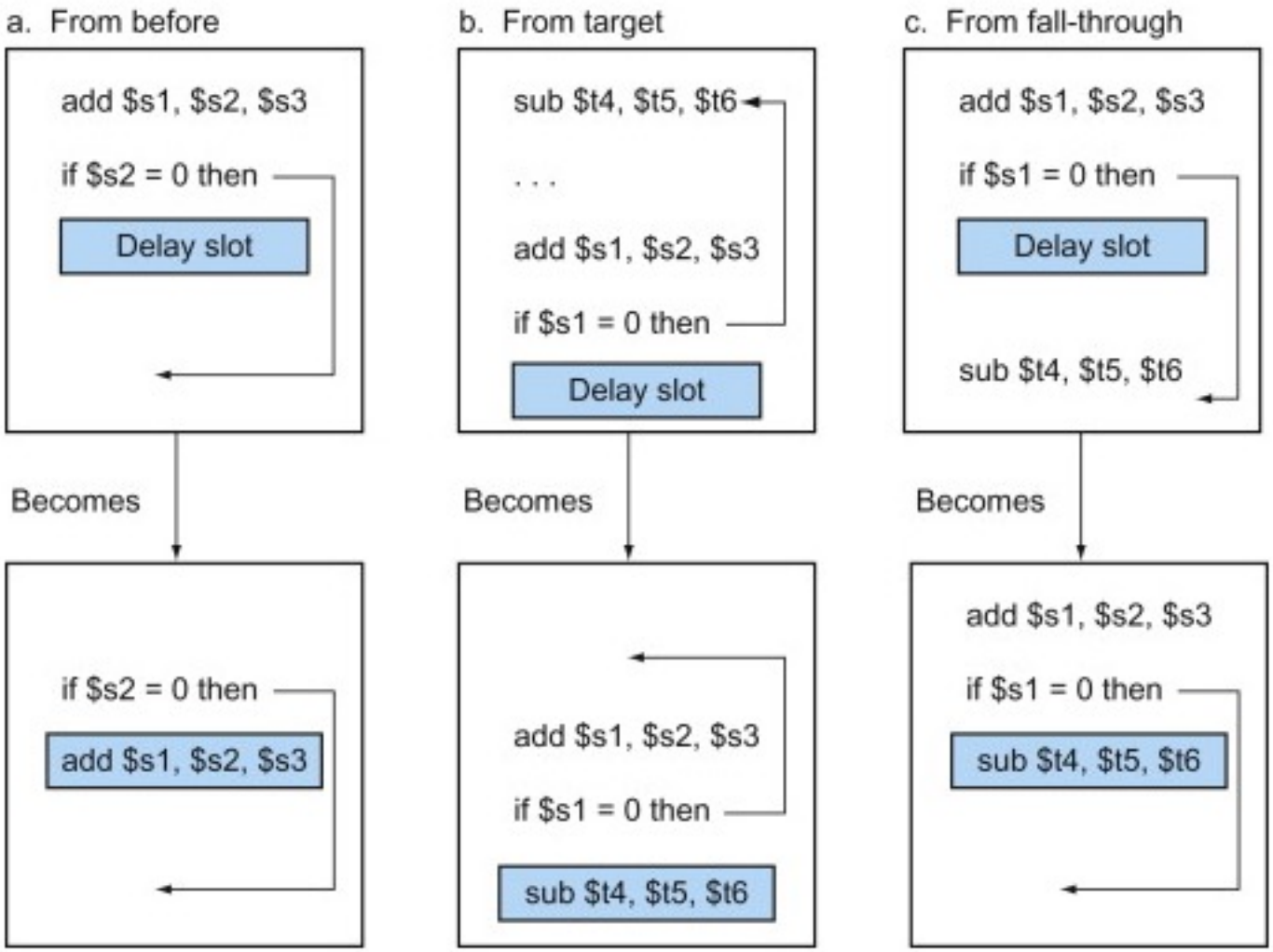


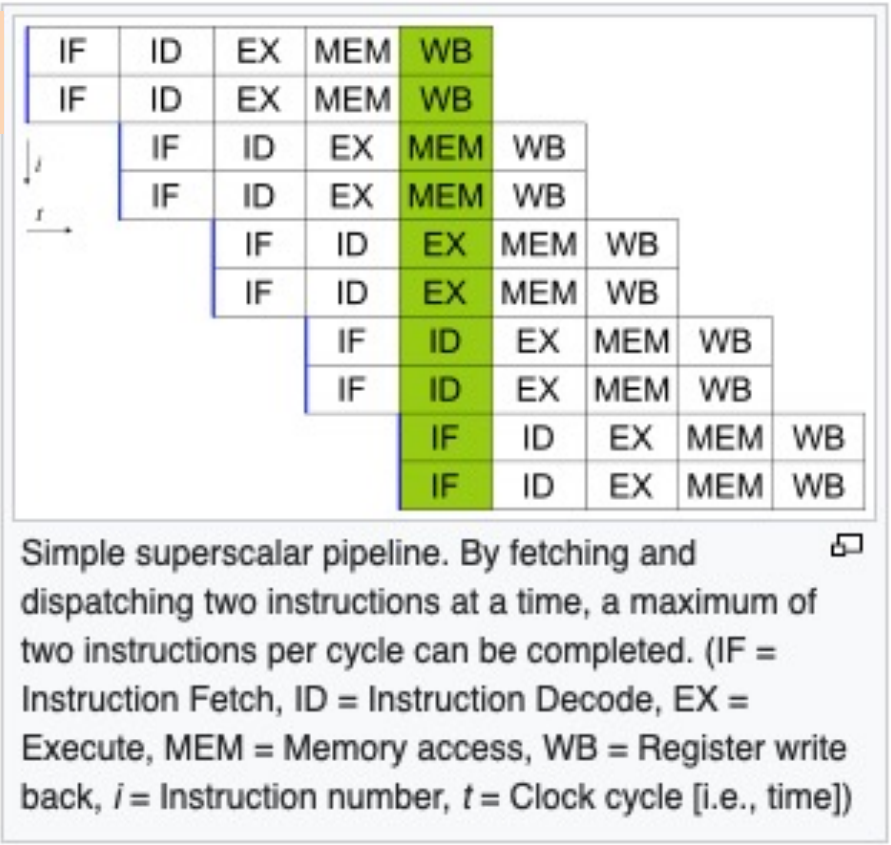
Figure 4.8.3: Scheduling the branch delay slot (

Superscalar

Super- *Scalar* SISD

- Multiple ***Execution Units*** (multi-issue pipelines)
  - each EU = ICU+ALU, with shared GR's
- Hardware + compiler schedules instruction streams

Th1	**EU1**	pipeline1
Th2	**EU2**	pipeline2

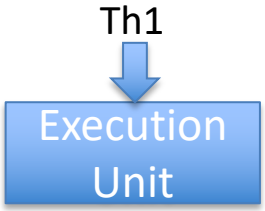


Simple superscalar pipeline. By fetching and dispatching two instructions at a time, a maximum of two instructions per cycle can be completed. (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back, *i* = Instruction number, *t* = Clock cycle [i.e., time])

Instruction Level *Parallelism*

Super- *Pipelining* SISD

- Split some pipeline stages (4-5 → 8-11)
- Faster clock cycle → higher *throughput* (*mips*)
- Affect CPI?

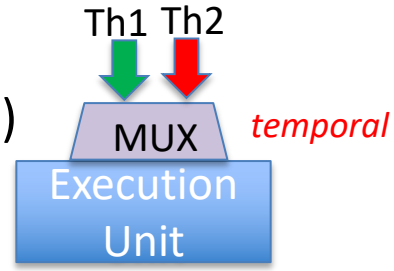


Super- *Scalar* SISD

- Multiple ***Execution Units*** (multi-issue pipelines)
  - each EU = ICU+ALU, with shared GR's
- Hardware + compiler *schedules* instruction streams

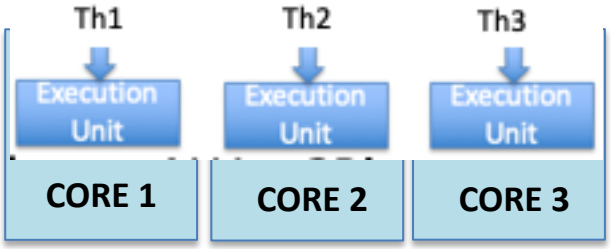
Multi- *Threading* SISD

- Multiple control threads (usually 2, same/dif program)
- Programs can *allocate* code to threads
- Automatic *scheduling* of control threads
- 2 types: *SMT/superscalar* or *temporal* (interleaved: coarse/fine)



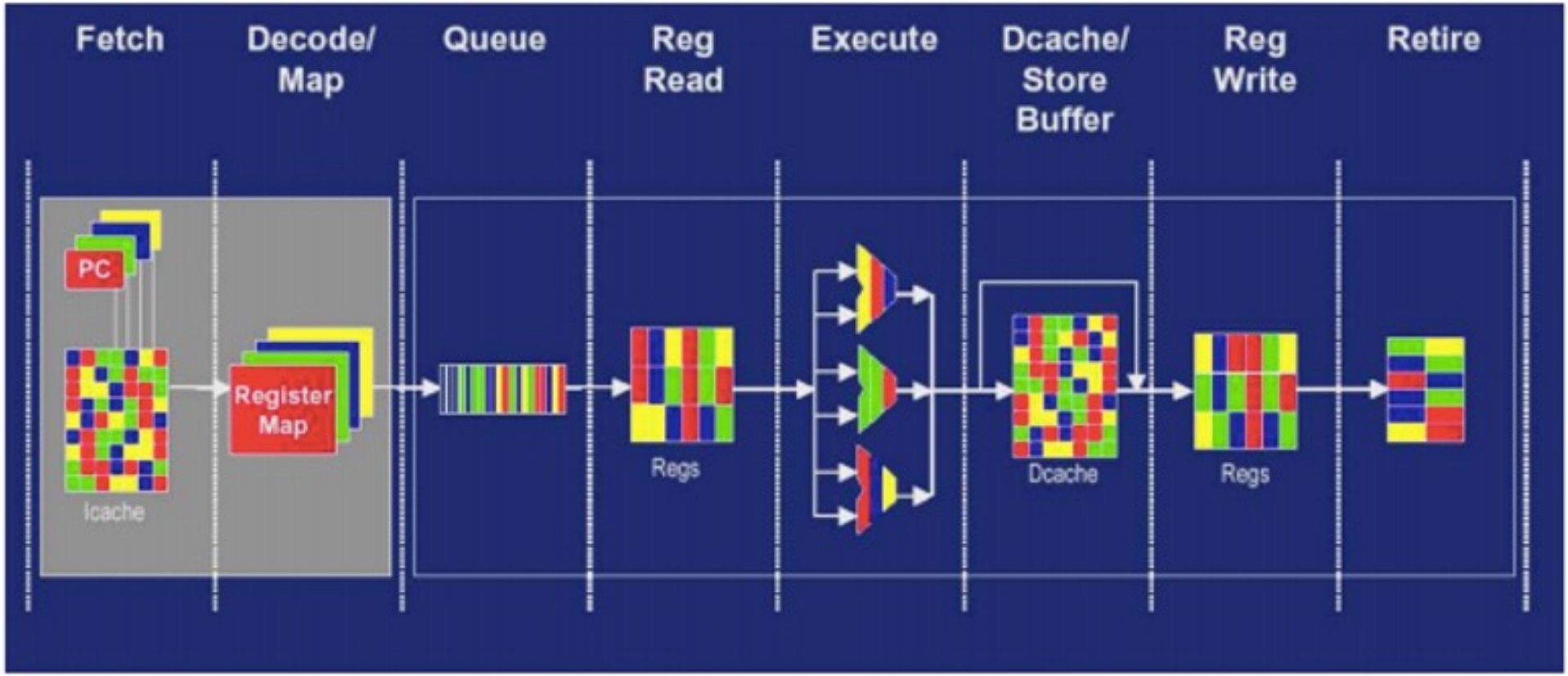
Multi- *Core* MISD

- Classic Parallelism: multiple copies of the CPU
- **Multiple L1/L2 caches** (one set per core)



Threads in a Pipeline

8 Stages



MIPS R4K SuperPipeline

MIPS R4000 Integer Pipeline

The MIPS R4000 architecture's use of superpipelining increases load and branch delays, and also the amount of forwarding required among the stages.

Load Delays:

If the result of a load is used in the next instruction, it causes a 2 cycle stall in the R4000 pipeline. This is due to the fact that the result of the load is not available to later instrucitons until the DS stage.

Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
LW R1,(R2)	IF	IS	RF	EX	DF	DS	TC	WB			
ADD R3,R4,R1		IF	IS	RF	stall	stall	EX	DF	DS	TC	WB

Branch Delays:

Taken branches result in a 3 cycle stall, as condition evaluation and branch target computation occur in the EX stage.

Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Branch	IF	IS	RF	EX	DF	DS	TC	WB	
Delay Slot		IF	IS	RF	EX	DF	DS	TC	WB
Stall			stall	stall	stall	stall	stall	stall	stall
Stall				stall	stall	stall	stall	stall	stall
Branch Target					IF	IS	RF	EX	DF

An untaken branch results in no stalls, it simply uses a one cycle delay slot.

Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Branch	IF	IS	RF	EX	DF	DS	TC	WB	
Delay Slot		IF	IS	RF	EX	DF	DS	TC	WB
Branch Instruction + 2			IF	IS	RF	EX	DF	DS	TC
Branch Instruction + 3				IF	IS	RF	EX	DF	DS

Section

Micro Architecture

- ❑ Overview
- ❑ Operand forwarding
- ❑ Branch prediction
- ❑ Speculation
- ❑ Re-Order

MT & Micro-Arch

Quora

Joe Zbiciak replied to your comment on an answer to: "Why does multithreading or hyperthreading only double your cores as threads? Why not triple or quadruple it?"

So the initial HT implementation was on Pentium 4, which was way off in a different architectural space with its trace cache, etc.

But once you returned to the Pentium Pro derived architecture (which went through Banias/Pentium M and became the Core architecture line), yes, superscalar processors converged on the same general plan;

- A front end that
  - renames registers
  - squashed moves and NOPs
  - decodes instructions into µops
  - possibly fuses µops into macro-ops
  - possibly caches the result
- A reorder buffer (ROB) that keeps track of all the outstanding µops/Mops
- Multiple execution pipelines for different categories of instructions
- Store buffers (incl. speculative store buffers) and store-to-load forwarding structures
- Store/load hazard violation detectors (to flush speculative loads that were violated by a store)

And of course the bread and butter branch predictors and prefetch engines for data.

HT & Micro-Arch

Quora

Joe Zbiciak replied to your comment on an answer to: "Why does multithreading or hyperthreading only double your cores as threads? Why not triple or quadruple it?"

The ROB and µarch rename register file really don't care what thread a given instruction comes from. It's largely a register rename problem.

For address translation, you need to tag all address generation with the logical thread the address belongs to so the µTLB, main TLB, and MMU table walker translate against the right translation context.

Likewise for any other privileged operations.

Intel® added a couple patented instructions to Pentium 4 to handle spinlocks more gracefully in an HT context. Basically, if one HT thread is blocked at a lock, it should yield to the other threads until something happens that's worth paying attention to. (ARM uses `WFE` for something similar, though I have doubts on its scalability.)

In any case, in principle, the ROB + rename file can handle any number of threads. Depending on how you handle architectural state, you may need to replicate your architecture register files, or replicate a portion of your rename file in proportion to the number of hardware threads

HT & Micro-Arch

Quora

Joe Zbiciak replied to your comment on an answer to: "Why does multithreading or hyperthreading only double your cores as threads? Why not triple or quadruple it?"

Jeff Drobman · Just now

yeah, so nail on head: now it is the ROB (Re-order Buffer) that manages EU slot issues which in essence supersedes Intel's "Hyper-threading". seems all major CPU's now use this micro- architecture. note that an ROB can contain any number of threads, including one, which then becomes simple superscalar. or am I wrong?

Yes, let's rule out temporal SMT.

For the Hyperthreading-style OoO SMT, the purpose is to increase performance and efficiency *per unit area*. Two cores will outperform a single similarly equipped core with SMT. If you normalize performance per unit of area, SMT wins.

Depending on the workload, 2-way SMT gives maybe 20% to 50% total throughput increase over a single thread on modern x86s. Workloads with experience a large number of L2/L3 stalls leave the core idle, and so those would benefit from other threads to scavenge resources.

It's too bad the Thunder X3 won't see the light of day. I was curious how its 4-way SMT was going to perform.

MT HW/SW SPI

Quora

Drazen Zoric · Follow
Lives in Cork, Ireland ·

System Programming Interface

Every thread has a block of memory where CPU state is stored during task switch, has some ID, priority, current thread state, etc. CPU state is copy of all CPU registers, including floating point but also MMU mappings which belong to a process (thread also has process ID).

When thread is created OS will create CPU state in memory, will set PC to thread entry point, EAX register to thread parameter (in Windows thread function is `HRESULT ThreadFunc( LPVOID )`), will populate other params (ID, priority, etc) and will add Thread ID (or pointer on CPU state) into Scheduler.

When time comes to execute a thread, Scheduler will store currently running thread CPU state into memory, will find next thread to run, copy CPU state from memory into CPU and on exit from tick interrupt address in PC will be next executed.

If you really wanna see how it works, download FreeRTOS and check in sources how Task Switch is performed. It's similar to above. For sure Schedulers in Windows/Linux are way more complex but that's basic operation.
I remember long ago I read how x86 has an instruction to store/restore CPU state in memory but is not used cause it is too slow.

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

Quora

MT HW/SW SPI

DR JEFF SOFTWARE INDIE APP DEVELOPER

COMP222

Drazen Zoric · Follow
Lives in Cork, Ireland ·

System Programming Interface

Jeff Drobman · Just now

thank you for this. I teach multi-threading architecture, but was missing this detail. but, seems you are addressing only Coarse-grained Temporal MT, where a "thread switch" occurs. SMT does no switching, but I suppose the handling of threads by the OS is the same.

Operand Forwarding

Arup Das, BE from Siddaganga Institute of Technology (2019)
Answered June 27

In today's architecture dependencies between instructions are checked statically by the compiler and/or dynamically by the the hardware at runtime.

Dynamic pipeline scheduling allows out of order execution giving rise to WAR and WAW data hazards (Reference: Computer Organization and Architecture | Pipelining | Set 2 (Dependencies and Data Hazard) - GeeksforGeeks).

For dynamic dependency checking there are 2 techniques (Reference: http://www2.cs.siu.edu/~cs401/Textbook/ch3.pdf) :

1. Tomasulo's Method
2. Scoreboard Method

Operand Forwarding

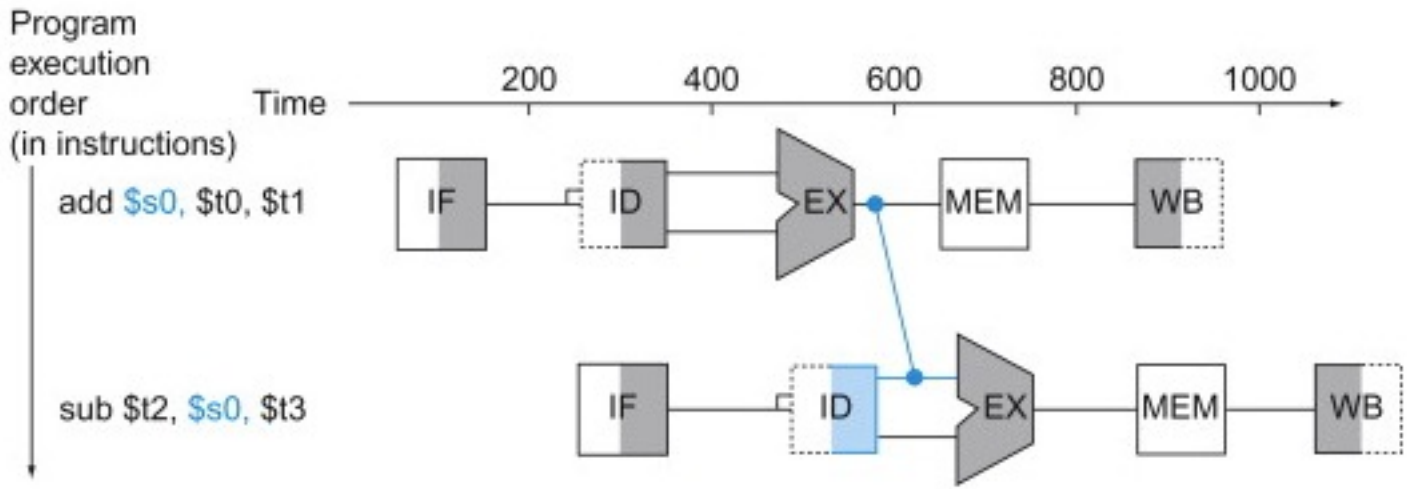
Now coming to operand forwarding. In a pipelined processor which allows in-order execution of instructions, operand forwarding which is a hardware technique is used to reduce or prevent stalls created to prevent RAW Hazards (Reference: http://meseec.ce.rit.edu/eecc551-spring2012/551-3-26-2012.pdf).

Operand forwarding or Memory forwarding is nothing but forwarding the data from output of one stage to input of another stage as soon as the data to be forwarded is ready. Data forwarding takes place from "left to right" in time and it solely depends upon the type of instruction and when the data is required during the execution of the dependent instruction. (Reference: http://web.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/forward.html)

Data is forwarded either to EX stage or to MEM stage.

NOTE: Split phase technique is used to forward data between EX stage and Decode stage OR WB stage and Decode stage (because generally whenever we need to write anything into register, this will be done at rising edge, and reading at falling edge). This happens within a single clock cycle unlike operand forwarding where the data is forwarded only after the previous clock cycle is complete.

Data Forwarding



Operand Forwarding

Forwarding

The problem with data hazards, introduced by this sequence of instructions can be solved with a simple hardware technique called ***forwarding***.

		1	2	3	4	5	6	7
ADD	**R1**, R2, R3	IF	ID	EX	MEM	**WB**		
SUB	R4, R5, **R1**		IF	ID_{sub}	EX	MEM	WB	
AND	R6, **R1**, R7			IF	ID_{and}	EX	MEM	WB

The key insight in forwarding is that the result is not really needed by SUB until after the ADD actually produces it. The only problem is to make it available for SUB when it needs it.

If the result can be moved from where the ADD produces it (EX/MEM register), to where the SUB needs it (ALU input latch), then the need for a stall can be avoided.
Using this observation , forwarding works as follows:

- The ALU result from the EX/MEM register is always ***fed back*** to the ALU input latches.
- If the forwarding hardware detects that the previous ALU operation has written the register corresponding to the source for the current ALU operation, ***control logic*** selects the forwarded result as the ALU input rather than the value read from the register file.

Operand Forwarding

Without forwarding our example will execute correctly with stalls:

		1	2	3	4	5	6	7	8	9
ADD	R1, R2, R3	IF	ID	EX	MEM	WB				
SUB	R4, R5, R1		IF	stall	stall	ID_{sub}	EX	MEM	WB	
AND	R6, R1, R7			stall	stall	IF	ID_{and}	EX	MEM	WB

As our example shows, we need to forward results not only from the immediately previous instruction, but possibly from an instruction that started three cycles earlier. Forwarding can be arranged from MEM/WB latch to ALU input also. Using those forwarding paths the code sequence can be executed without stalls:

		1	2	3	4	5	6	7
ADD	R1, R2, R3	IF	ID	EX_{add}	MEM_{add}	WB		
SUB	R4, R5, R1		IF	ID	EX_{sub}	MEM	WB	
AND	R6, R1, R7			IF	ID	EX_{and}	MEM	WB

The first forwarding is for value of R1 from EX_{add} to EX_{sub} .
The second forwarding is also for value of R1 from MEM_{add} to EX_{and}.
This code now can be executed without stalls.

Forwarding can be generalized to include passing the result directly to the functional unit that requires it: a result is forwarded from the output of one unit to the input of another, rather than just from the result of a unit to the input of the same unit.

Operand Forwarding

One more Example

To prevent a stall in this example, we would need to forward the values of R1 and R4 from the pipeline registers to the ALU and data memory inputs.

		1	2	3	4	5	6	7
ADD	R1, R2, R3	IF	ID	EX_{add}	MEM_{add}	WB		
LW	R4, d (R1)		IF	ID	EX_{lw}	MEM_{lw}	WB	
SW	R4,12(R1)			IF	ID	EX_{sw}	MEM_{sw}	WB

Stores require an operand during MEM, and forwarding of that operand is shown here.
The first forwarding is for value of R1 from EX_{add} to EX_{lw} .
The second forwarding is also for value of R1 from MEM_{add} to EX_{sw}.
The third forwarding is for value of R4 **from** MEM_{lw} to MEM_{sw}.
Observe that the SW instruction is storing the value of R4 into a memory location computed by adding the displacement 12 to the value contained in register R1. This effective address computation is done in the ALU during the EX stage of the SW instruction. The value to be stored (R4 in this case) is needed only in the MEM stage as an input to Data Memory. Thus the value of R1 is forwarded to the EX stage for effective address computation and is needed earlier in time than the value of R4 which is forwarded to the input of Data Memory in the MEM stage.
So forwarding takes place from "left-to-right" in time, but operands are not ALWAYS forwarded to the EX stage - it depends on the instruction and the point in the Datapath where the operand is needed. Of course, hardware support is necessary to support data forwarding.

MIPS R4K SuperPipeline

Data Forwarding

Forwarding

When compared to the simple 5-Stage DLX, the R4000 requires many more levels of forwarding.

5-Stage DLX forwarding for ALU register-register instructions can occur from the EX/MEM or the MEM/WB registers. The following table shows the possible sources and destination of forwarding for the 5-Stage DLX Integer pipeline.

Pipeline Register Source	Opcode of Source Instruction	Pipeline Register Destination	Opcode of Destination Instruciton	Destination of Forwarded Result
EX/MEM	Register-register ALU	ID/EX	All	Top ALU Input
EX/MEM	Register-register ALU	ID/EX	Register-register ALU	Bottom ALU Input
MEM/WB	Register-register ALU	ID/EX	All	Top ALU Input
MEM/WB	Register-register ALU	ID/EX	Register-register ALU	Bottom ALU Input
EX/MEM	ALU Immediate	ID/EX	All	Top ALU Input
EX/MEM	ALU Immediate	ID/EX	Register-register ALU	Bottom ALU Input
MEM/WB	ALU Immediate	ID/EX	All	Top ALU Input
MEM/WB	ALU Immediate	ID/EX	Register-register ALU	Bottom ALU Input
MEM/WB	Load	ID/EX	All	Top ALU Input
MEM/WB	Load	ID/EX	Register-register ALU	Bottom ALU Input

* Note: 'All' corresponds to the following instruction types: Register-register ALU, ALU-Immediate, Load, Store, and Branch

MIPS R4K SuperPipeline

Data Forwarding

The R4000 pipeline has four sources for ALU register-register operations: EX/DF, DF/DS, DS/TC, TC/WB. The following table shows the possible sources and destinations of forwarding for the R4000 Integer pipeline.

Pipeline Register Source	Opcode of Source Instruction	Pipeline Register Destination	Opcode of Destination Instruciton	Destination of Forwarded Result
EX/DF	Register-register ALU	RF/EX	All	Top ALU Input
EX/DF	Register-register ALU	RF/EX	Register-register ALU	Bottom ALU Input
DF/DS	Register-register ALU	RF/EX	All	Top ALU Input
DF/DS	Register-register ALU	RF/EX	Register-register ALU	Bottom ALU Input
DS/TC	Register-register ALU	RF/EX	All	Top ALU Input
DS/TC	Register-register ALU	RF/EX	Register-register ALU	Bottom ALU Input
TC/WB	Register-register ALU	RF/EX	All	Top ALU Input
TC/WB	Register-register ALU	RF/EX	Register-register ALU	Bottom ALU Input
EX/DF	ALU Immediate	RF/EX	All	Top ALU Input
EX/DF	ALU Immediate	RF/EX	Register-register ALU	Bottom ALU Input
DF/DS	ALU Immediate	RF/EX	All	Top ALU Input
DF/DS	ALU Immediate	RF/EX	Register-register ALU	Bottom ALU Input
DS/TC	ALU Immediate	RF/EX	All	Top ALU Input
DS/TC	ALU Immediate	RF/EX	Register-register ALU	Bottom ALU Input
TC/WB	ALU Immediate	RF/EX	All	Top ALU Input
TC/WB	ALU Immediate	RF/EX	Register-register ALU	Bottom ALU Input
DS/TC	Load	RF/EX	All	Top ALU Input
DS/TC	Load	RF/EX	Register-register ALU	Bottom ALU Input
TC/WB	Load	RF/EX	All	Top ALU Input
TC/WB	Load	RF/EX	Register-register ALU	Bottom ALU Input

* Note: 'All' corresponds to the following instruction types: Register-register ALU, ALU-Immediate, Load, Store, and Branch

Micro-Arch: Branches

What is branch prediction in computer architecture?

Jeff Drobman, Lecturer at California State University, Northridge (2016-present)

Answered 4m ago

branch prediction: instruction control units fetch instructions in sequence, until a branch (or jump) instruction. conditional branches may or may not result in a branch, but the ICU has to decide which instruction to fetch next: the next one or the branch target. while an ICU could be designed to always make the same choice, it is more efficient to ***predict*** which way to go. typically, branch prediction can add a small (5–10%) boost to performance (CPI).

Branch Prediction

Quora

Dani Richard · Follow

B.S. in Information and Computer Science & Systems Programming, Georgia Institute of Technology · 2y

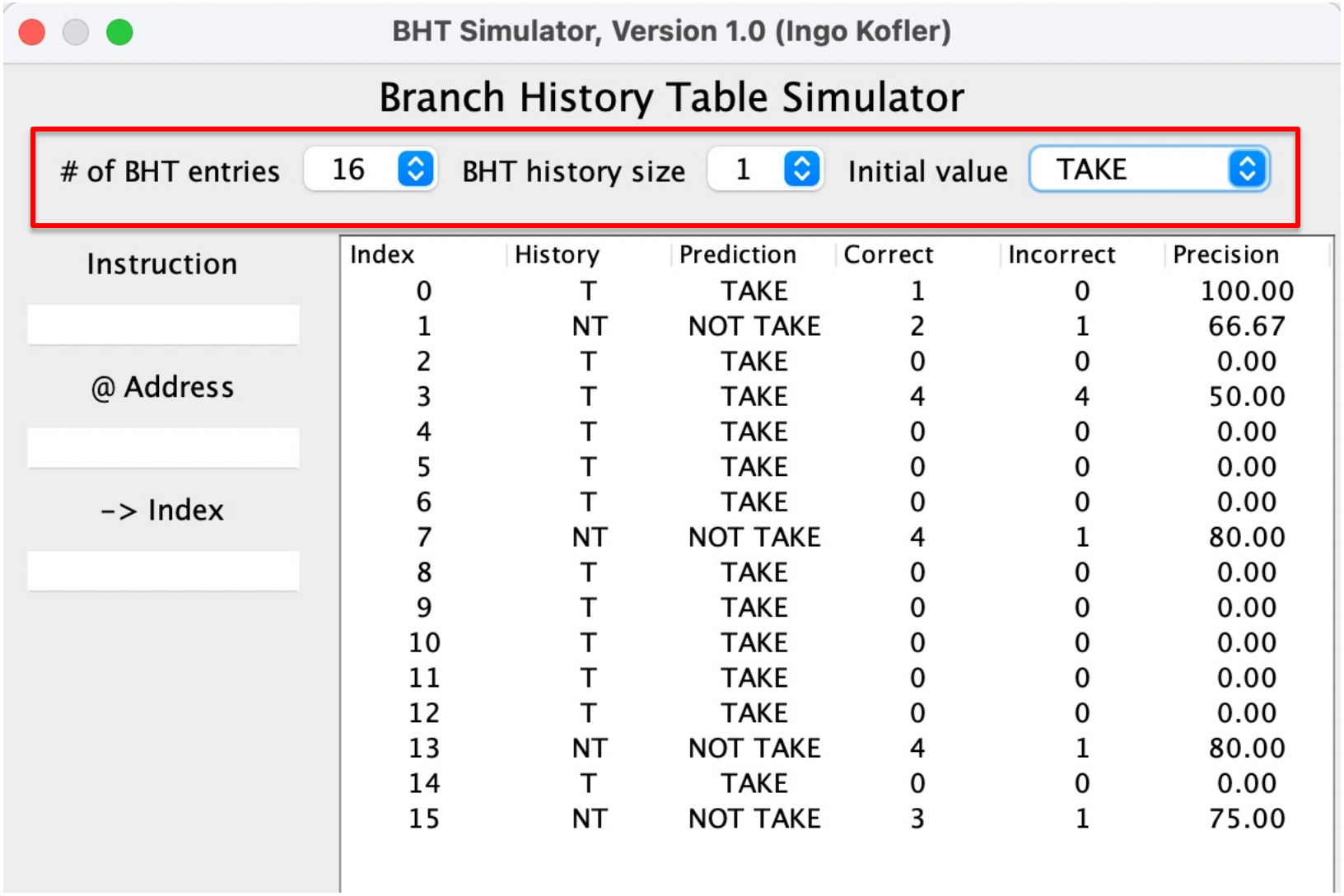
Related **Can branch prediction go beyond 99%? If so how does it affect the performance, and what's the newest type of branch prediction scientists are working on?**

Question: Can branch prediction go beyond 99%? If so how does it affect the performance, and what's the newest type of branch prediction scientist are working on?

Answer: I only have knowledge to answer your first question.

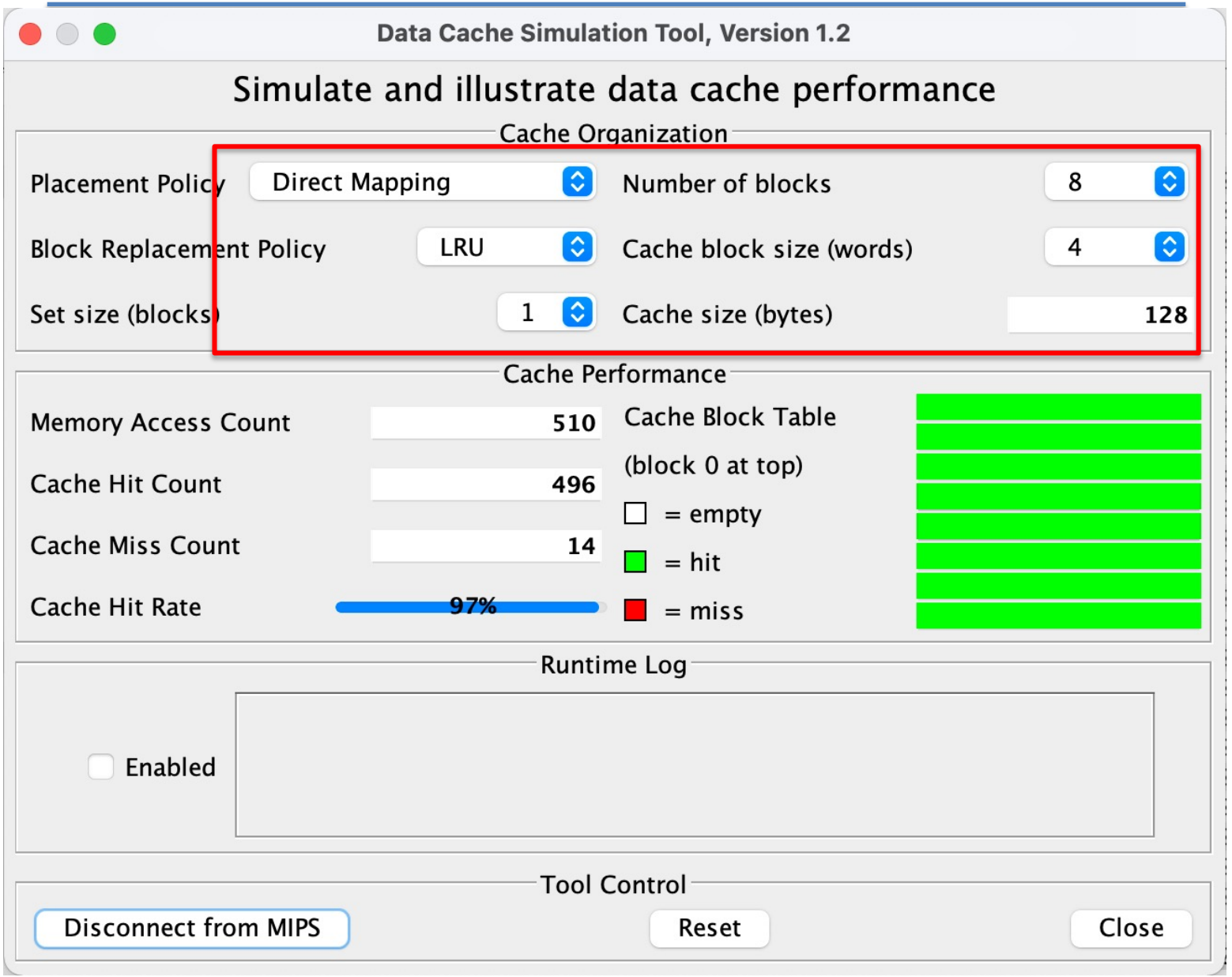
Yes, the PowerPC G5 processor had three branch processing sub-units. Assuming all were available and an un-resolved branch was encountered, a branch-sub unit would start executing down BOTH branches! Should another condition branch be encountered, a speculative branch based either history or hit would be taken. Any additional branches encountered would cause a stall until branches resolved. When the first branch is resolved, the Not-Taken computations would be throw away. They become waste heat. With at least two branch sub-units available, all branches are correctly taken. That waste heat for computations not used contribute to G5s running hot.

Branch Prediction



Index	History	Prediction	Correct	Incorrect	Precision
0	T	TAKE	1	0	100.00
1	NT	NOT TAKE	2	1	66.67
2	T	TAKE	0	0	0.00
3	T	TAKE	4	4	50.00
4	T	TAKE	0	0	0.00
5	T	TAKE	0	0	0.00
6	T	TAKE	0	0	0.00
7	NT	NOT TAKE	4	1	80.00
8	T	TAKE	0	0	0.00
9	T	TAKE	0	0	0.00
10	T	TAKE	0	0	0.00
11	T	TAKE	0	0	0.00
12	T	TAKE	0	0	0.00
13	NT	NOT TAKE	4	1	80.00
14	T	TAKE	0	0	0.00
15	NT	NOT TAKE	3	1	75.00

Cache Simulation



OOE → ROB

Quora

SPARC?

Joe Zbiciak · 1h ago

Up until near the end, SPARC was in-order, and they made use of their execute resources with extreme threading. Peak single thread performance wasn't the objective,,peak throughput for the system was.

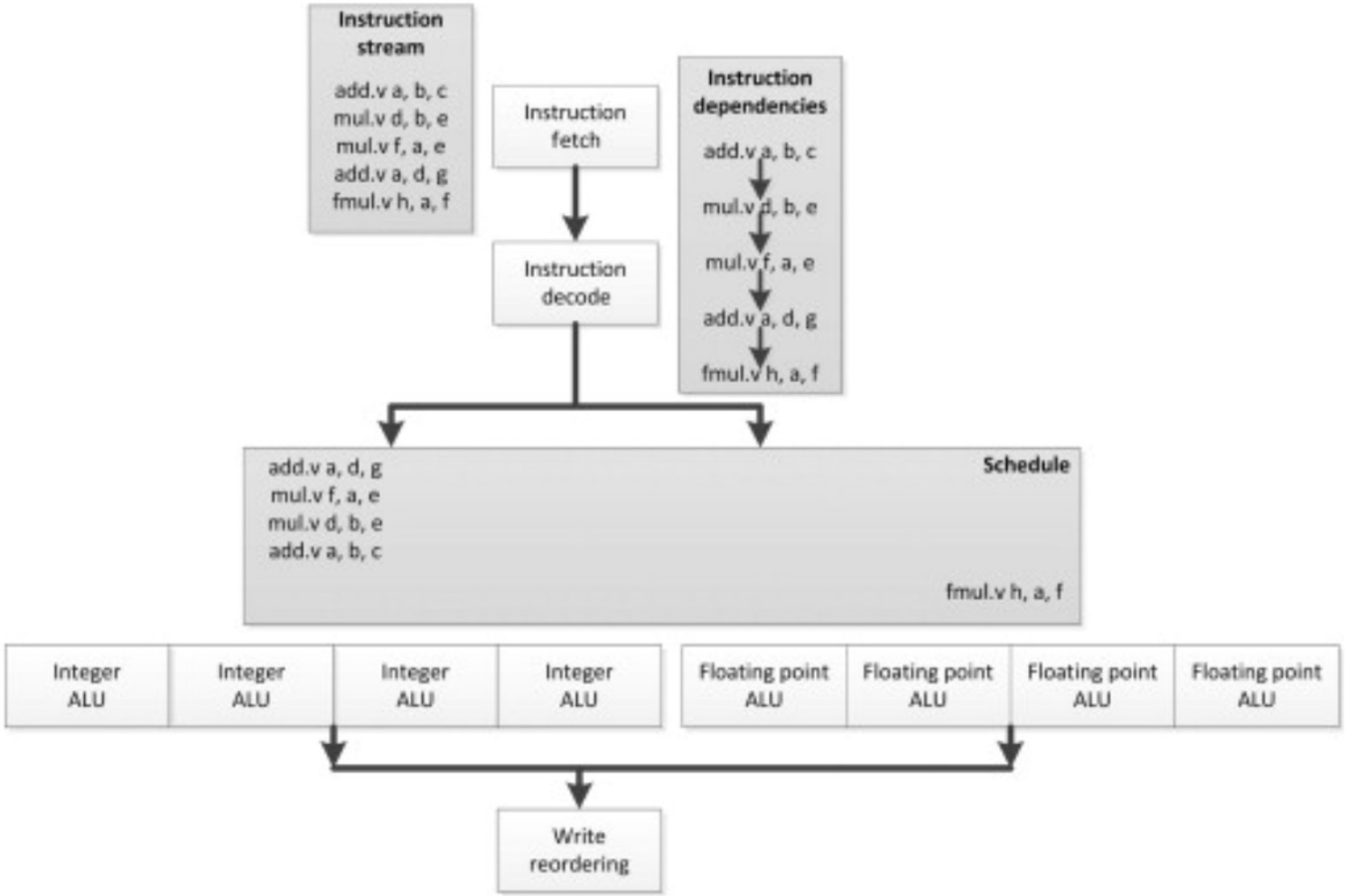
On x86, single thread performance is still king. And with two virtual threads per core you can still do OK on highly threaded tasks.

SPARC's deep multithreading was designed to hide memory latency.

You can also get there with more cores and fewer hardware threads per core. If a large enough portion of your benchmark is dominated by single thread performance (or threads that have good cache locality), the latter is a more compelling configuration.

ARM SIMD Re-Order

- "ARM has also introduced SIMD extensions to ARM-Cortex architecture with their NEON technology" (8).



Section

Brand Micro-architecture

- x86
- Intel
- AMD
- Apple
- Fujitsu

x86 uArch

EXTREMETECH

And here, at last, we arrive at the heart of the question: Just how heavy a penalty do modern AMD and Intel CPUs pay for x86 compatibility?

The decode bottleneck, branch prediction, and pipeline complexities that Agner refers to above are part of the "CISC tax" that ARM argues x86 incurs. In the past, Intel and AMD have told us decode power is a single-digit percentage of total chip power consumption. But that doesn't mean much if a CPU is burning power for a micro-op cache or complex branch predictor to compensate for the lack of decode bandwidth. Micro-op cache power consumption and branch prediction power consumption are both determined by the CPU's microarchitecture and its manufacturing process node. "RISC versus CISC" does not adequately capture the complexity of the relationship between these three variables.

It's going to take a few years before we know if Apple's M1 and future CPUs from Qualcomm represent a sea change in the market or the next challenge AMD and Intel will rise to. Whether maintaining x86 compatibility is a burden for modern CPUs is both a new question and a very old one. New, because until the M1 launched, there was no meaningful comparison to be made. Old, because this topic used to get quite a bit of discussion back when there were non-x86 CPUs still being used in personal computers.

AMD continues to improve Zen by 1.15x – 1.2x per year. We know Intel's Alder Lake will also use low-power x86 CPU cores to improve idle power consumption. Both x86 manufacturers continue to evolve their approaches to performance. It will take time to see how these cores, and their successors, map against future Apple products — but x86 is not out of this fight.

x86 uArch

EXTREMETECH

> The complicated x86 ISA makes decoding a bottleneck. An x86 instruction can have any length from 1 to 15 bytes, and it is quite complicated to calculate the length. And you need to know the length of one instruction before you can begin to decode the next one. This is certainly a problem if you want to decode 4 or 6 instructions per clock cycle! Both Intel and AMD now keep adding bigger micro-op caches to overcome this bottleneck. ARM has fixed-size instructions so this bottleneck doesn't exist and there is no need for a micro-op cache.
>
> Another problem with x86 is that it needs a long pipeline to deal with the complexity. The branch misprediction penalty is equal to the length of the pipeline. So they are adding ever-more complicated branch prediction mechanisms with large branch history tables and branch target buffers. All this, of course, requires more silicon space and more power consumption.
>
> The x86 ISA is quite successful despite of these burdens. This is because it can do more work per instruction. For example, A RISC ISA with 32-bit instructions cannot load a memory operand in one instruction if it needs 32 bits just for the memory address.

In his microarchitectural manual, Agner also writes that more recent trends in AMD and Intel CPU designs have hearkened back to CISC principles to make better use of limited code caches, increase pipeline bandwidth, and reduce power consumption by keeping fewer micro-ops in the pipeline. These improvements represent microarchitectural offsets that have improved overall x86 performance and power efficiency.

x86 uArch

EXTREMETECH

The K5 re-used parts of the execution back-end AMD developed for its Am29000 family of RISC CPUs, and it implements an internal instruction set that is more RISC-like than the native x86 ISA. The RISC-style techniques NexGen and AMD refer to during this period reference concepts like data caches, pipelining, and superscalar architectures. Two of these — caches and pipelining — are named in Patterson's paper. None of these ideas are *strictly* RISC, but they all debuted in RISC CPUs first, and they were advantages associated with RISC CPUs when K5 was new. Marketing these capabilities as "RISC-like" made sense for the same reason it made sense for OEMs of the era to describe their PCs as "IBM-compatible."

The degree to which these features are RISC and the answer to whether x86 CPUs decode RISC-style instructions depends on the criteria you choose to frame the question. The argument is larger than the Pentium Pro, even if P6 is the microarchitecture most associated with the evolution of techniques like an out-of-order execution engine. Different engineers at different companies had their own viewpoints.

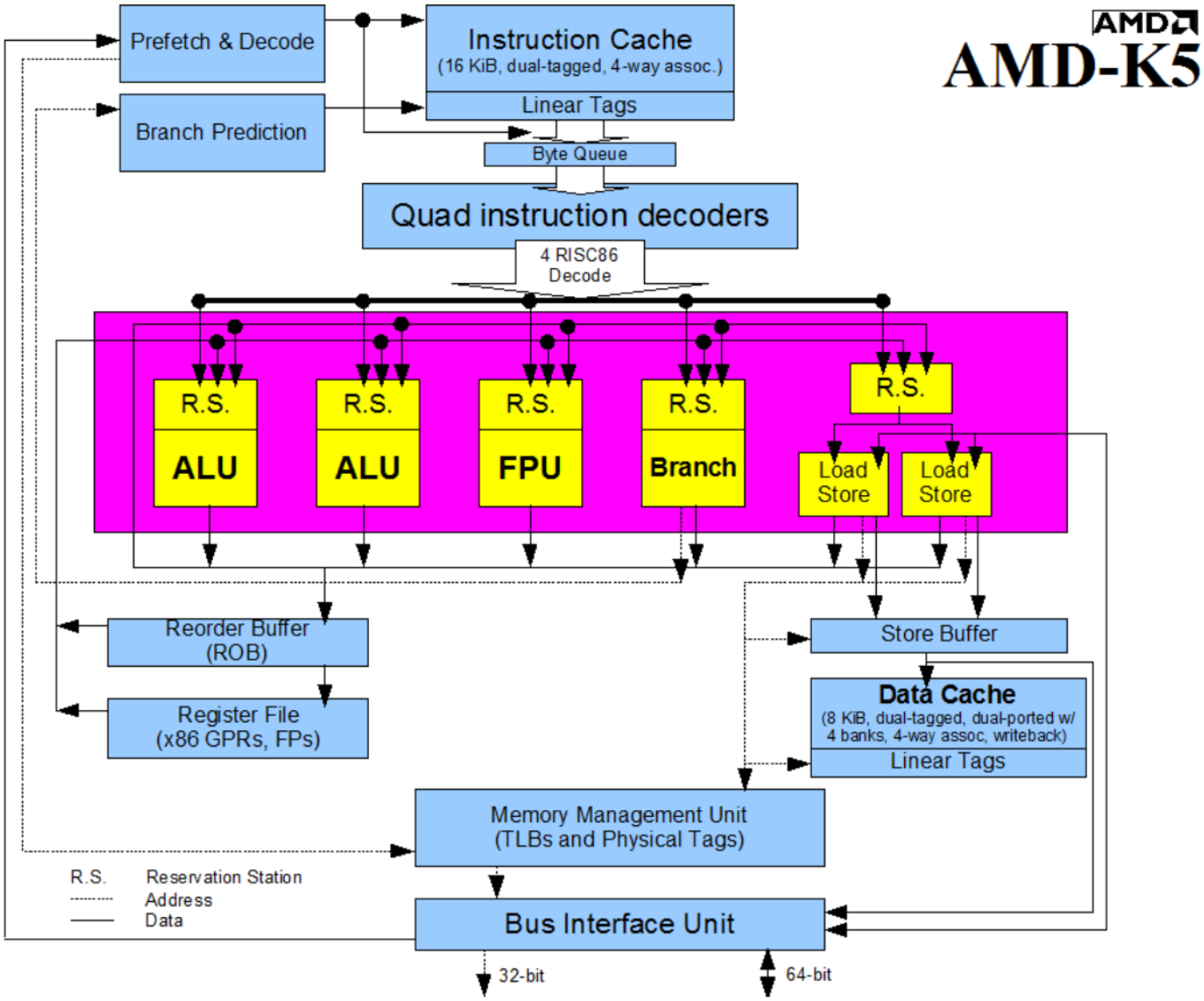
How Encumbered Are x86 CPUs in the Modern Era?

The past is never dead. It's not even past. — William Faulker

It's time to pull this discussion into the modern era and consider what the implications of this "RISC versus CISC" comparison are for the ARM and x86 CPUs actually shipping today. The question we're really asking when we compare AMD and Intel CPUs with Apple's M1 and future M2 is whether there are historical x86 bottlenecks that will prevent x86 from competing effectively with Apple and future ARM chips from companies such as

x86 uArch

Now, compare the Pentium against the AMD K5.



x86 uArch

EXTREMETECH

The P6 microarchitecture was the first Intel microarchitecture to implement out-of-order execution and a native x86-to-micro-op decode engine. P6 was shipped as the Pentium Pro and it evolved into the Pentium II, Pentium 3, and beyond. It's the grandfather of modern x86 CPUs. If anyone ought to know the answer to this question, it would be Colwell, so here's what he had to say:

> Intel's x86's do NOT have a RISC engine "under the hood." They implement the x86 instruction set architecture via a decode/execution scheme relying on mapping the x86 instructions into machine operations, or sequences of machine operations for complex instructions, and those operations then find their way through the microarchitecture, obeying various rules about data dependencies and ultimately time-sequencing.
>
> The "micro-ops" that perform this feat are over 100 bits wide, carry all sorts of odd information, cannot be directly generated by a compiler, are not necessarily single cycle. But most of all, they are a microarchitecture artifice — RISC/CISC is about the instruction set architecture... The micro-op idea was not "RISC-inspired", "RISC-like", or related to RISC at all. It was our design team finding a way to break the complexity of a very elaborate instruction set away from the microarchitecture opportunities and constraints present in a competitive microprocessor.

Section

Intel

Intel Micro-architecture

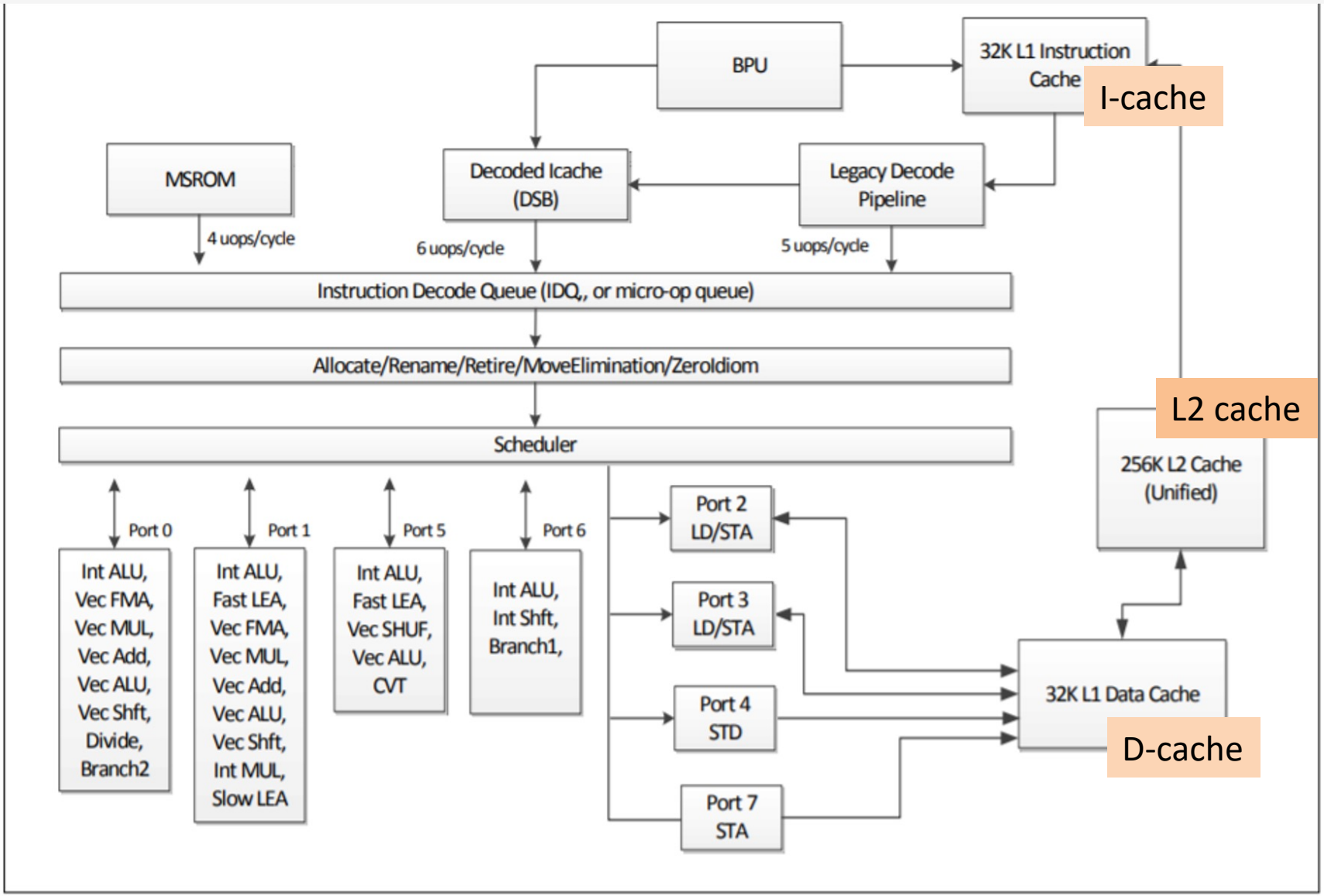
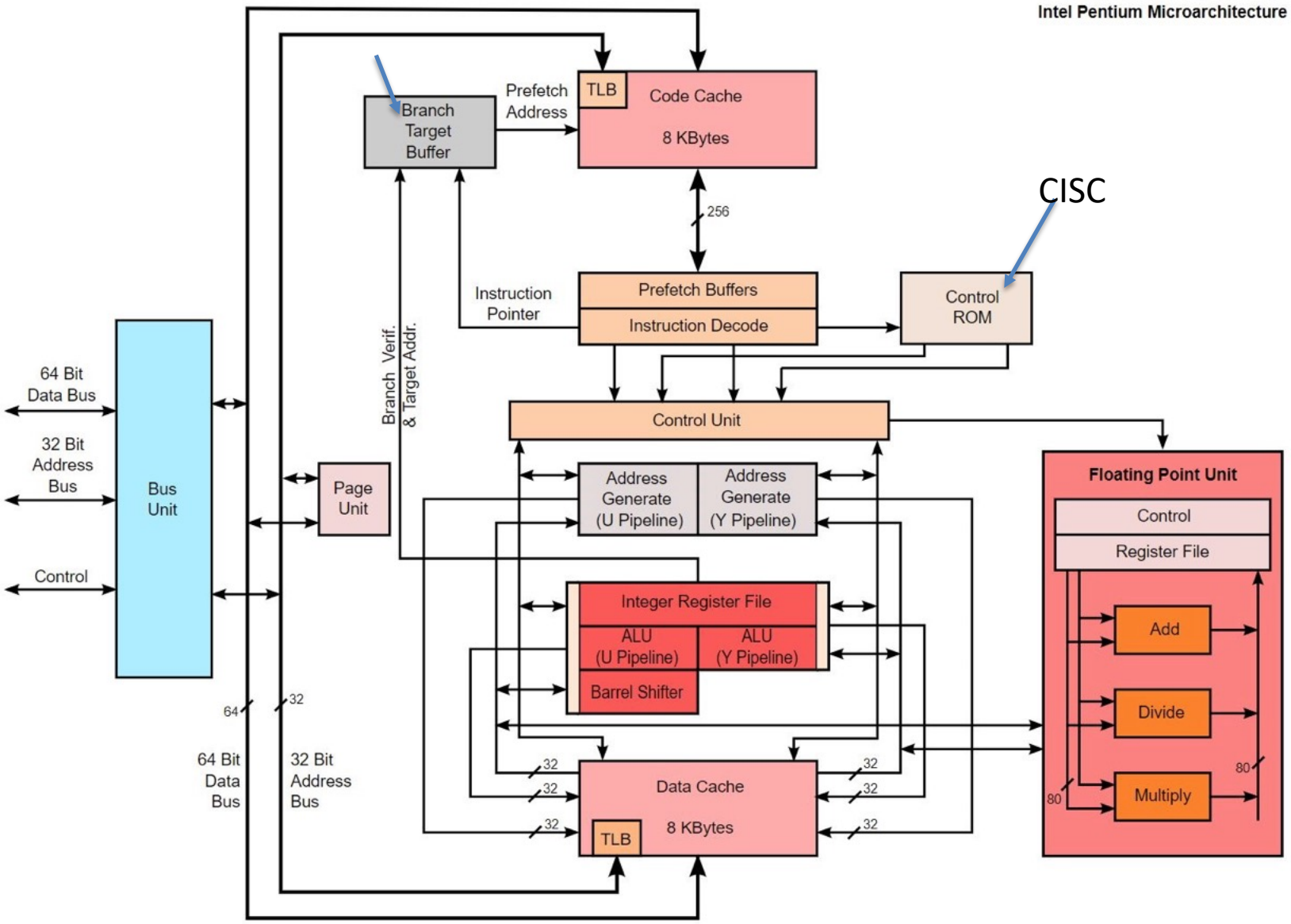


Figure 2-1. CPU Core Pipeline Functionality of the Skylake Microarchitecture

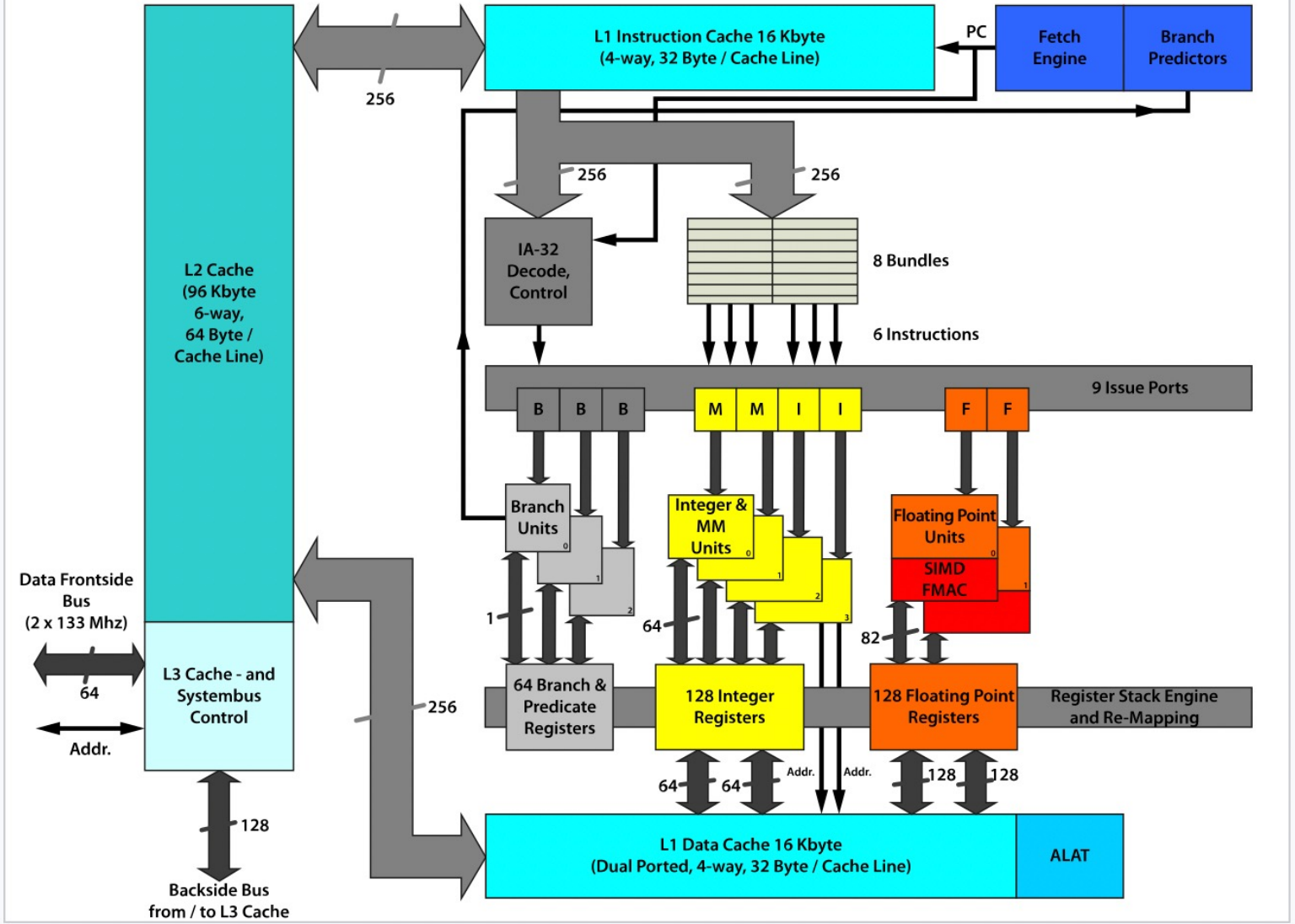
Intel uArch

EXTREMETECH



Intel uArch

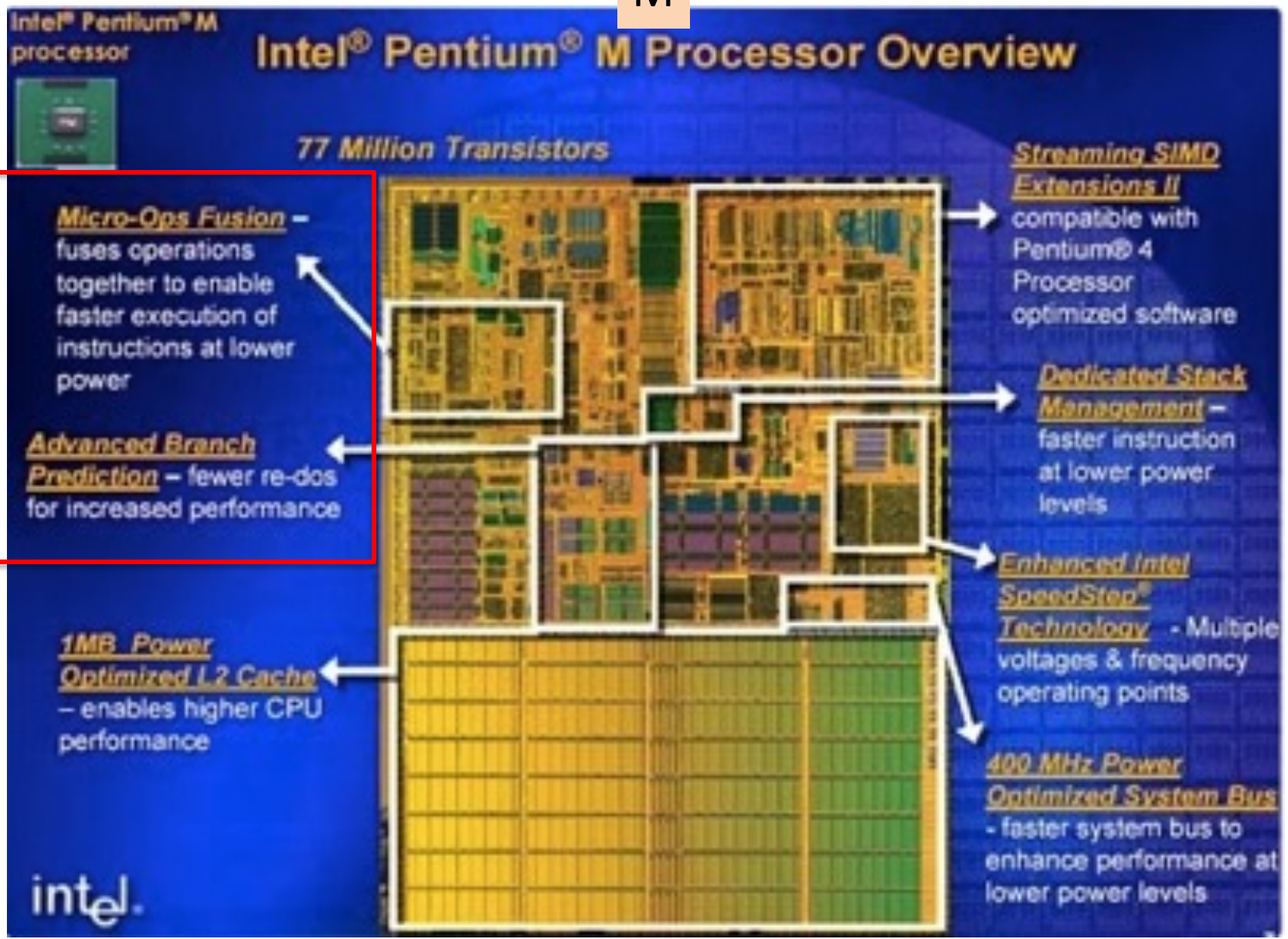
Itanium



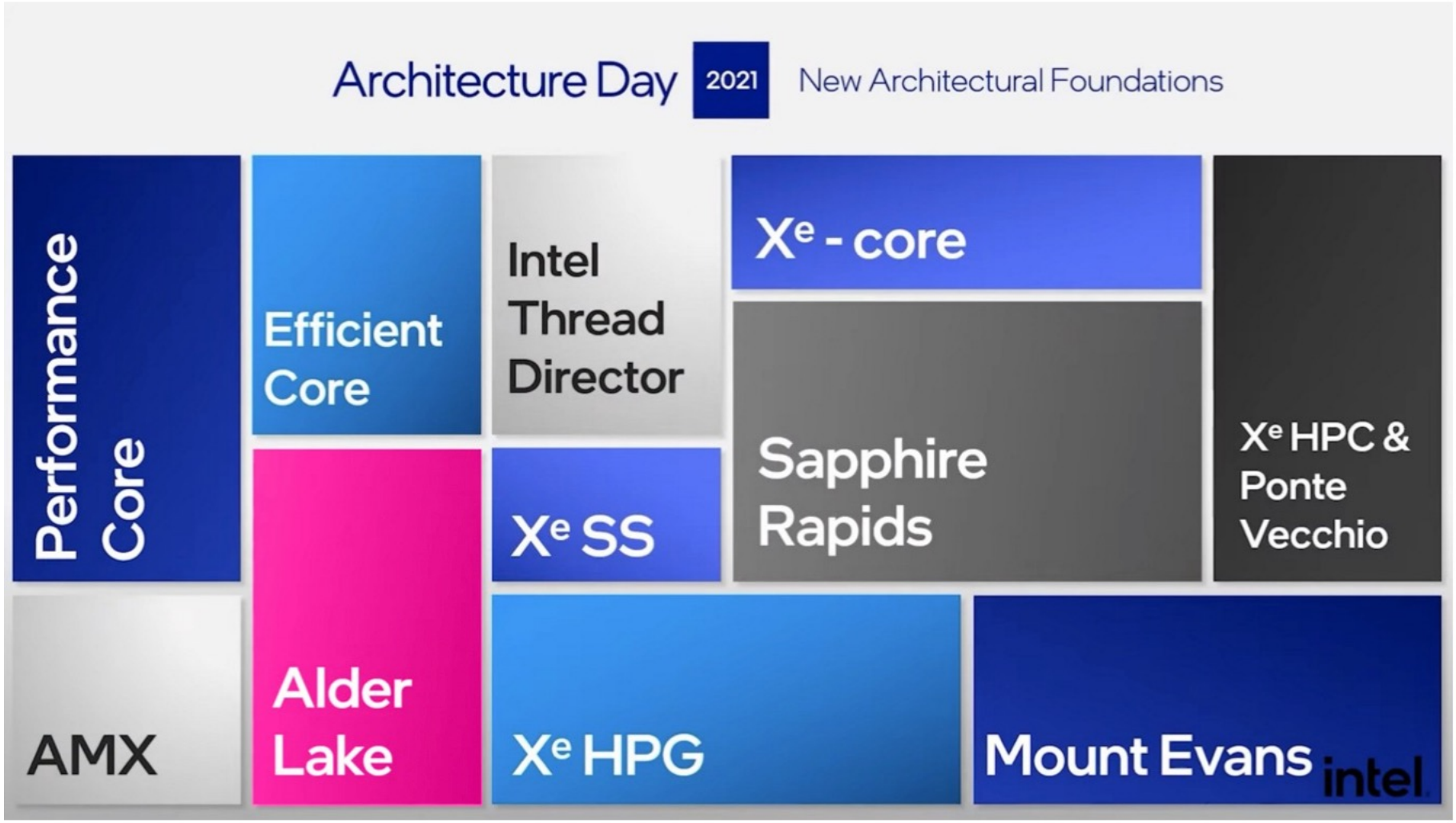
The Intel Itanium architecture

Intel uArch

M



Intel Architecture Day



Source: Intel Architecture Day (highlight added by author)

Architecture Day 2021

After claiming that the Golden Cove represents Intel's largest architectural shift in over a decade, Yoaz went on to provide additional architecture details, including:

- To reduce latency the length decode was doubled to run at 32 bytes per cycle and 2 decoders were added (increasing from 4 to 6)
- Double the number of 4K pages stored in iTLB to better support software with a large code footprint
- A 2x+ branch target buffer that utilizes a machine learning algorithm for improved branch prediction and reduced jump mispredicts
- A wider, deeper, and smarter out-of-order engine
- Enhanced integer execution units with increased single-cycle operations
- Improved memory subsystem with reduced effective load latency and increased parallelism

Architecture Day 2021

Robinson went on to provide additional microarchitecture details, including:

- Improved branch prediction with a 5,000 entry branch target cache
- 64KB instruction cache
- Intel's first on-demand instruction length decoder, which enables bypassing the length decoder on subsequent execution, increasing performance while saving power
- Hardware-driven load balancing that extracts parallelism via a 256-entry out-of-order window and seventeen execution ports.
- A dual load + dual store memory subsystem

Architecture Day 2021

Section

AMD

AMD vs Intel: CPU Families

Market Segment	AMD	Intel
Desktop	Ryzen 4K/Athlon 3K	Core i7/i9 (10th gen)
Laptop	Ryzen 4000	Ice Lake
Gaming	Ryzen Threadripper +Radeon	Core Extreme
Server/Workstn	Epyc	Xeon

According to the company, the AMD Ryzen 4700 G series desktop processor offers up to 2.5x multi-threaded performance compared to the previous generation, up to 5% greater single-thread performance than the Intel Core i7-9700, up to 31% greater multithreaded performance than the Intel Core i7-9700, and up to 202% better graphics performance than the Intel Core i7-9700.

AMD's Zen

AMD News

AMD's new Zen Processor

AMD Zen

stock news usa (2016)

(Click image to view full size)

AMD made radical alterations to its Zen design while keeping itself distant from an ugly past. The company knew it had to make the changes to become a force to reckon with in the server and PC markets. So when the designers of the chip sat down to map the Zen design, they had two priorities: To boost CPU performance to maximum and to stabilize power efficiency.

According to a company spokesperson, the chips will come with 8 to 32 cores. The 32-core chips may come in the quad-CPU configurations although those details haven't been finalized yet.

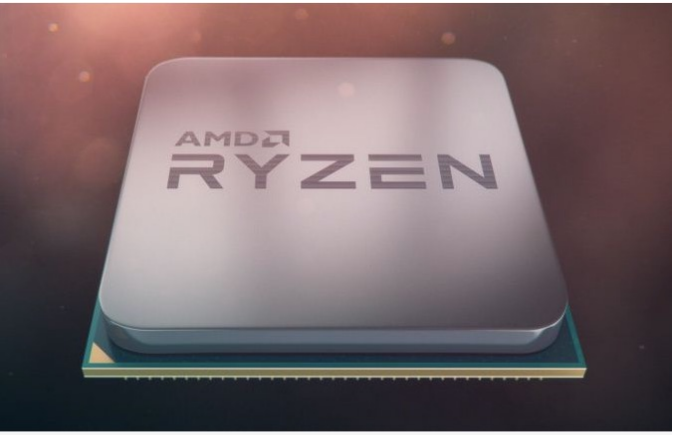
5-19B Transistors

- CPU performance
- Power efficiency

8-32 cores

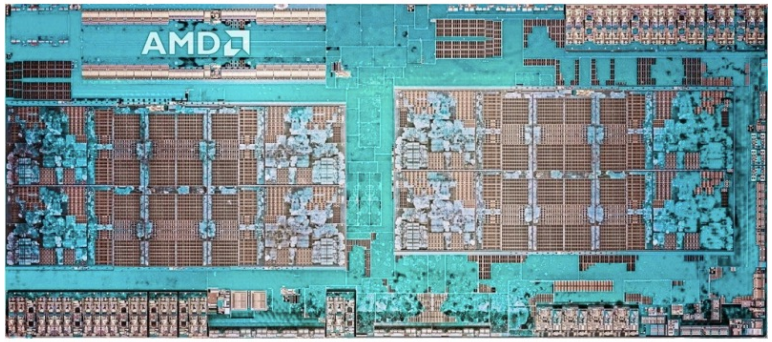
Source: stocknewsusa (2016-08-26)

Inside AMD's Production Of The Zen CPU



Ryzen 7 will have three CPUs to start, all having eight cores and supporting simultaneous multi-threading:

- *Ryzen 7 1800X: 8C/16T, 3.6 GHz base, 4.0 GHz turbo, 95W, $499*
- *Ryzen 7 1700X: 8C/16T, 3.4 GHz base, 3.8 GHz turbo, 95W, $399*
- *Ryzen 7 1700: 8C/16T, 3.0 GHz base, 3.7 GHz turbo, $329*

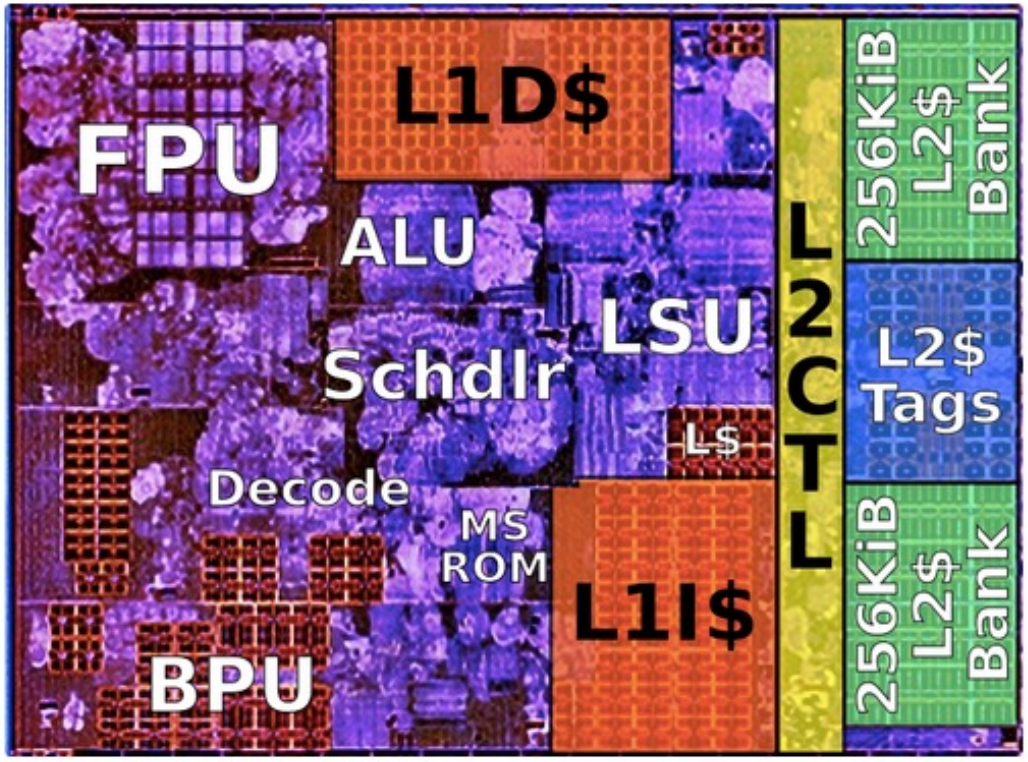


Ryzen

4.8 billion transistors and more than 2,000m of signal wire

AMD Zen Cores

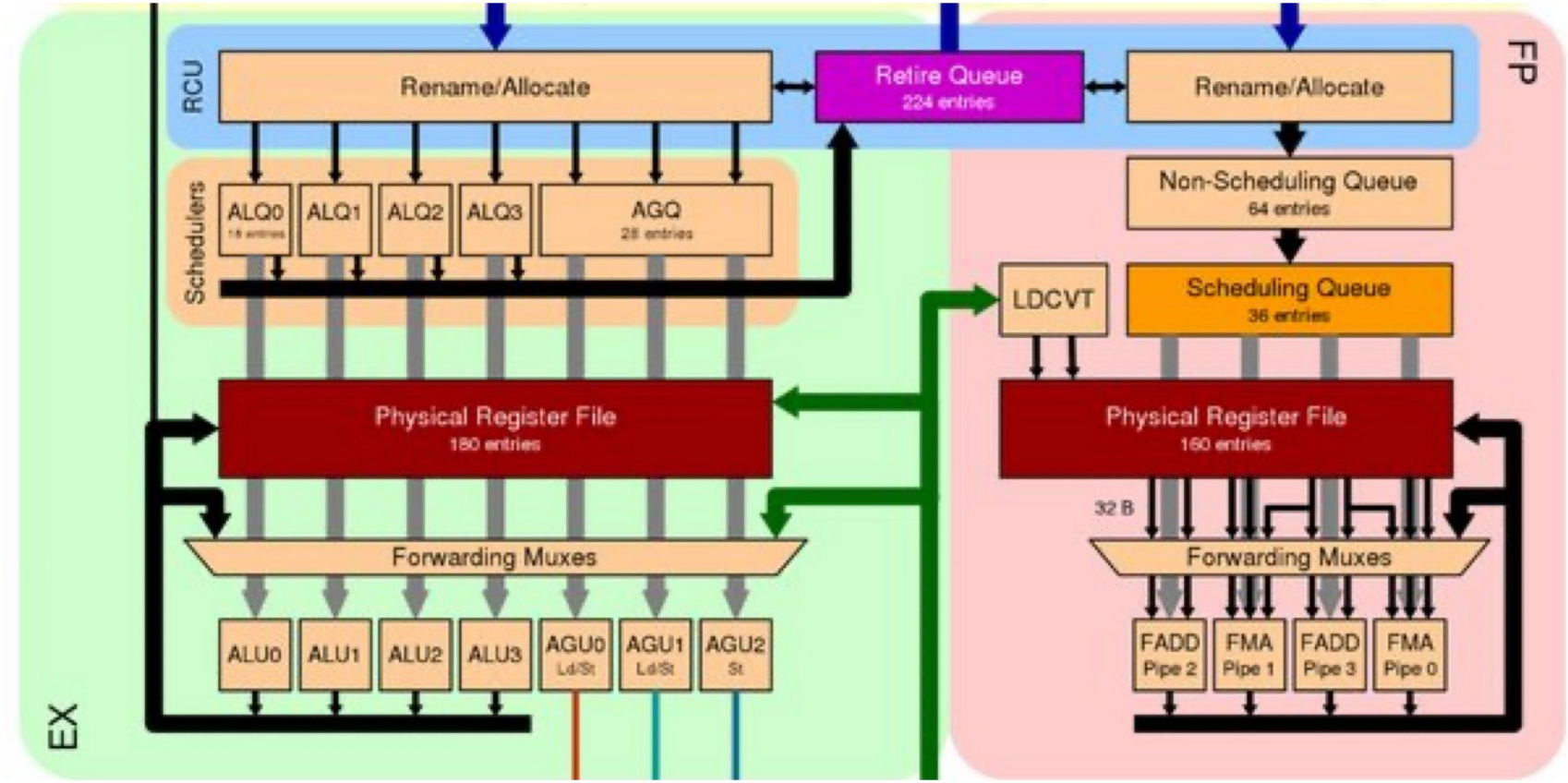
To make the CPUs execute the serial code as quickly as possible, they have very big and complex cores which have things like very advanced branch predictors and huge caches to minimize time wasted for stalls. The actual execution units are only very small part of the transistor count or area of the core.



Here is a picture of AMD Zen core. The very small part "ALU" consists all the execution units for most commonly used integer instructions, and FPU is mostly the execution units of the floating point and SIMD instructions. L1D$, L1I$ and L2$ are cache memory, BPU is branch prediction unit which just tries to guess what the core should do next to minimize stalls.

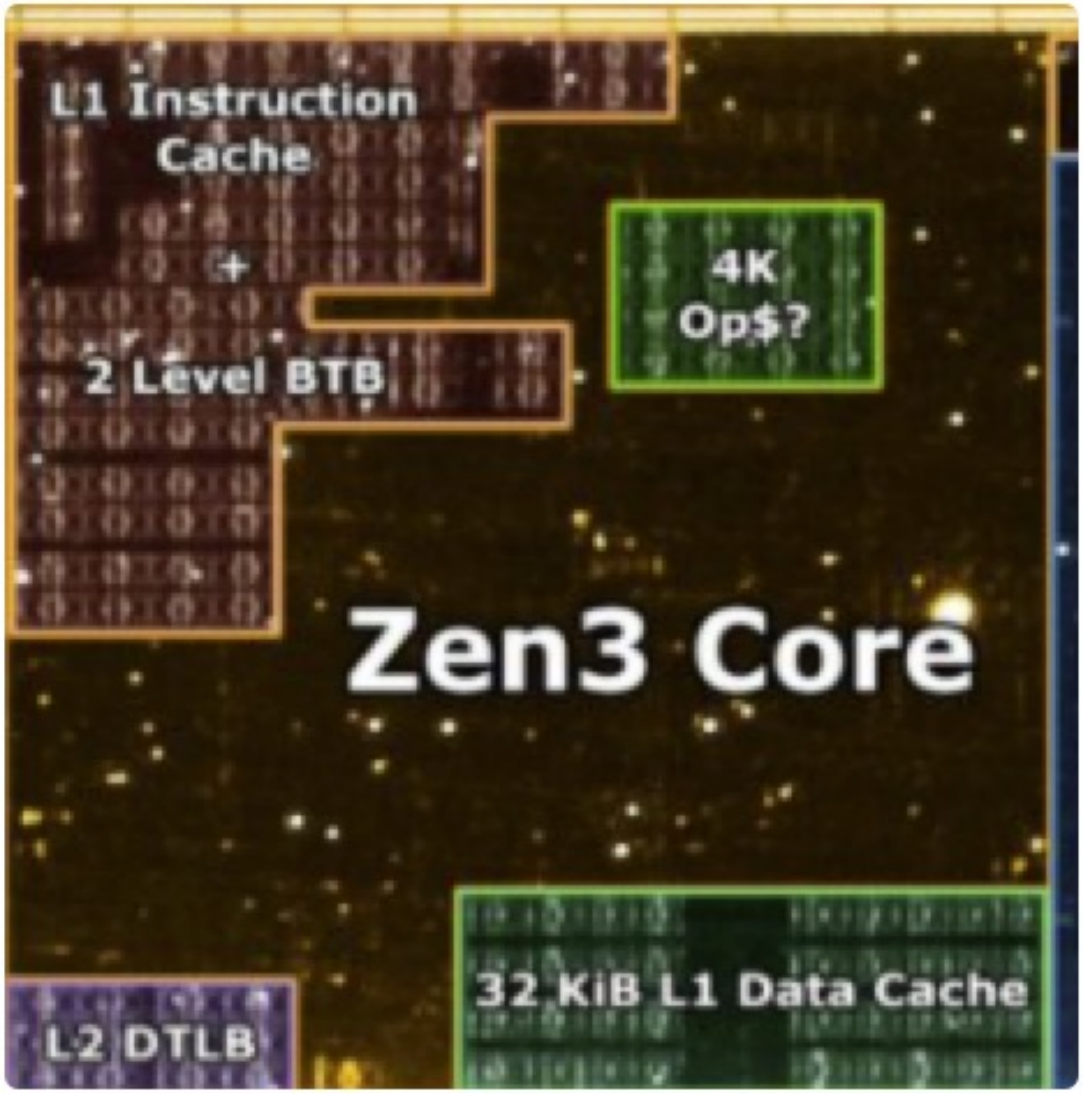
AMD Zen 2

This is AMD Zen2 executor:



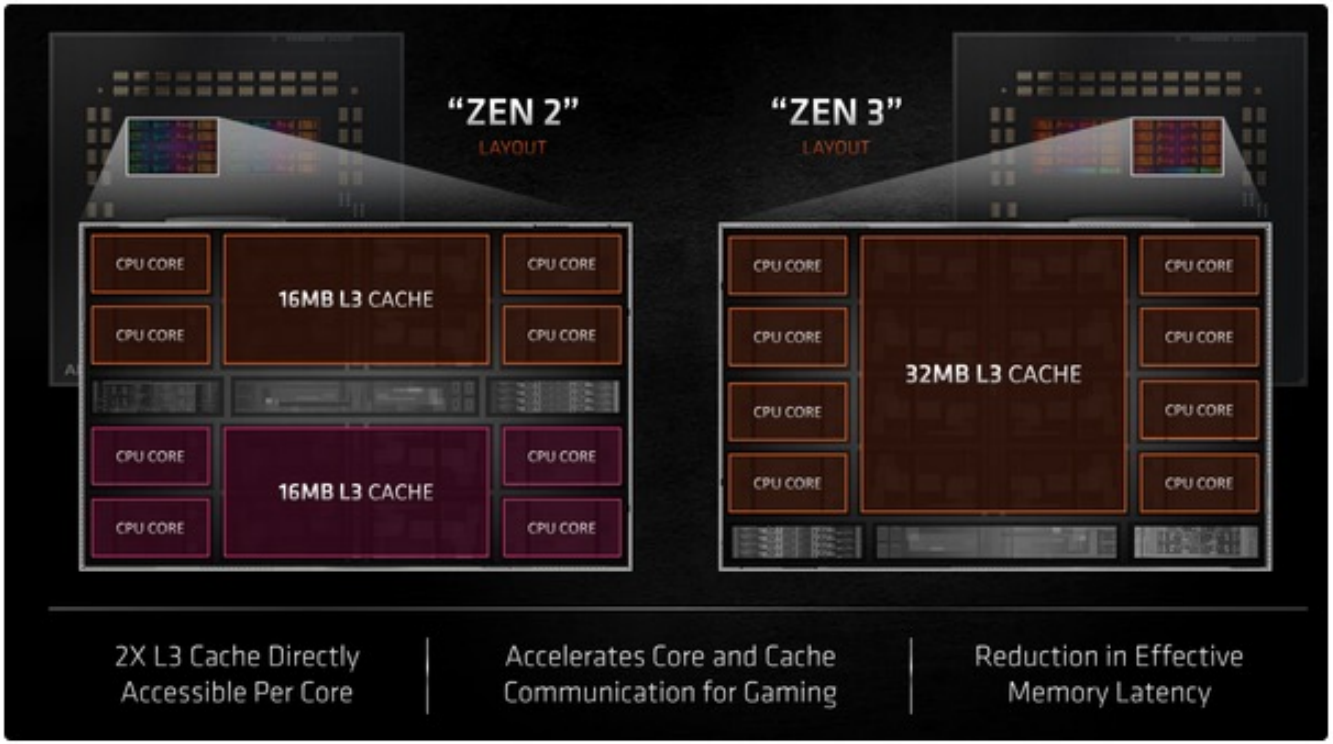
As could be see it has separate integer and floating point blocks where integer one has 4/5 units in parallel. In other words, AMD CPUs are capable of executing 4/5 integer operations in parallel. Add to this 4 floating point in parallel

AMD Zen 3 Die



AMD Zen L3 Cache

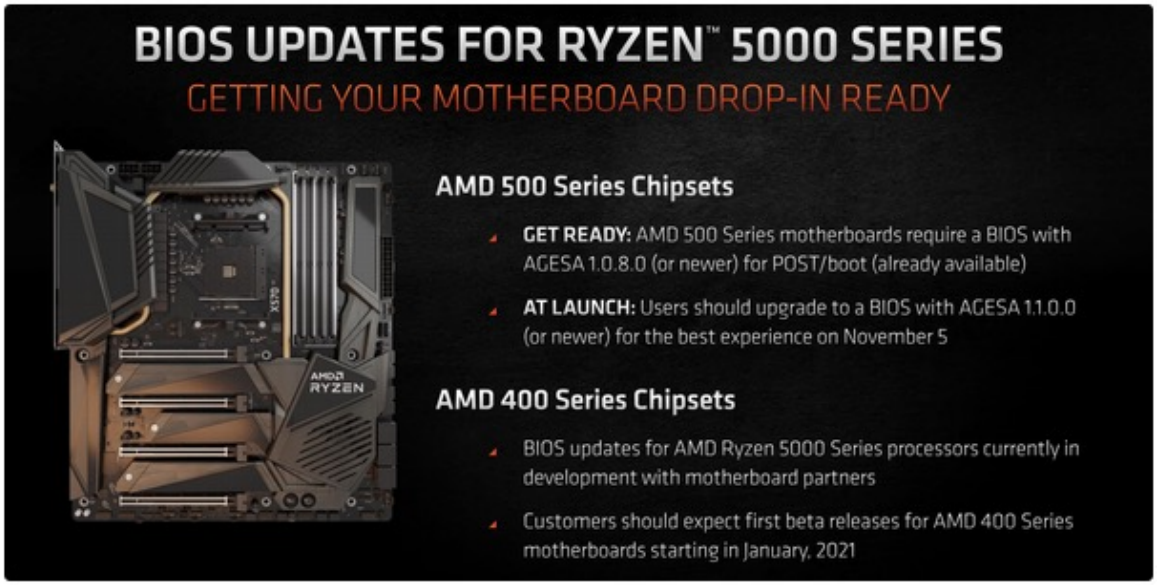
One area that AMD has lagged behind Intel over the lifetime of the Zen brand is in gaming performance. It's no secret that in the company's push to lower the cost per core of its flagship processors (through the introduction of chiplet-based architectures), the design decisions have resulted in more latency between core complexes. That manifests itself in reduced performance in certain PC gaming scenarios--especially at the favored 1080p resolution used by most gamers.



This is down to how chips are designed, and, more specifically, how they're laid out on

AMD Zen 3

Another way that AMD looks poised to continue its winning streak over Intel is in its platform and required socket adoption for new PCs, PC builds, and upgrades...or rather, its of required adoption. Instead of forcing buyers onto a new motherboard platform with a new style of CPU socket every other generation of chips (the typical cadence in recent years for Intel's desktop processors), Zen 3 will mark the third launch in the Zen line to feature some level of compatibility with motherboards based on the now-venerable Socket AM4.

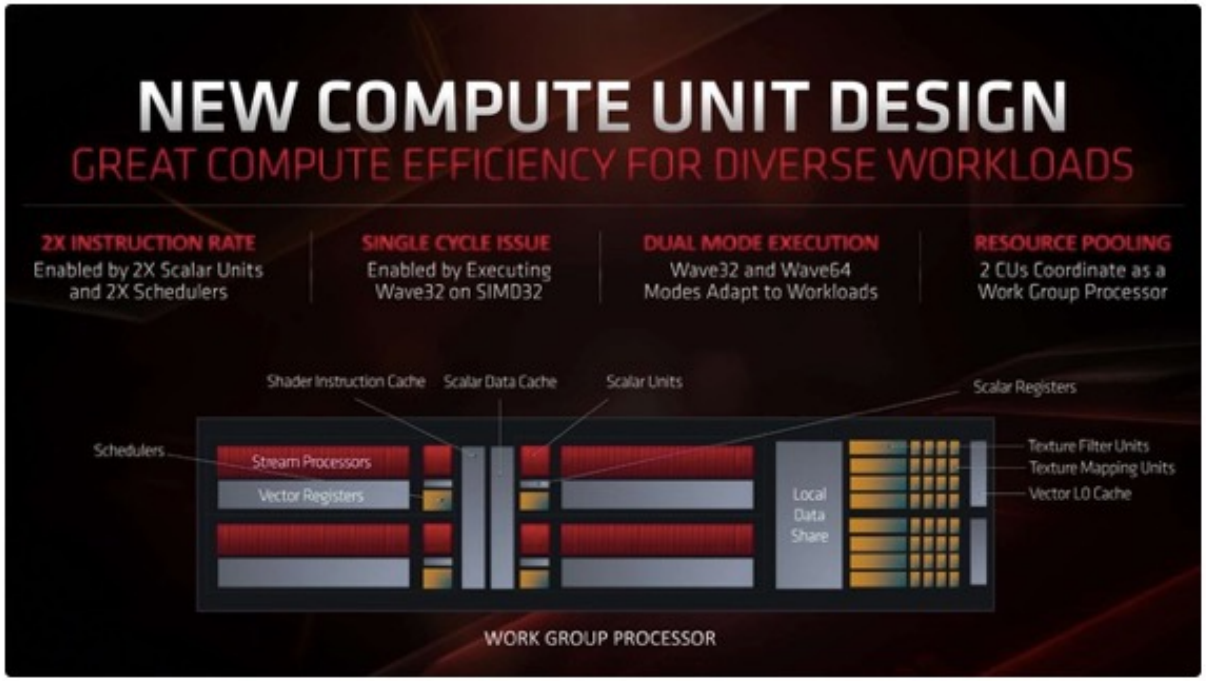


The one caveat? Unlike Zen 2, which is compatible with just about AM4-based motherboard, the cutoff for Zen 3 is a bit higher up the chipset stack this time. The new CPUs will work only with motherboards from the X470, B450, and later chipset generations. (That includes the new X570 and B550 boards.) Plus, it's down to motherboard manufacturers to make it work, issuing the proper updates.

Back in May, the company clarified Zen 3 AM4 compatibility, claiming that a BIOS update would be required for any users of either X470 or B450 motherboards. Now, we

AMD Zen Cores

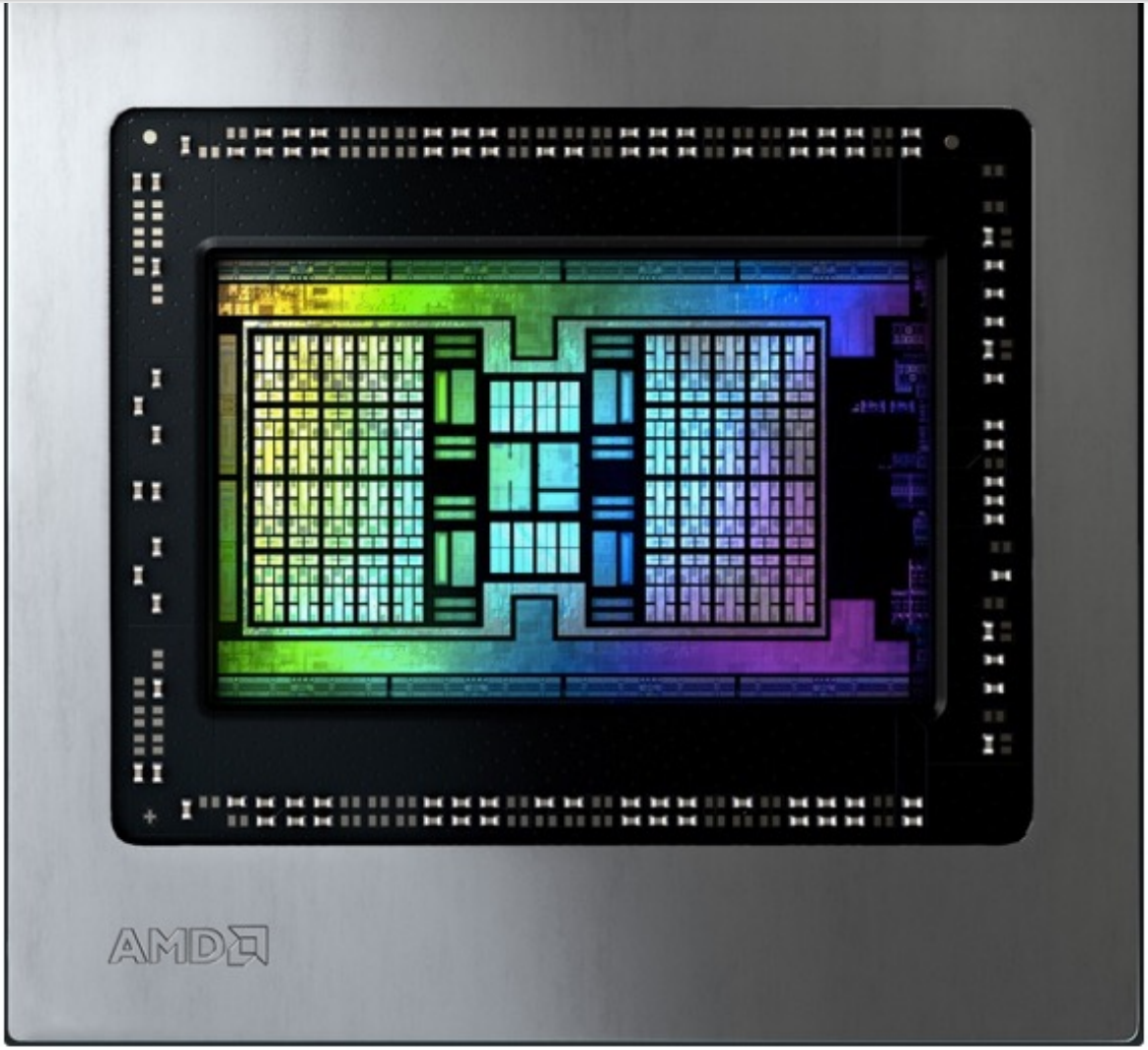
Each core (DCU) has four 32-lane SIMD units, so that it each core can perform 128 32-bit floating point multiply-accumulations(FMAs) per clock cycle and the whole chip 5120 32-bit FMAs per clock cycle.



But each Zen2 or Zen3 CPU core also has 2 256-bit (8-lane for 32-bit) SIMD FMA units, so that each core is capable of calculating 16 32-bit FMAs per clock cycle, so for 8-core CPU such as 3800X or 5800X, the whole chip can perform 128 32-bit FMAs per clock cycle.

So, the core count difference between CPUs and GPUs is not many hundreds of times in reality, it's less than 10 times, and the difference in parallel execution units is only about 20-40 times, not hundreds of times.

AMD Zen Cores

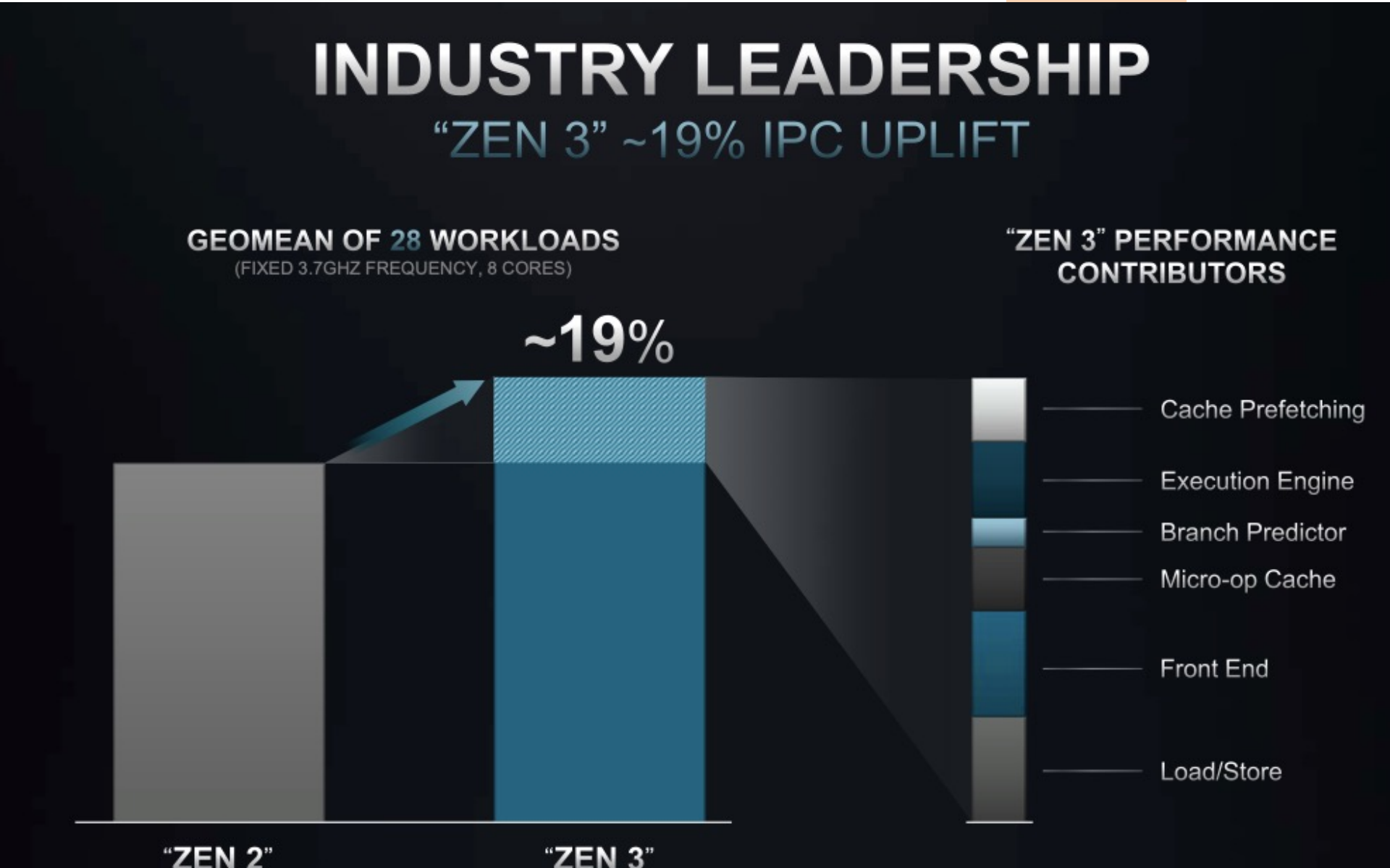


Each core (DCU) has four 32-lane SIMD units, so that it each core can perform 128 32-bit floating point multiply-accumulations(FMAs) per clock cycle and the whole chip 5120 32-bit FMAs per clock cycle.

Zen 3

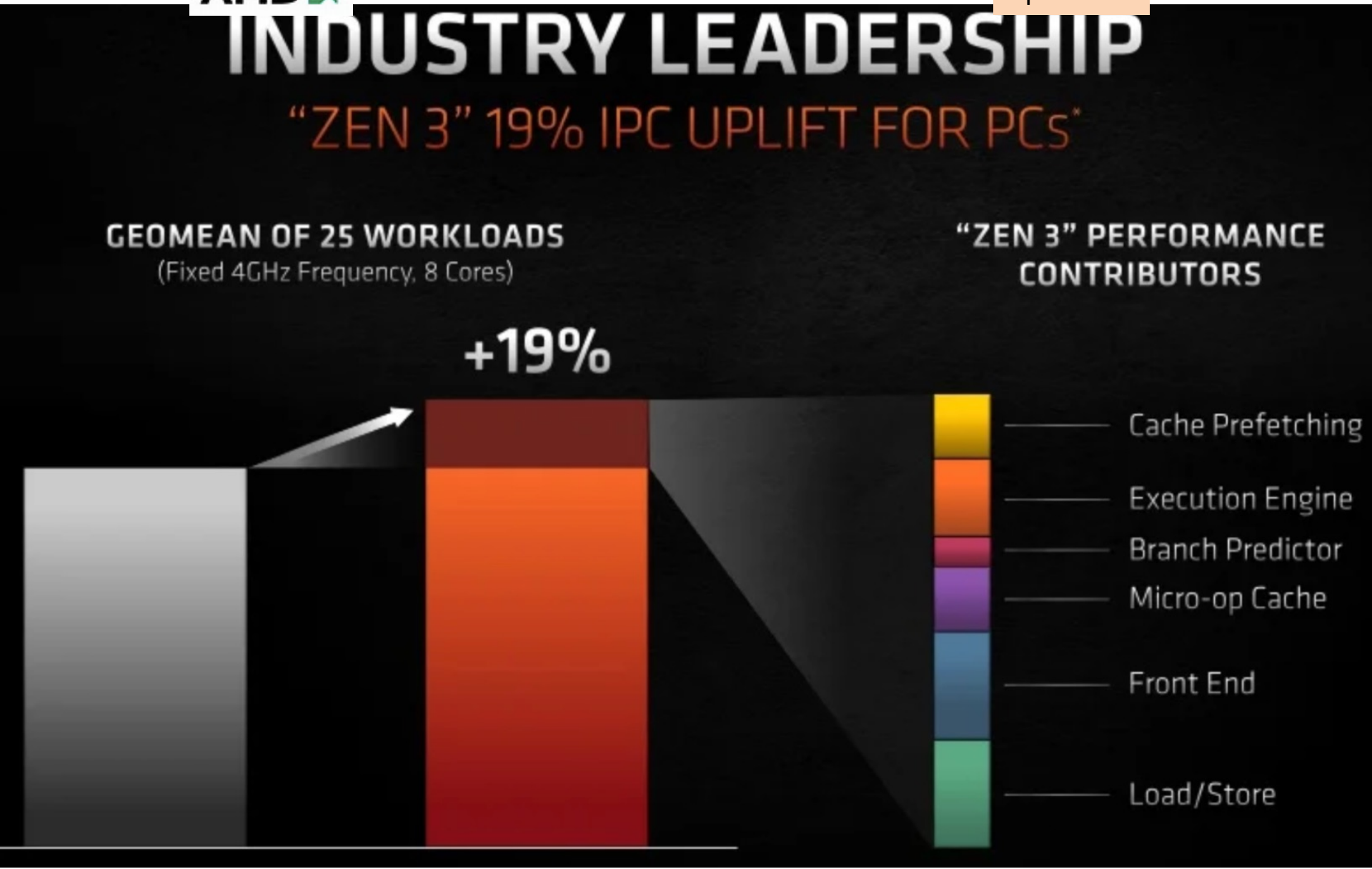
AMD

March 2021



Zen 3

April 2022



Zen 3

AMD

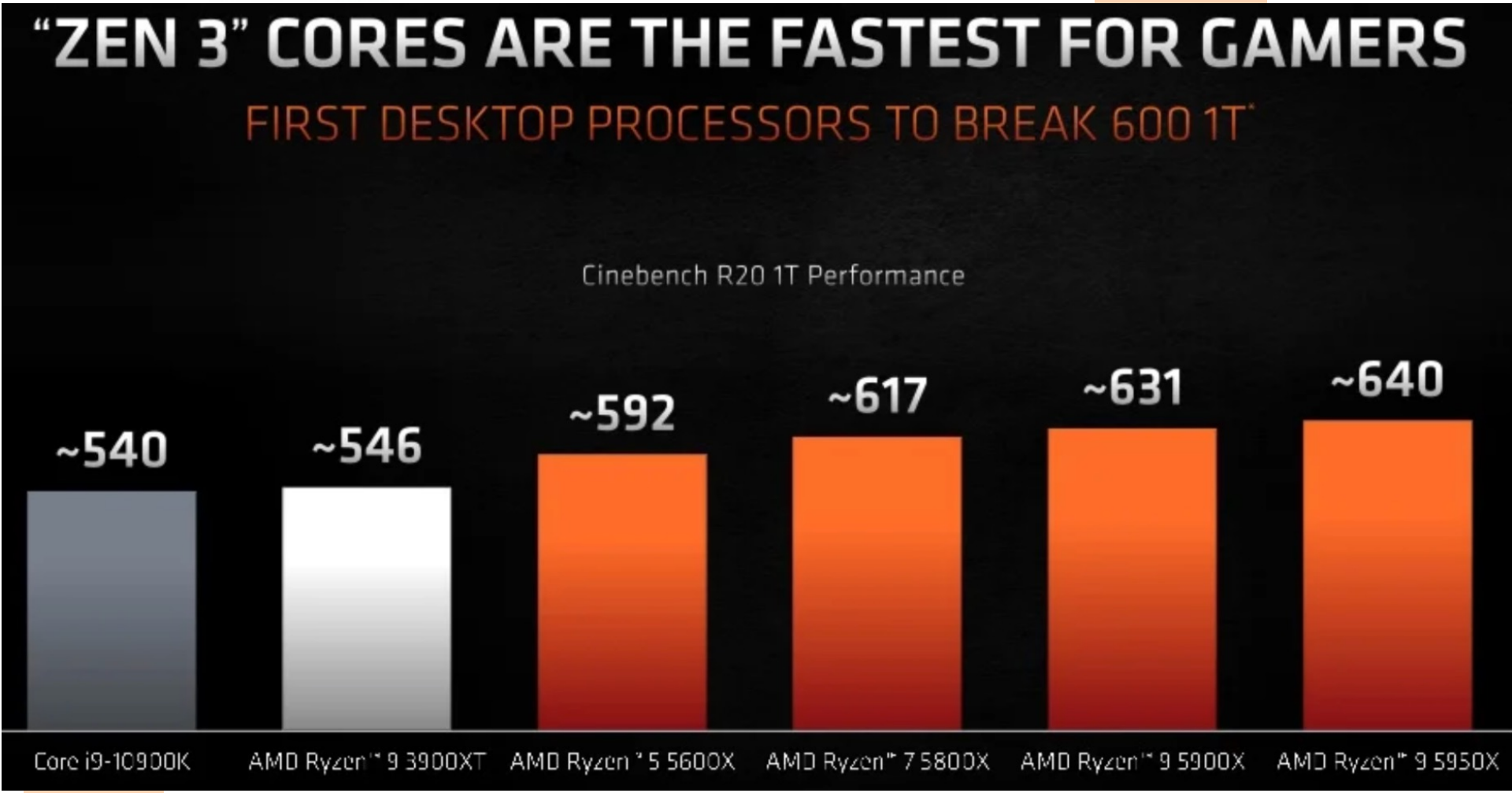
April 2022

Core i9

Zen 3

AMD

April 2022

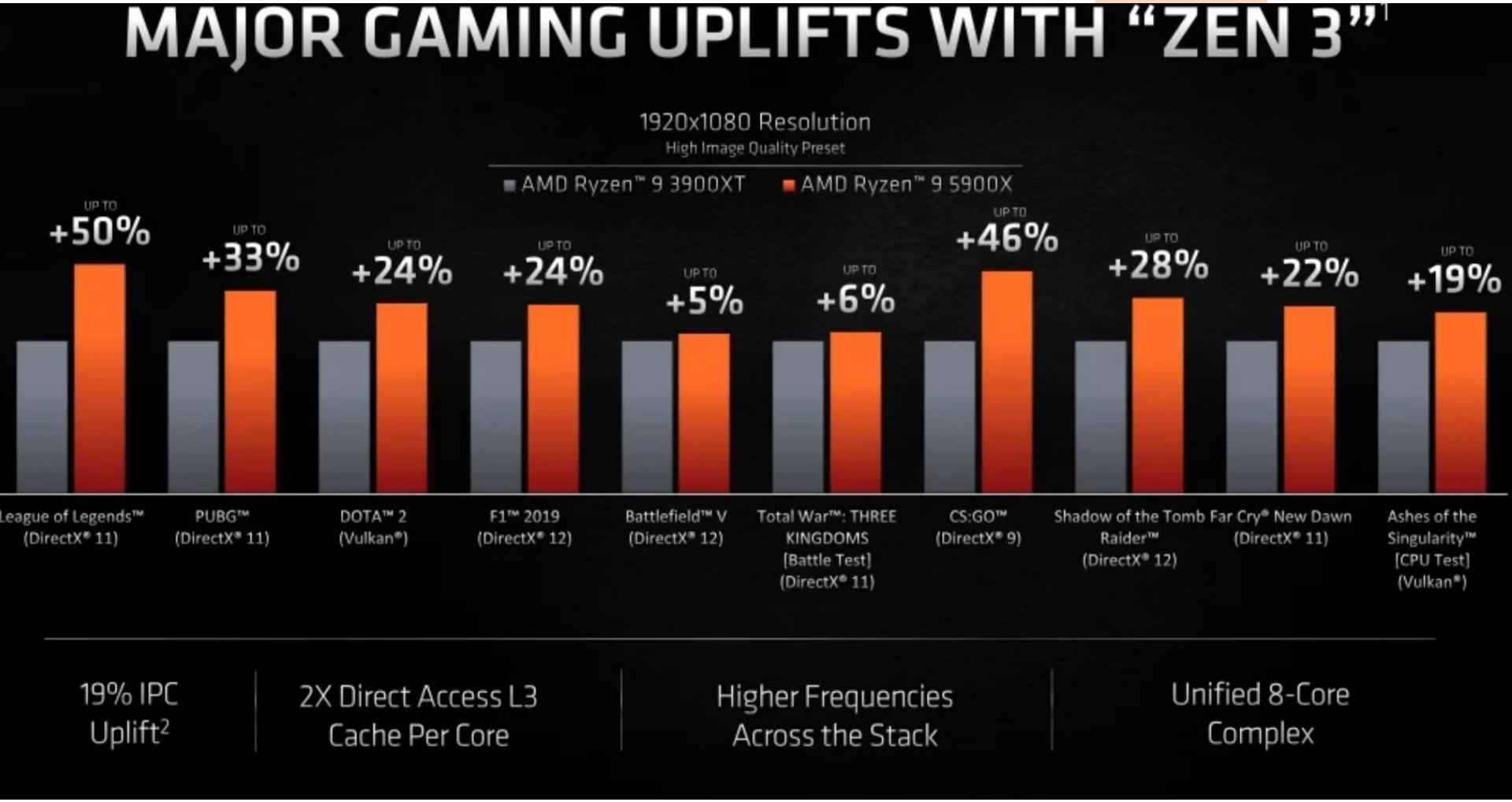


Core i9

Zen 3

AMD

April 2022



Zen 3 uArch

AMD

April 2022

"ZEN 3" OVERVIEW

2 THREADS PER CORE (SMT)

STATE-OF-THE-ART BRANCH PREDICTOR

CACHES
- I-cache 32k, 8-way
- Op-cache, 4K instructions
- D-cache 32k, 8-way
- L2 cache 512k, 8-way

DECODE
- 4 instructions / cycle from decode or 8 ops from Op-cache
- 6 ops / cycle dispatched to Integer or Floating Point

EXECUTION CAPABILITIES
- 4 integer units
- Dedicated branch and store data units
- 3 address generations per cycle

3 MEMORY OPS PER CYCLE
- Max 2 can be stores

TLBs
- L1 64 entries I & D, all page sizes
- L2 512 I, 2K D, everything but 1G

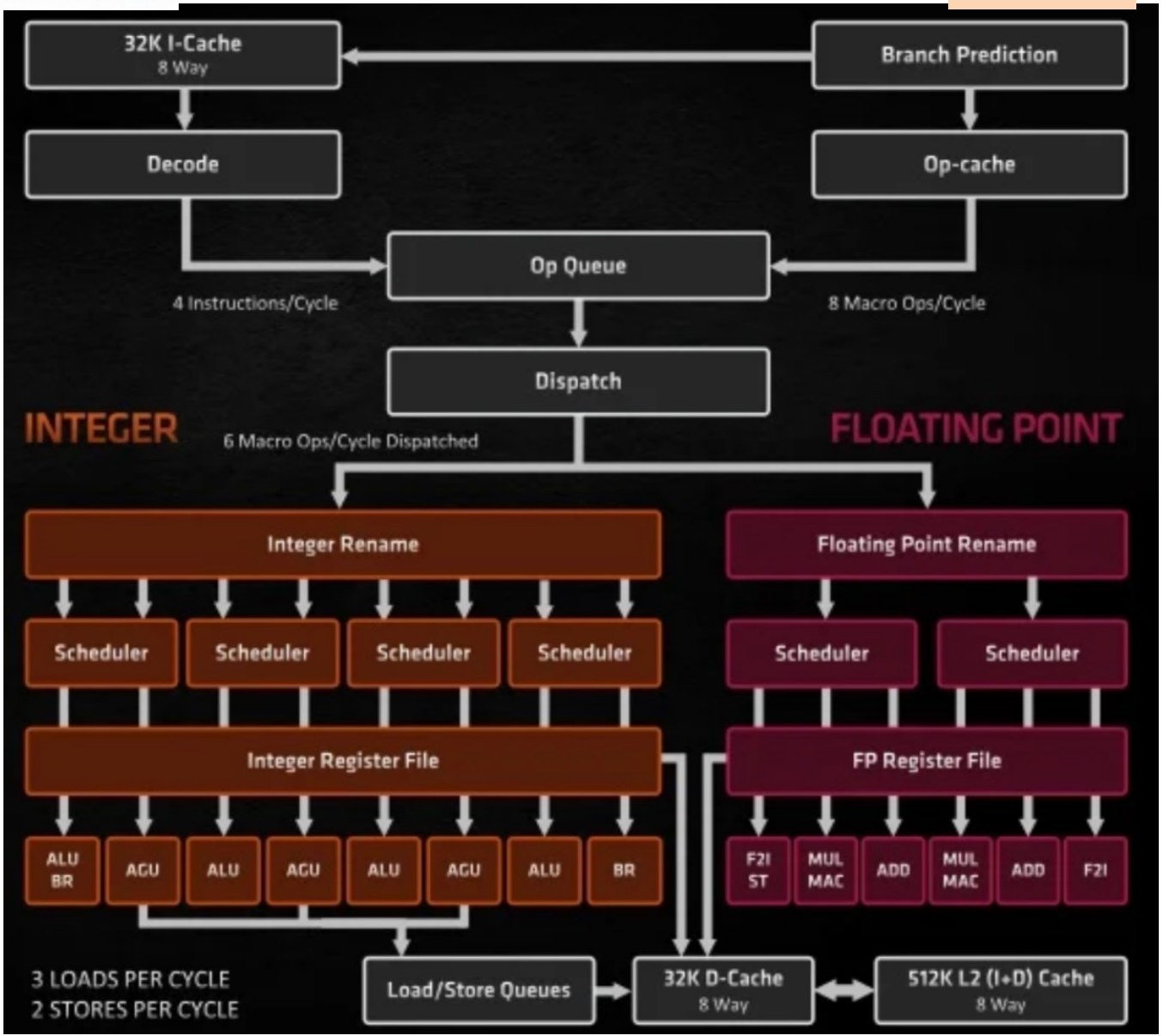
TWO 256-BIT FP MULTIPLY ACCUMULATE / CYCLE

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP222

Zen 3 uArch

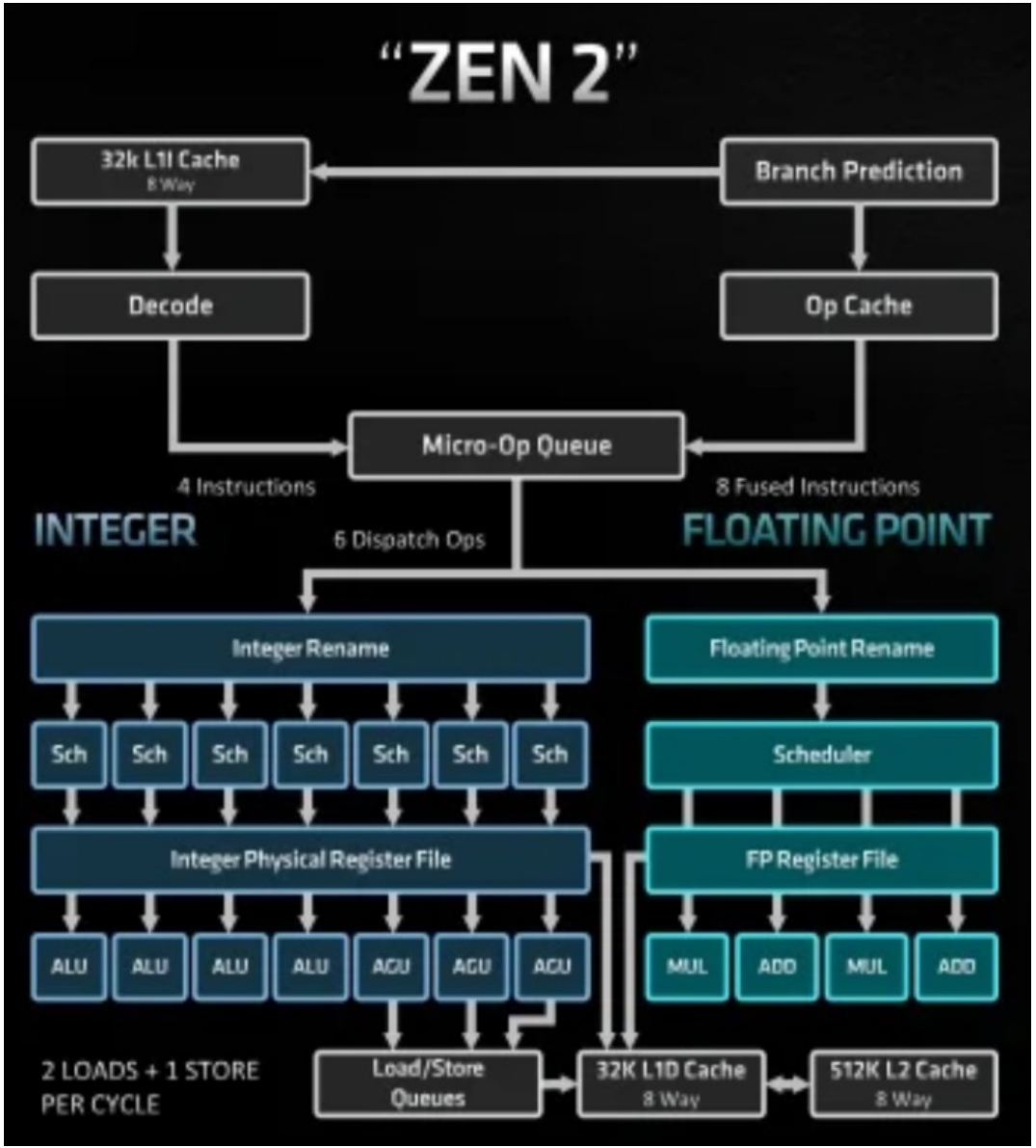
AMD

April 2022



Zen 2 uArch

AMD MAJOR CHANGES VS. "ZEN 2" April 2022



Zen 3 uArch

AMD

MAJOR CHANGES VS. "ZEN 2"

April 2022

FRONT-END ENHANCEMENTS

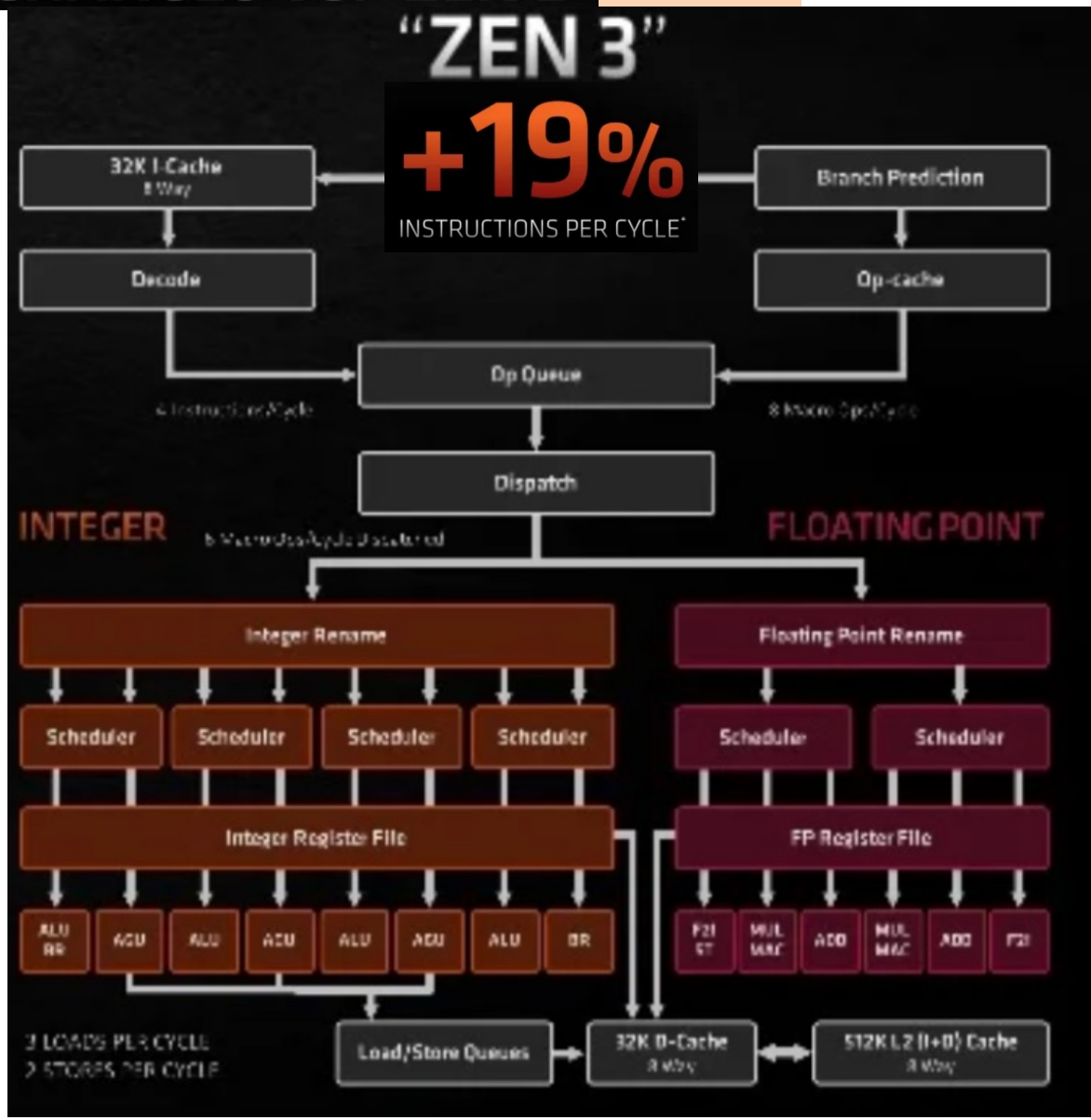
2X Larger L1 BTB (1024)
Improved branch predictor bandwidth
"No-bubble" branch prediction
Faster recovery from mispredict
Faster sequencing of Op-cache fetches
Finer-grained switching of Op-cache pipes

EXECUTION

Int: Dedicated Branch and St-data pickers
Int: Larger windows (+32)
FP/Int: Reduced latency for select ops
FP: 6-wide dispatch and issue (+2)
FP: Faster FMAC (-1 cycle)

LOAD / STORE

Higher load bandwidth (+1)
Higher store bandwidth (+1)
More flexibility in load/store ops
Improved memory dependence detection
TLB: 6 table walkers (+4)



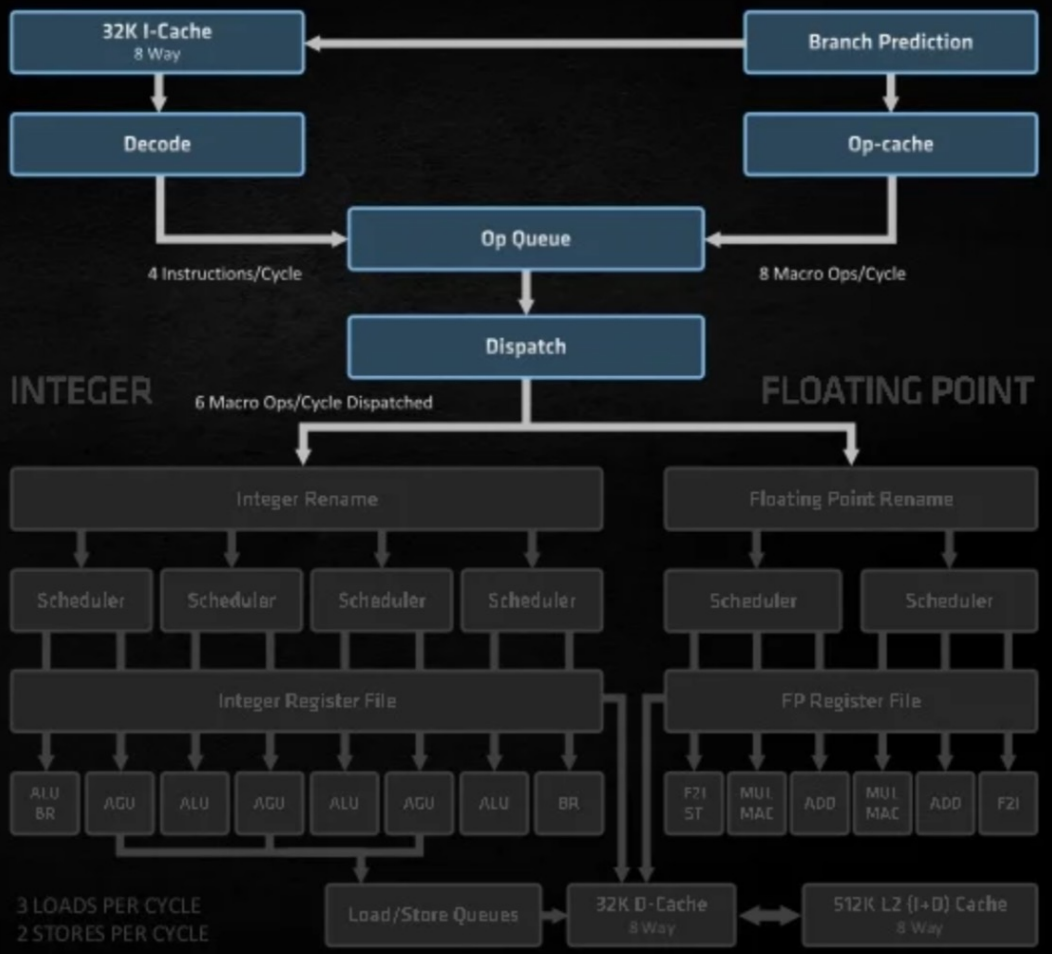
Zen 3 uArch

AMD

April 2022

FRONT-END ADVANCES

- Faster Branch Predictor
- Faster Switching Between Op-cache and Instruction Cache
- Faster Branch Mispredict Recovery
- Branch Predictor Accuracy Tweaks



Zen 3 uArch

AMD

April 2022

FETCH/DECODE

"ZEN 3" DESIGN GOAL: FASTER FETCH, ESPECIALLY FOR BRANCHY AND LARGE FOOTPRINT CODE

IMPROVED BRANCH PREDICTION

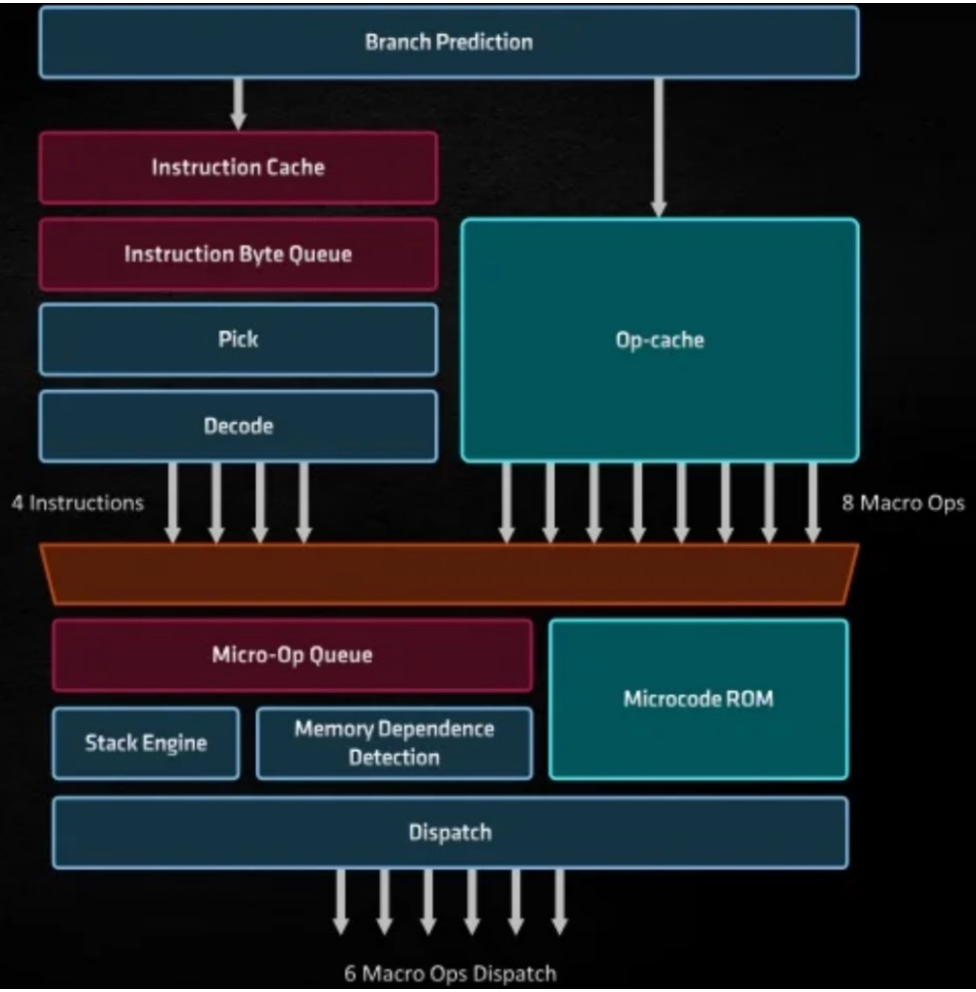
- TAGE branch predictor
- Redistributed BTBs for better prediction latency
  - L1 BTB, 1024 entries
  - L2 BTB, 6.5K entries
- Larger 1.5K indirect target array (ITA)
- Lower mispredict latency
- No "bubble" on most predictions

OPTIMIZED 32KB, 8-WAY L1I CACHE

- Improved prefetching
- Improved utilization

STREAMLINED OP-CACHE

- Faster sequencing of Op-cache fetches
- Finer-grained switching of Op-cache / I-cache pipes



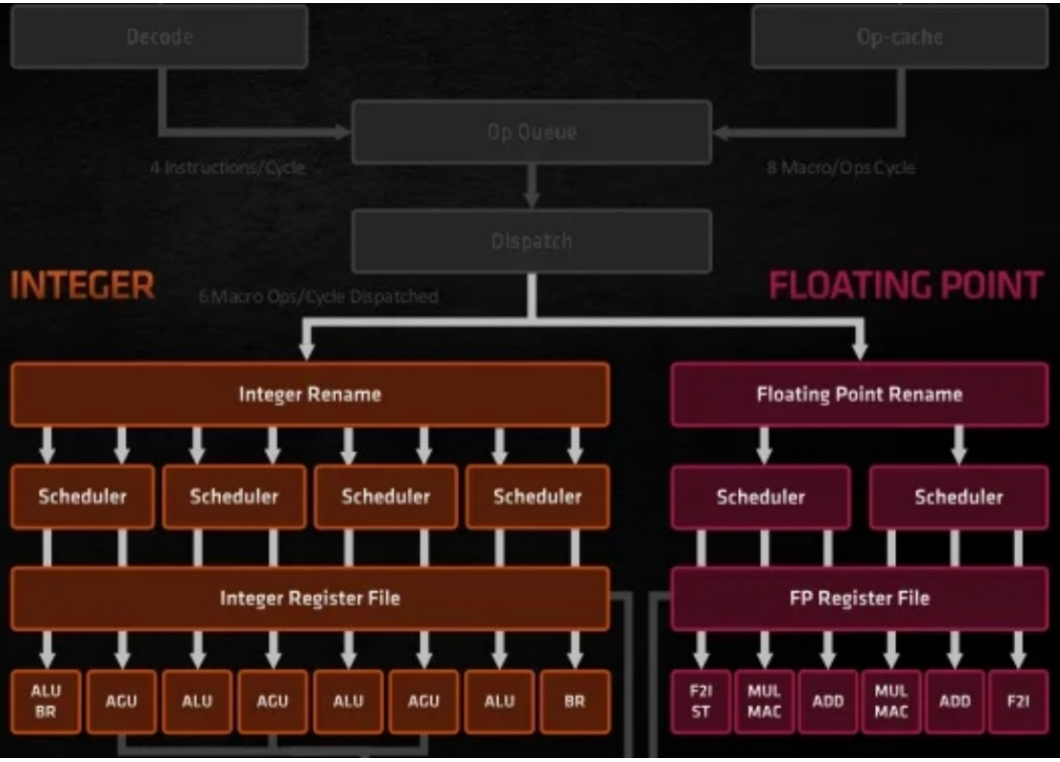
Zen 3 uArch

AMD

April 2022

EXECUTION ENGINE ADVANCES

- Wider Floating Point Issue
- Wider Integer Issue
- Faster FMAC
- Larger Execution Windows
- New integer data pickers



Zen 3 uArch

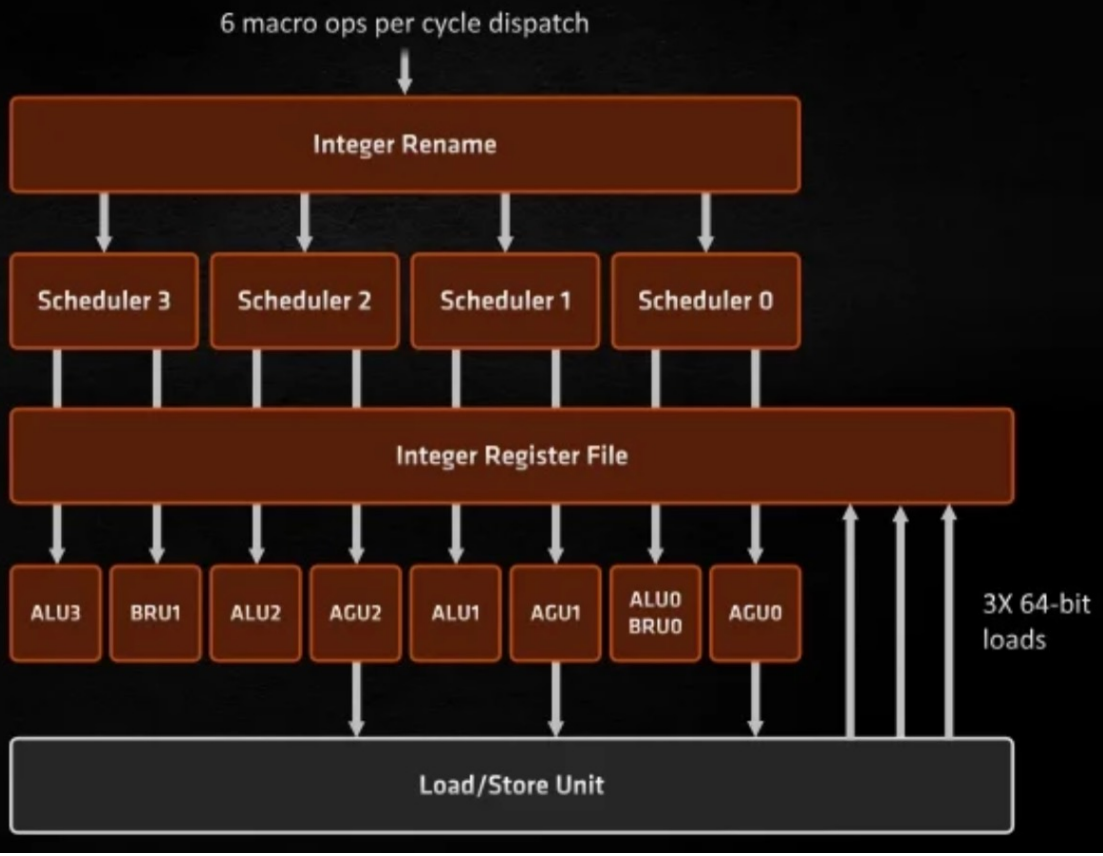
AMD

April 2022

INT EXECUTION

"ZEN 3" DESIGN GOAL: LOWER LATENCIES AND LARGER STRUCTURES TO EXTRACT ILP FOR FEEDING THE EXECUTION ENGINES

- 96 entry integer scheduler, up from 92
  - 4x 24-entry ALU/AGU schedulers
- 192 entry physical register file (up from 180)
- 10 issue per cycle, up from 7
  - 4 ALUs, 3 AGUs, 1 dedicated branch, 2 St-data
- 256 entry ROB, up from 224



Zen 3 uArch

AMD

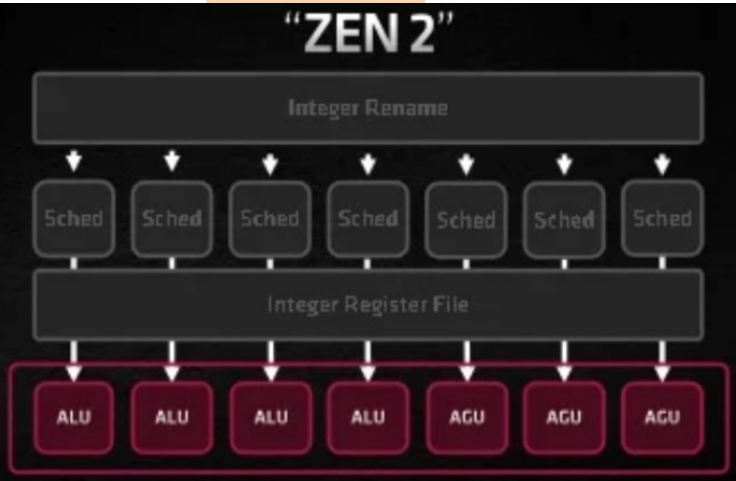
April 2022

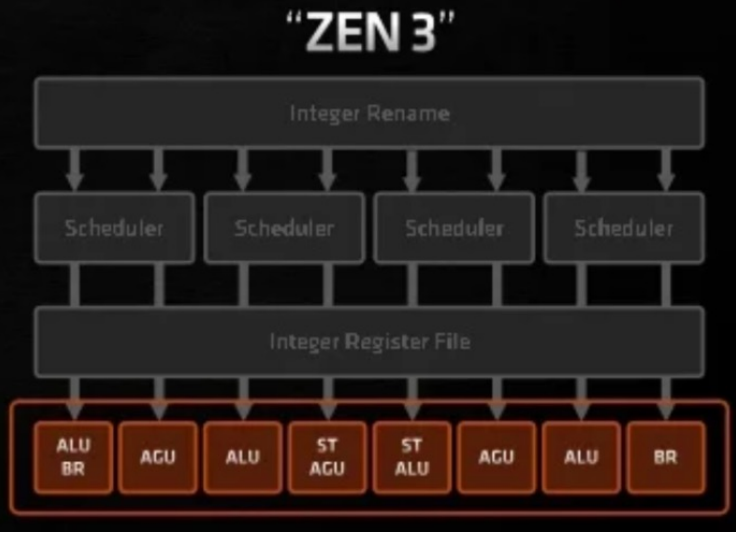
WIDER INTEGER EXECUTION

"ZEN 3" DESIGN GOAL: DELIVER WIDER EXECUTION RESOURCES IN A POWER- AND AREA-EFFICIENT MANNER

PICK BANDWIDTH IS INCREASED

- Still same number of "ALU" execution units
- Shared ALU/AGU schedulers allow for balanced use across workloads
- No increase in register file write ports or bypass network inputs





Zen 3 uArch

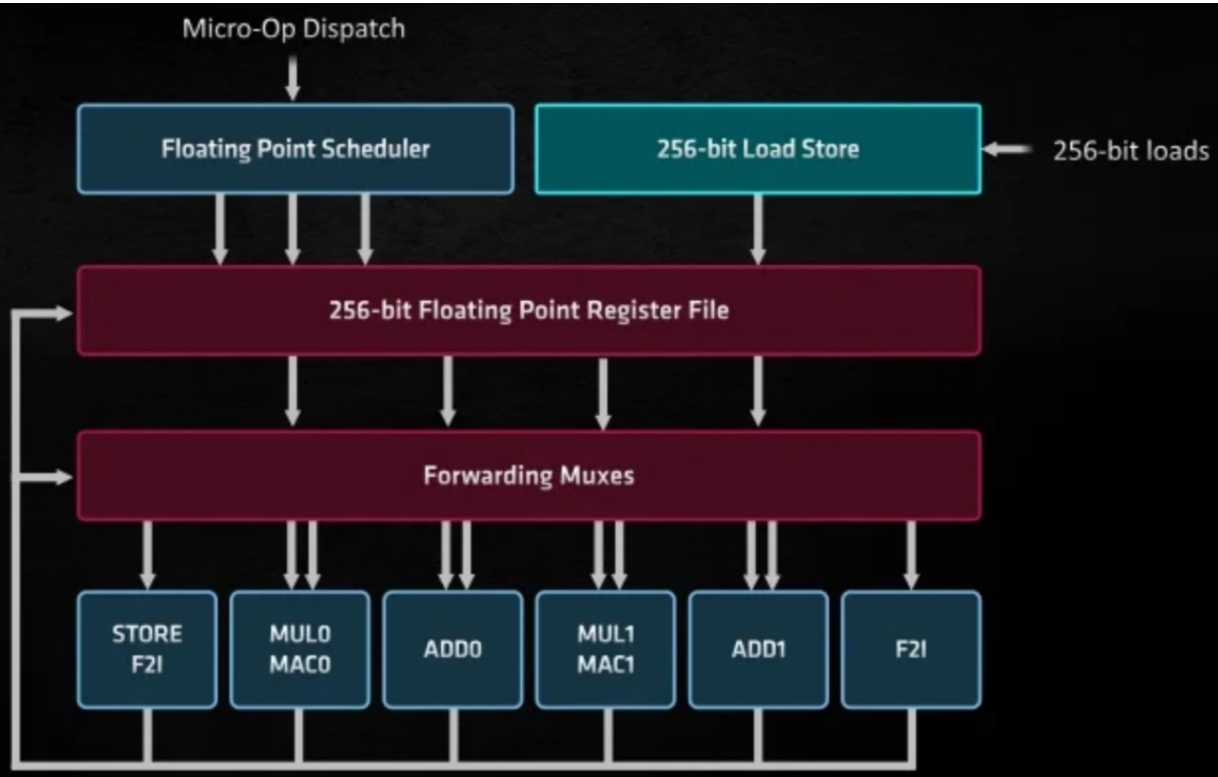
AMD

April 2022

FP EXECUTION

"ZEN 3" DESIGN GOAL: LOWER LATENCIES AND LARGER STRUCTURES TO EXTRACT ILP FOR FEEDING THE EXECUTION ENGINES

- Faster 4-cycle FMAC
- Increased Dispatch Bandwidth
- Separate F2I/Store Units
- Larger Scheduler



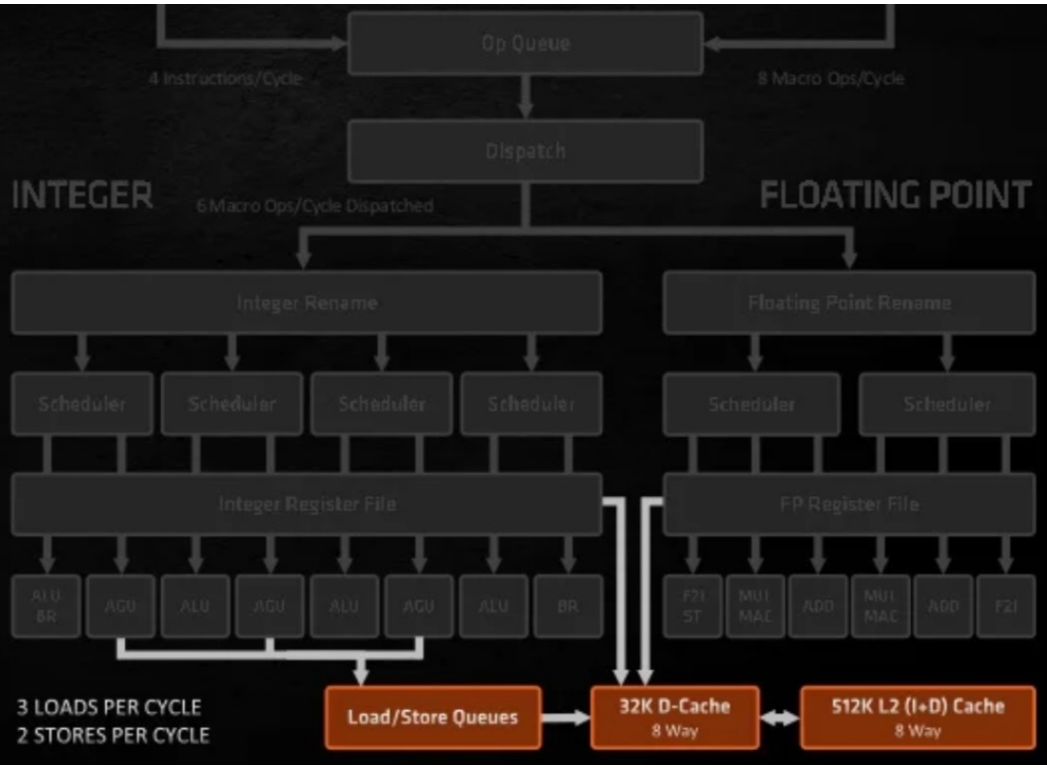
Zen 3 uArch

AMD

April 2022

LOAD/STORE ADVANCES

- Higher Bandwidth
- Greater Flexibility
- Improved Memory Dependence Detection
- +4 TLB Walkers



Zen 3 uArch

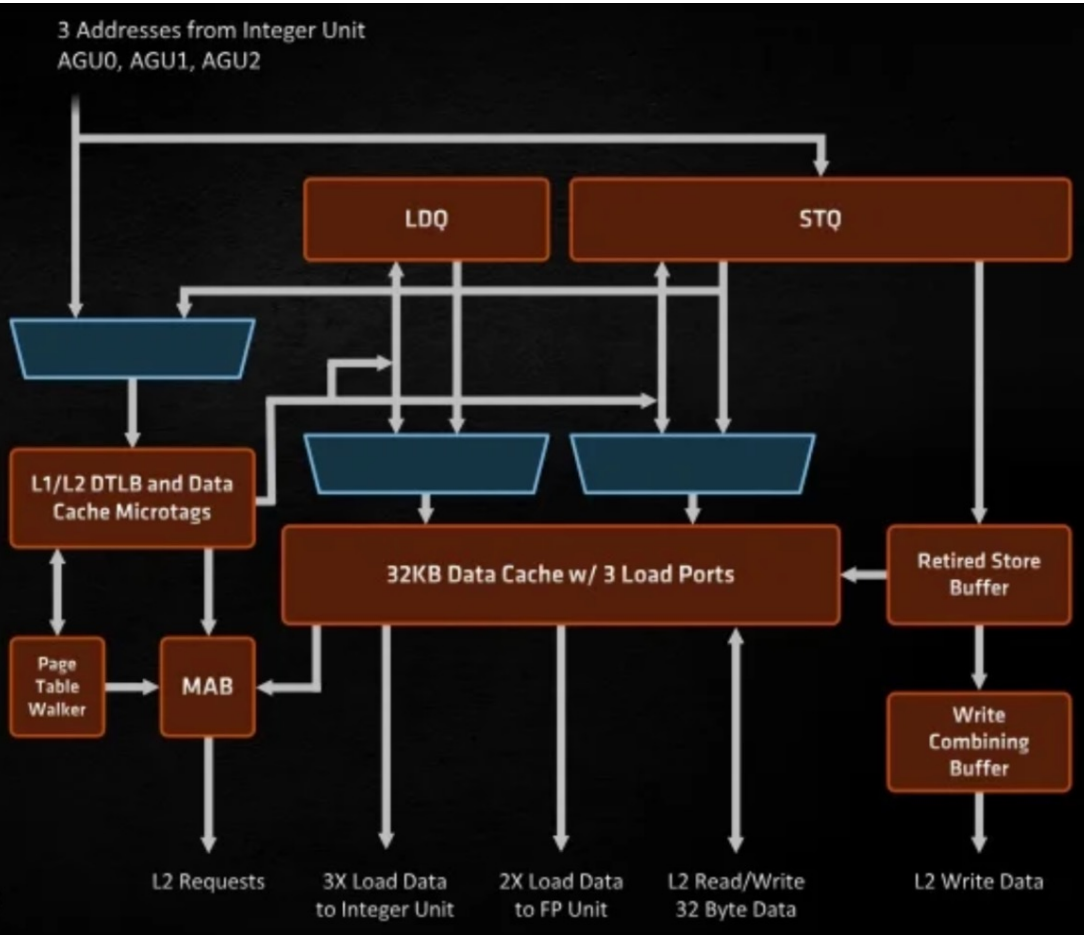
AMD

April 2022

LOAD/STORE

"ZEN 3" DESIGN GOAL: LARGER STRUCTURES AND BETTER PREFETCHING TO EXTRACT ILP FOR FEEDING WIDER EXECUTION

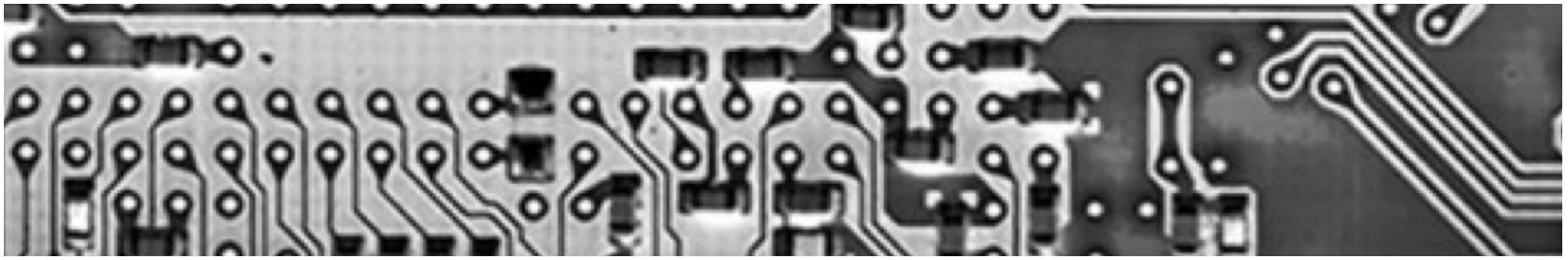
- 64 entry store queue, up from 48
- 2K entry L2 DTLB
- 32KB, 8-way L1 data cache
  - 3x memory ops per cycle
  - Max 3 loads per cycle (max 2 if 256b)
  - Max 2 stores per cycle (max 1 if 256b)
- Faster copy of short strings
- Improved prefetching across page boundaries
- Better prediction of store-to-load forward dependencies



CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP222

AMD uArch

https://www.amd.com/system/files/documents/security-whitepaper.pdf



INTRODUCTION

This document provides in depth descriptions of AMD CPU micro-architecture and how it handles speculative execution in a variety of architectural scenarios. This document is referring to the latest Family 17h processors which include AMD's Ryzen™ and EPYC™ processors, unless otherwise specified. This document does necessarily describe general micro-architectural principles that exist in all AMD microprocessors.

AMD's processor architecture includes hardware protection checks that AMD believes help AMD processors not be affected by many side-channel vulnerabilities. These checks happen in various speculation scenarios including during TLB validation, architectural exception handling, loads and floating point operations.

White Paper | SPECULATION BEHAVIOR IN AMD MICRO-ARCHITECTURES

5.14.19

AMD uArch

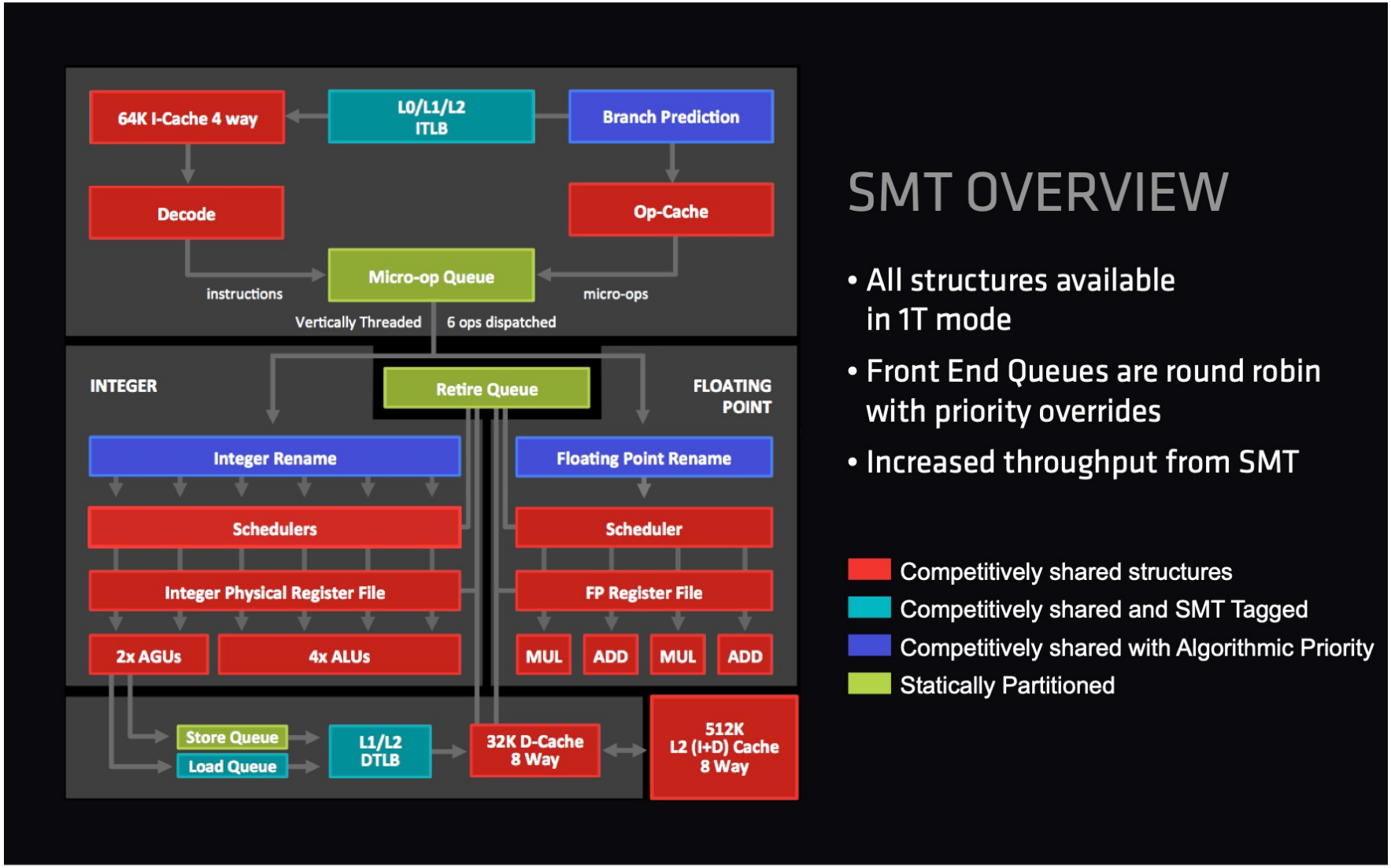
TLB ARCHITECTURE

The x86 architecture uses virtual addressing and hierarchical page tables to map the virtual address to the physical memory address used to reference caches and memory. This mapping allows privileged system software, whether the operating system or a hypervisor, to isolate different software environments by only allowing certain areas of the memory system to be accessed by each respective environment. This isolation is achieved by creating unique page tables for each environment. These page tables are isolated by either marking the page tables as not-present in the page table entry or using the protection attribute fields in the page table entry to restrict access.

For performance reasons, processors store a copy of these virtual to physical translations in a Translation Lookaside Buffer (TLB). AMD processors store translations in the TLB with a valid bit and all the protection bits from the page table which include user/supervisor, read/write bits along with other information. On each instruction that uses virtual addresses to access memory, AMD processors access the TLB and use the valid bit and the protection attributes to decide whether to access the caches. If the protection check fails, AMD processors operate as if the memory address is invalid and no data is accessed from either the cache or memory. This occurs whether the access is speculative or non-speculative. When the instruction becomes the oldest in the machine, a page fault exception will occur. A validated address is required for AMD processors to access data from both the caches and memory. **The result is AMD processors are designed to not speculate into memory that is not valid in the current virtual address memory range defined by the software defined page tables.**

Page Table → Valid, Protection bits

AMD uArch



The above diagram shows how all major blocks are shared within the SMT architecture.

AMD uArch

VECTOR	FAULT/TRAP TYPE	DESIGNED TO STOP SPECULATIVE EXECUTION	SPECULATIVELY FORWARDS DATA TO YOUNGER DEPENDENT INSTRUCTIONS
0	Divide by Zero(#DE)	NO	NO
1	Debug Trap(#DB)	NO	YES
1	Debug Instruction(#DB)	YES	N/A
3	INT3 Breakpoint(#BP)	YES	N/A
4	Overflow (#OF)	NO	YES
5	Bound(#BR)	NO	YES
6	Invalid Opcode(#UD)	YES	N/A
7	Device Not Available(#NM)	YES	N/A
8	Double Fault(#DF)	NO	YES
10	Invalid TSS(#TS)	NO	YES
11	Segment not Present(#NP)	NO	YES
12	Stack Fault(#SS)	NO	YES
13	General Protection Data Access(#GP)	NO	YES
13	General Protection Instruction Mode(#GP)	NO	NO
14	Page Fault(#PF)	NO	NO
16	X87 Floating-Point Pending(#MF)	YES	N/A
17	Alignment Check(#AC)	NO	YES

AMD uArch

FLOATING POINT SPECULATION

To improve performance of some floating point routines, AMD processors may predict the value of the x87 floating point control word (FCW) fields precision control (PC) and rounding control (RC) when a FLDCW instruction is executed. A similar prediction is made for the SSE and AVX register MXCSR with the rounding control (RC), flush to zero (FTX) and denormals as zero (DAZ) mode bits on a LDMXCSR instruction. In both cases, younger instructions may speculatively calculate results with the wrong rounding or precision control. When the real value of the FCW or MXCSR is known and does not match the predicted value, the processor will cause an internal exception to flush the younger instructions and re-issue them with the correct mode. Software that is sensitive to this type of speculation can place an LFENCE after the FLDCW or LDMXCSR instruction to restrict speculation.

CONCLUSION

In conclusion, AMD microprocessors have many micro-architectural mechanisms that allow for speculative execution. For software that cannot use the natural isolation that the TLB provides to prevent speculative execution into memory, AMD provides other software techniques to prevent speculative execution. These are described in the Software techniques for Managing Speculation on AMD processors.[3]

AMD believes that our hardware paging architecture and protection checks help AMD processors not be affected by many side-channel vulnerabilities, regardless of whether Simultaneous Multi-Threading (SMT) is enabled or disabled.

Cache Set Assoc (Ways)

AMD GPU

>14nm 2016-20

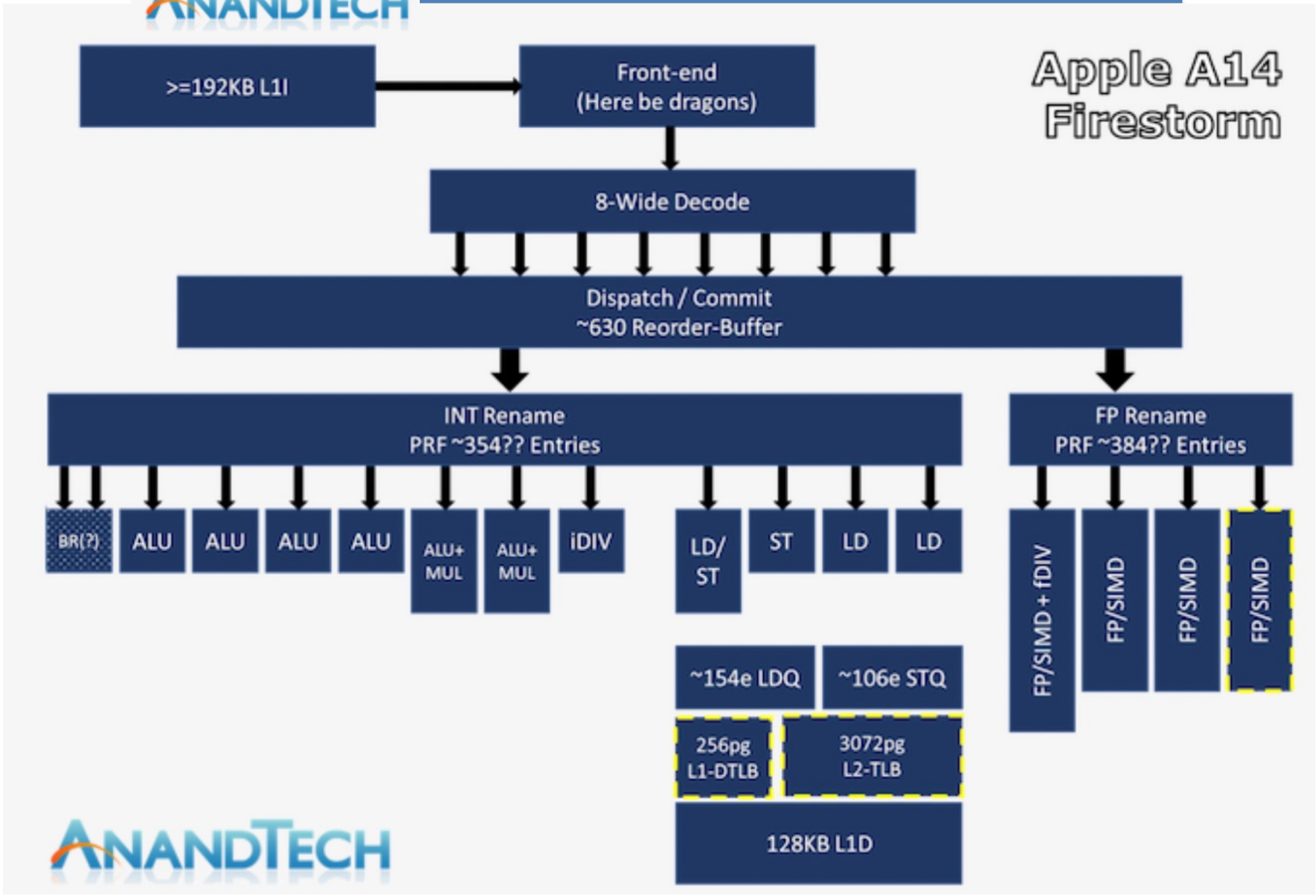
L1 data cache per core (KiB)	64	16			
L1 data cache associativity (ways)	2	4		8	
L1 instruction caches per core	1	0.5		1	
Max APU total L1 instruction cache (KiB)	256	128	192	256	512
L1 instruction cache associativity (ways)	2		3	4	8
L2 caches per core	1	0.5		1	
Max APU total L2 cache (MiB)	4		2		4
L2 cache associativity (ways)	16			8	
APU total L3 cache (MiB)	N/A			4	8
APU L3 cache associativity (ways)	N/A			16	
L3 cache scheme	Victim		N/A	Victim	

Section

Apple

Apple A14 uArch

ANANDTECH



ANANDTECH

Apple A14 uArch

ANANDTECH

The above diagram is an estimated feature layout of Apple's latest big core design – what's represented here is my best effort attempt in identifying the new designs' capabilities, but certainly is not an exhaustive drill-down into everything that Apple's design has to offer – so naturally some inaccuracies might be present.

What really defines Apple's Firestorm CPU core from other designs in the industry is just the sheer width of the microarchitecture. Featuring an 8-wide decode block, Apple's Firestorm is by far the current widest commercialized design in the industry. IBM's upcoming P10 Core in the POWER10 is the only other official design that's expected to come to market with such a wide decoder design, following Samsung's cancellation of their own M6 core which also was described as being design with such a wide design.

Other contemporary designs such as AMD's Zen(1 through 3) and Intel's µarch's, x86 CPUs today still only feature a 4-wide decoder designs (Intel is 1+3) that is seemingly limited from going wider at this point in time due to the ISA's inherent variable instruction length nature, making designing decoders that are able to deal with aspect of the architecture more difficult compared to the ARM ISA's fixed-length instructions. On the ARM side of things, Samsung's designs had been 6-wide from the M3 onwards, whilst Arm's own Cortex cores had been steadily going wider with each generation, currently 4-wide in currently available silicon, and expected to see an increase to a 5-wide design in upcoming Cortex-X1 cores.

Apple's microarchitecture being 8-wide actually isn't new to the new A14. I had gone back to the A13 and it seems I had made a mistake in the tests as I had originally deemed it a 7-wide machine. Re-testing it recently, I confirmed that it was in that generation that Apple had upgraded from a 7-wide decode which had been present in the A11 and 12.

Apple A14 uArch

Many, Many Execution Units

ANANDTECH

Having high ILP also means that these instructions need to be executed in parallel by the machine, and here we also see Apple's back-end execution engines feature extremely wide capabilities. On the Integer side, whose in-flight instructions and renaming physical register file capacity we estimate at around 354 entries, we find at least 7 execution ports for actual arithmetic operations. These include 4 simple ALUs capable of ADD instructions, 2 complex units which feature also MUL (multiply) capabilities, and what appears to be a dedicated integer division unit. The core is able to handle 2 branches per cycle, which I think is enabled by also one or two dedicated branch forwarding ports, but I wasn't able to 100% confirm the layout of the design here.

The Firestorm core here doesn't appear to have major changes on the Integer side of the design, as the only noteworthy change was an apparent slight increase (yes) in the integer division latency of that unit.

On the floating point and vector execution side of things, the new Firestorm cores are actually more impressive as they a 33% increase in capabilities, enabled by Apple's addition of a fourth execution pipeline. The FP rename registers here seem to land at 384 entries, which is again comparatively massive. The four 128-bit NEON pipelines thus on paper match the current throughput capabilities of desktop cores from AMD and Intel, albeit with smaller vectors. Floating-point operations throughput here is 1:1 with the pipeline count, meaning Firestorm can do 4 FADDs and 4 FMULs per cycle with respectively 3 and 4 cycles latency. That's quadruple the per-cycle throughput of Intel CPUs and previous AMD CPUs, and still double that of the recent Zen3, of course, still running at lower frequency. This might be one reason why Apples does so well in browser benchmarks (JavaScript numbers are floating-point doubles).

Vector abilities of the 4 pipelines seem to be identical, with the only instructions that see lower throughput being FP divisions, reciprocals and square-root operations that only have an throughput of 1, on one of the four pipes.

Apple A14 uArch

ANANDTECH

On the load-store front, we're seeing what appears to be four execution ports: One load/store, one dedicated store and two dedicated load units. The core can do at max 3 loads per cycle and two stores per cycle, but a maximum of only 2 loads and 2 stores concurrently.

What's interesting here is again the depth of which Apple can handle outstanding memory transactions. We're measuring up to around 148-154 outstanding loads and around 106 outstanding stores, which should be the equivalent figures of the load-queues and store-queues of the memory subsystem. To not surprise, this is also again deeper than any other microarchitecture on the market. Interesting comparisons are AMD's Zen3 at 44/64 loads & stores, and Intel's Sunny Cove at 128/72. The Intel design here isn't far off from Apple and actually the throughput of these latest microarchitecture is relatively matched – it would be interesting to see where Apple is going to go once they deploy the design to non-mobile memory subsystems and DRAM.

One large improvement on the part of the Firestorm cores this generation has been on the side of the TLBs. The L1 TLB has been doubled from 128 pages to 256 pages, and the L2 TLB goes up from 2048 pages to 3072 pages. On today's iPhones this is an absolutely overkill change as the page size is 16KB, which means that the L2 TLB covers 48MB which is well beyond the cache capacity of even the A14. With Apple moving the microarchitecture onto Mac systems, having compatibility with 4KB pages and making sure the design still offers enough performance would be a key part as to why Apple chose to make such a large upgrade this generation.

Apple A14 uArch

128KB L1 Caches

On the cache hierarchy side of things, we've known for a long time that Apple's designs are monstrous, and the A14 Firestorm cores continue this trend. Last year we had speculated that the A13 had 128KB L1 Instruction cache, similar to the 128KB L1 Data cache for which we can test for, however following Darwin kernel source dumps Apple has confirmed that it's actually a massive 192KB instruction cache. That's absolutely enormous and is 3x larger than the competing Arm designs, and 6x larger than current x86 designs, which yet again might explain why Apple does extremely well in very high instruction pressure workloads, such as the popular JavaScript benchmarks.

The huge caches also appear to be extremely fast – the L1D lands in at a 3-cycle load-use latency. We don't know if this is clever load-load cascading such as described on Samsung's cores, but in any case, it's very impressive for such a large structure. AMD has a 32KB 4-cycle cache, whilst Intel's latest Sunny Cove saw a regression to 5 cycles when they grew the size to 48KB. Food for thought on the advantages or disadvantages of slow of fast frequency designs.

Apple A14 uArch

16MB L2=LLC Cache

On the L2 side of things, Apple has been employing an 8MB structure that's shared between their two big cores. This is an extremely unusual cache hierarchy and contrasts to everybody else's use of an intermediary sized private L2 combined with a larger slower L3. Apple here disregards the norms, and chooses a large and fast L2. Oddly enough, this generation the A14 saw the L2 of the big cores make a regression in terms of access latency, going back from 14 cycles to 16 cycles, reverting the improvements that had been made with the A13. We don't know for sure why this happened, I do see higher parallel access bandwidth into the cache for scalar workloads, however peak bandwidth still seems to be the same as the previous generation. Another point of hypothesis is that because Apple shares the L2 amongst cores, that this might be an indicator of changes for Apple Silicon SoCs with more than just two cores connected to a single cache, much like the A12X generation.

Apple has had employed a large LLC on their SoCs for many generations now. On the A14 this appears to be again a 16MB cache that is serving all the IP blocks on the SoC, most useful of course for the CPU and GPU. Comparatively speaking, this cache hierarchy isn't nearly as fast as the actual CPU-cluster L3s of other designs out there, and in recent years we've seen more mobile SoC vendors employ such LLC in front

Apple A14 uArch

DR JEFF SOFTWARE
INDIE APP DEVELOPER

ANANDTECH

Maximum Frequency vs Loaded Threads Per-Core Maximum MHz						
Apple A14	1	2	3	4	5	6
Performance 1	2998	2890	2890	2890	2890	2890
Performance 2		2890	2890	2890	2890	2890
Efficiency 1			1823	1823	1823	1823
Efficiency 2				1823	1823	1823
Efficiency 3					1823	1823
Efficiency 4						1823

Of course, the old argument about having a very wide architecture is that you cannot clock as high as something which is narrower. This is somewhat true; however, I wouldn't come to any conclusion as to the capabilities of Apple's design in a higher power device. On the A14 inside of the new iPhones the new Firestorm cores are able to reach 3GHz clock speeds, clocking down to 2.89GHz when there's two cores active at any time.

We'll be investigating power in more detail in just a bit, but I currently see Apple being limited by the thermal envelope of the actual phones rather than it being some intrinsic clock ceiling of the microarchitecture. The new Firestorm cores are clocking in now at roughly the same speed any other mobile CPU microarchitecture from Arm even though it's a significantly wider design – so the argument about having to clock slower because of the more complex design also doesn't seem to apply in this instance. It will be very interesting to see what Apple could do not only in a higher thermal envelope device such as a laptop, but also on a wall-powered device such as a Mac.

Section

Fujitsu

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP222

Fujitsu Microarchitecture

Micro Architecture