Introduction to SIMD

(Theory Part)

Presented by the NOTLANDERS

Demo

Overview

- What is SIMD?
- How does SIMD work on a hardware level?
- How does SIMD make programs run faster?
- How to write SIMD?

Why should I learn about SIMD?

- Many areas of game programming are performance sensitive
- Tool in your toolbox
- Internship / job interview

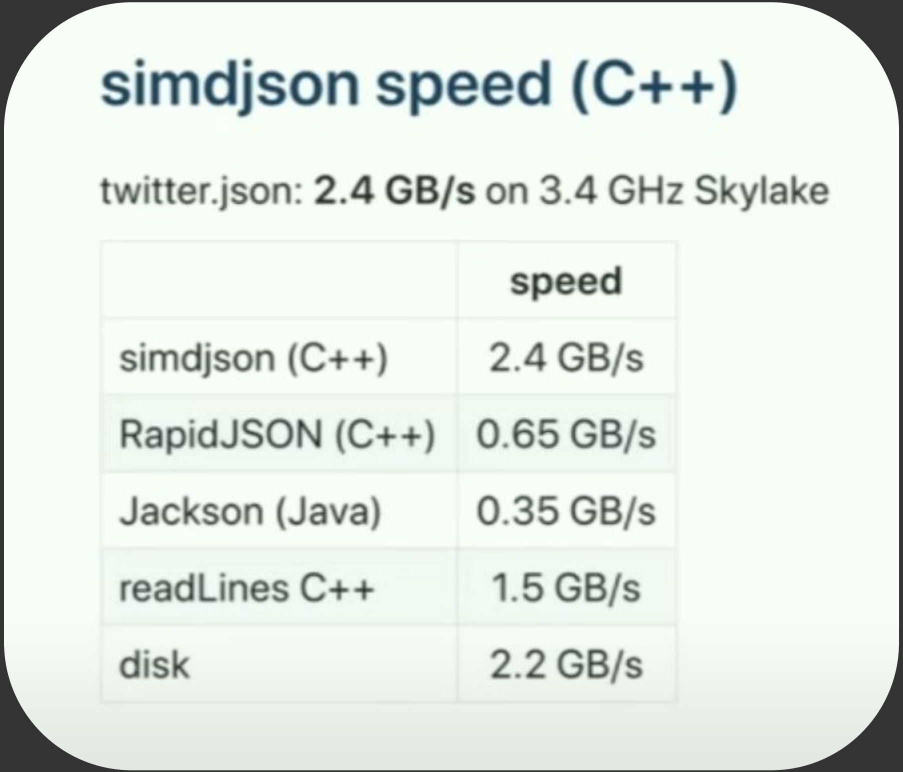
Single instruction, multiple data (SIMD)

A way to utilize the CPU hardware to process multiple values simultaneously for better performance.

Areas of usage

Performance sensitive applications that process data in batches in a way that can be parallelized.

- Multimedia
- 3D Graphics
- Math / Physics



simdjson speed (C++)

twitter.json: **2.4 GB/s** on 3.4 GHz Skylake

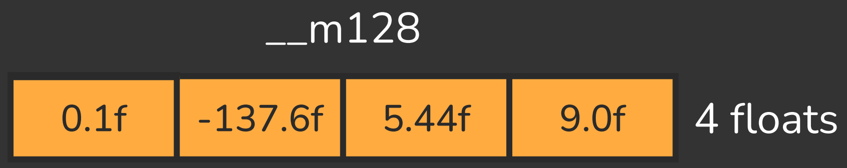
	speed
simdjson (C++)	2.4 GB/s
RapidJSON (C++)	0.65 GB/s
Jackson (Java)	0.35 GB/s
readLines C++	1.5 GB/s
disk	2.2 GB/s

Vector/vectorization in the context of SIMD

Think of a SIMD vector as a fixed-size array.

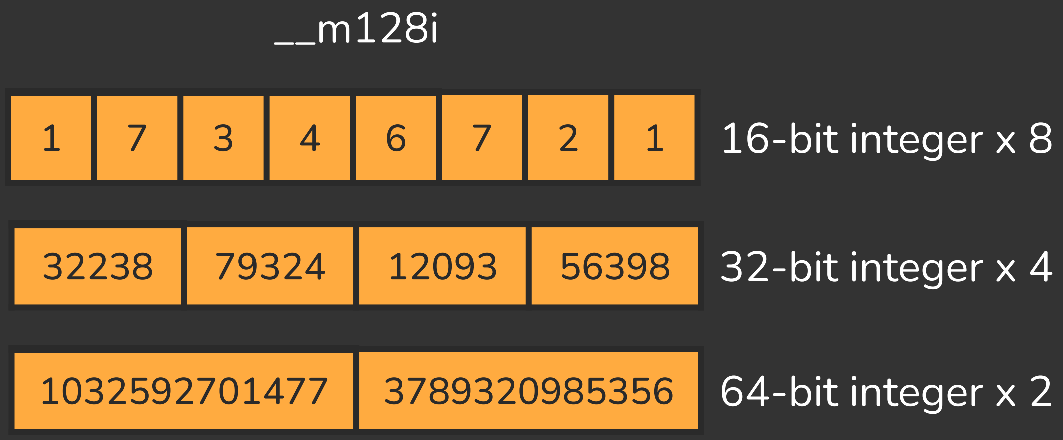
No relation to:

- std::vector
- Mathematical vector
- Vector2, Vector3

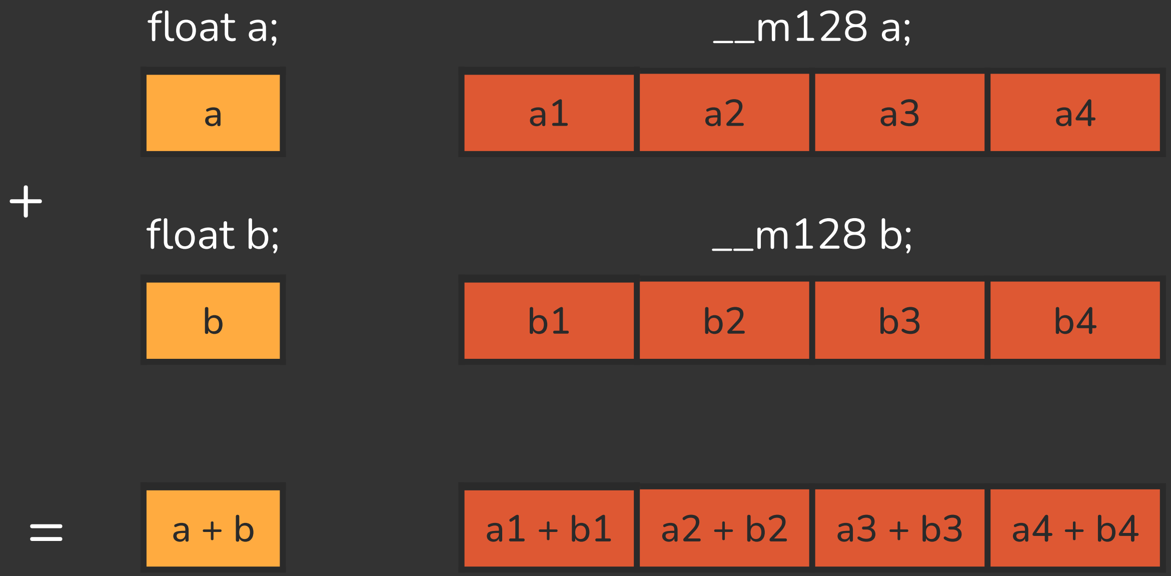


Lanes

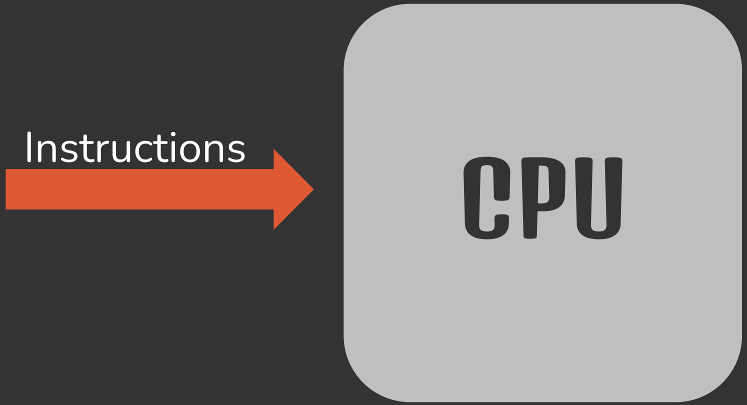
The number of elements in these vectors is not fixed. The proper term to use is lanes.

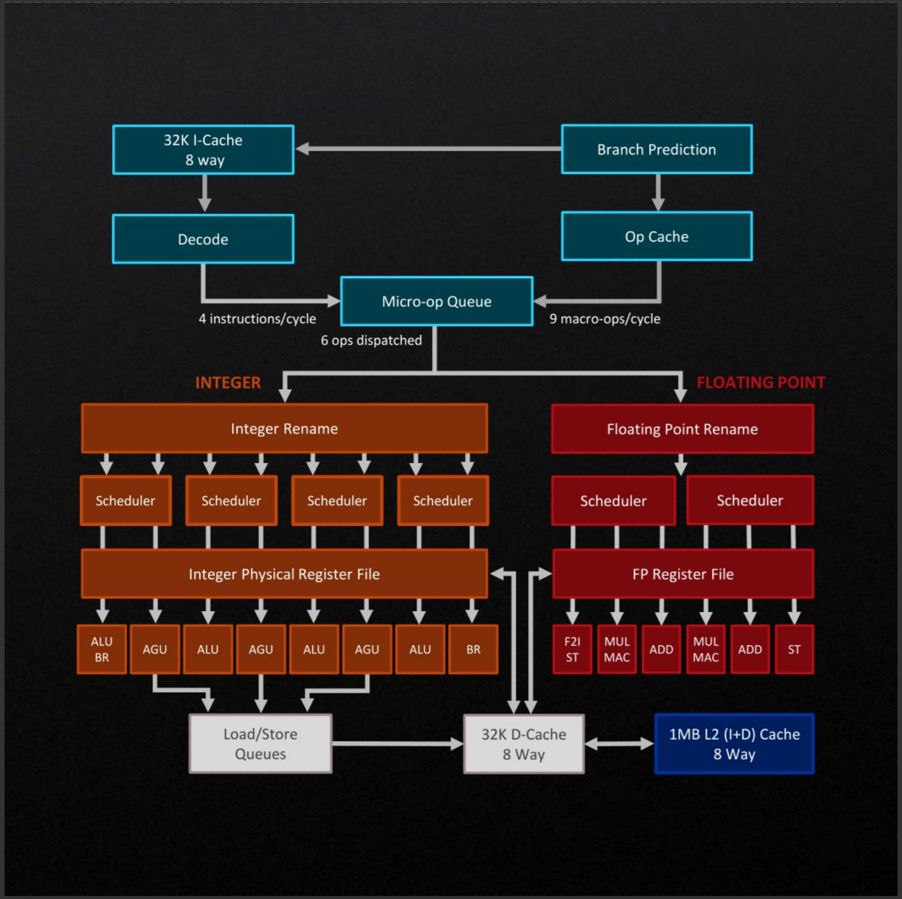


Scalar vs. Vector

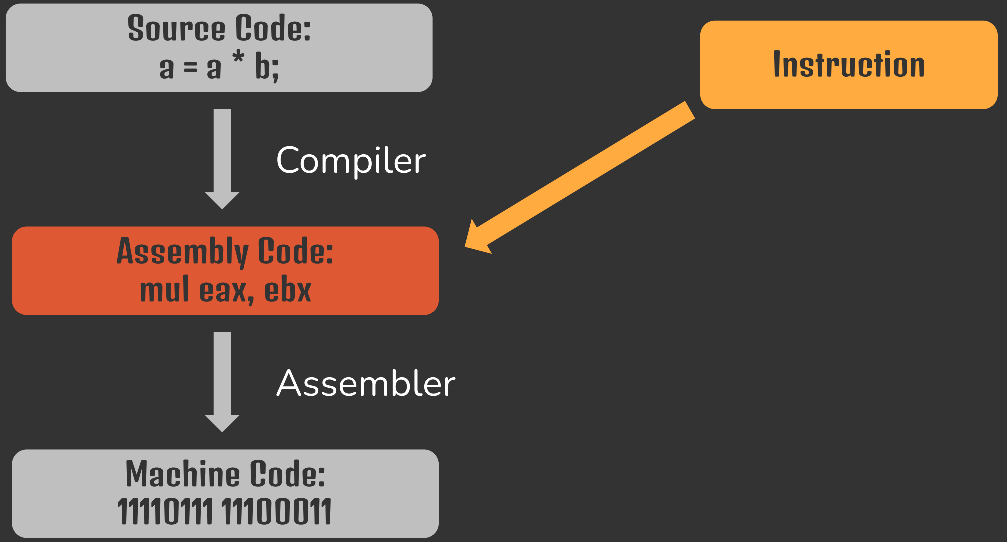


CPU Architecture Overview



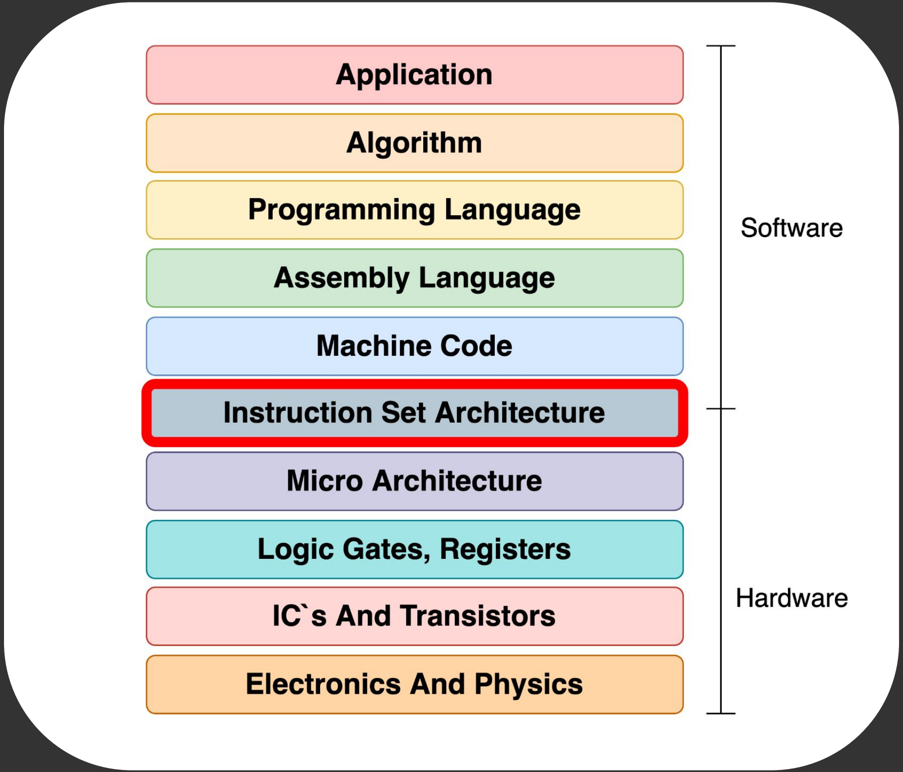


What is an instruction?



Instruction Set Architecture (ISA)

- x86 - Desktop, PS5, Xbox
  - Intel
  - AMD
- ARM - Mobile, Switch
  - Apple Silicon
  - Qualcomm Snapdragon

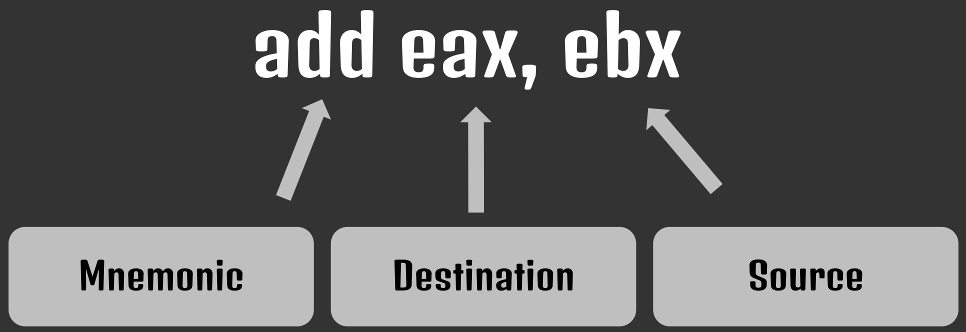


Instruction set extensions

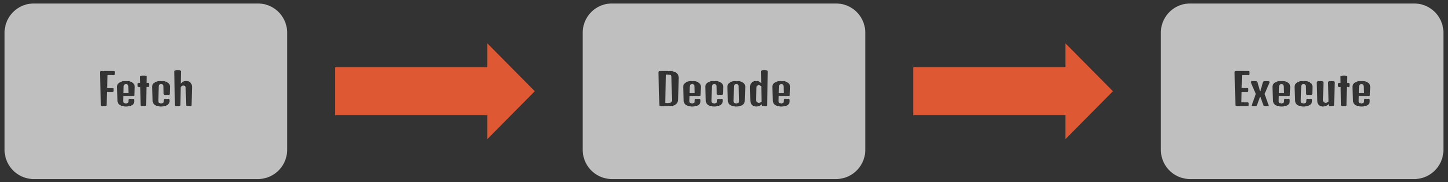
SIMD instructions aren’t generally part of the base instruction set. Instead, they come in the form of extensions.

- x86
  - SSE
  - AVX
- ARM
  - NEON

Anatomy of an x86 instruction



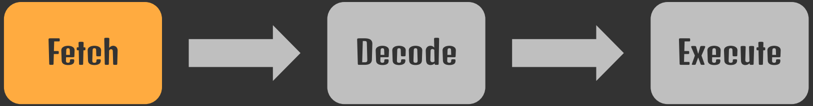
Instruction cycle

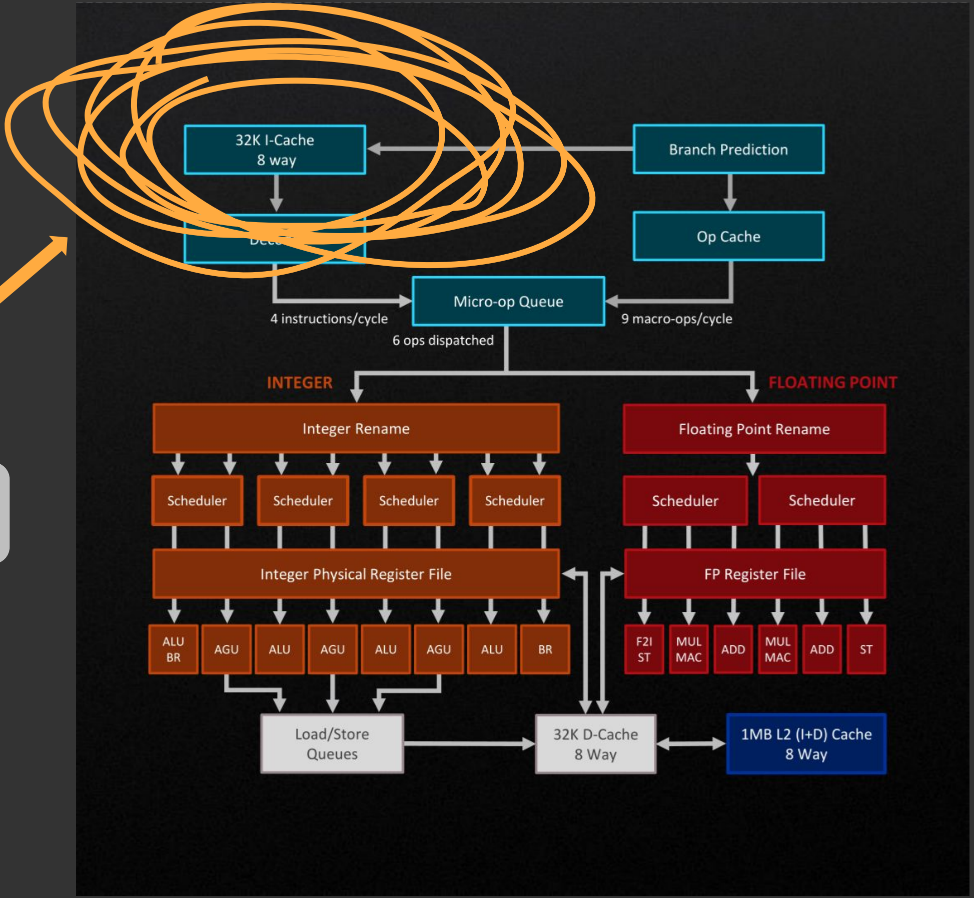


Fetch

Retrieve the next instruction(s) from memory.

Instruction Cache

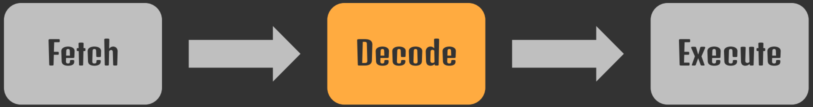


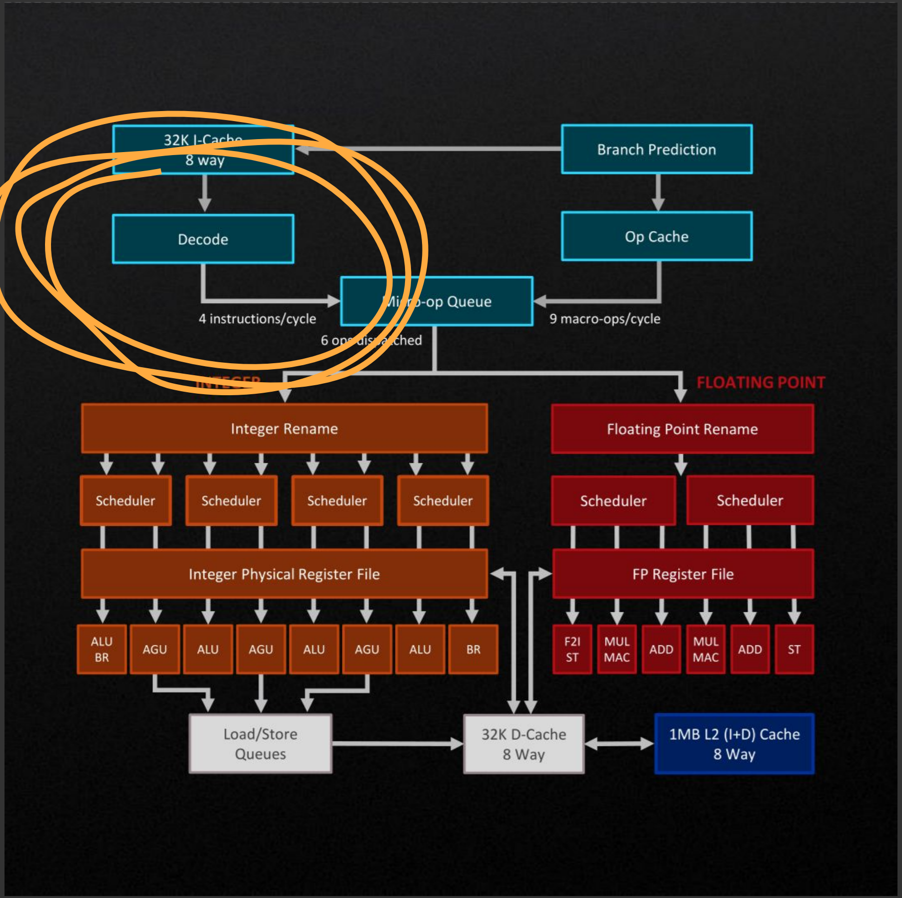


Decode

Interpret the instruction, breaking it down into specific operations.

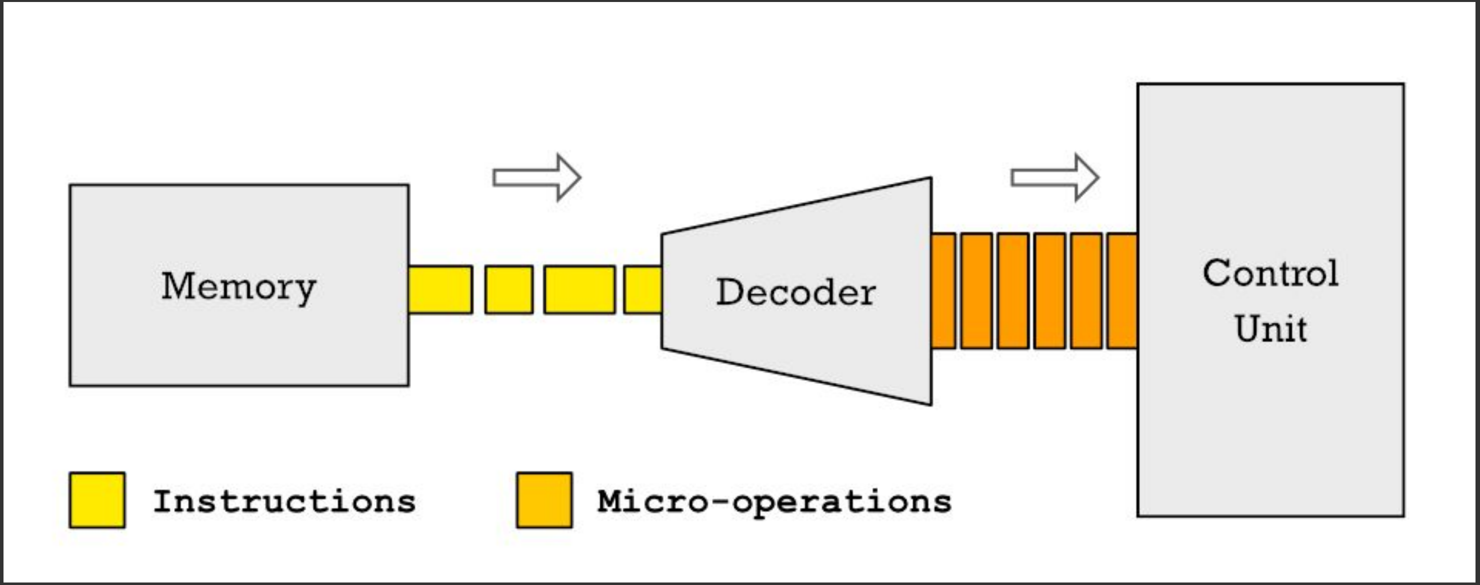
Decoders





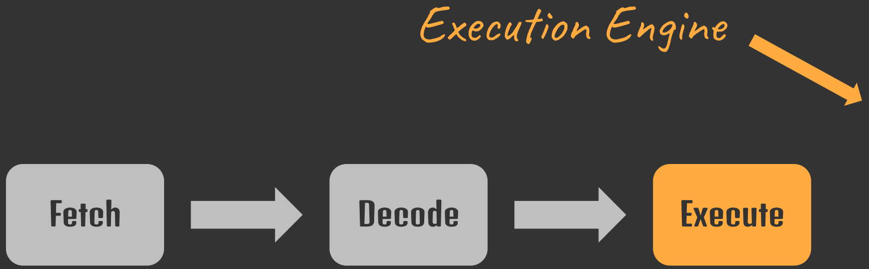
Micro-operations (micro-ops / μops)

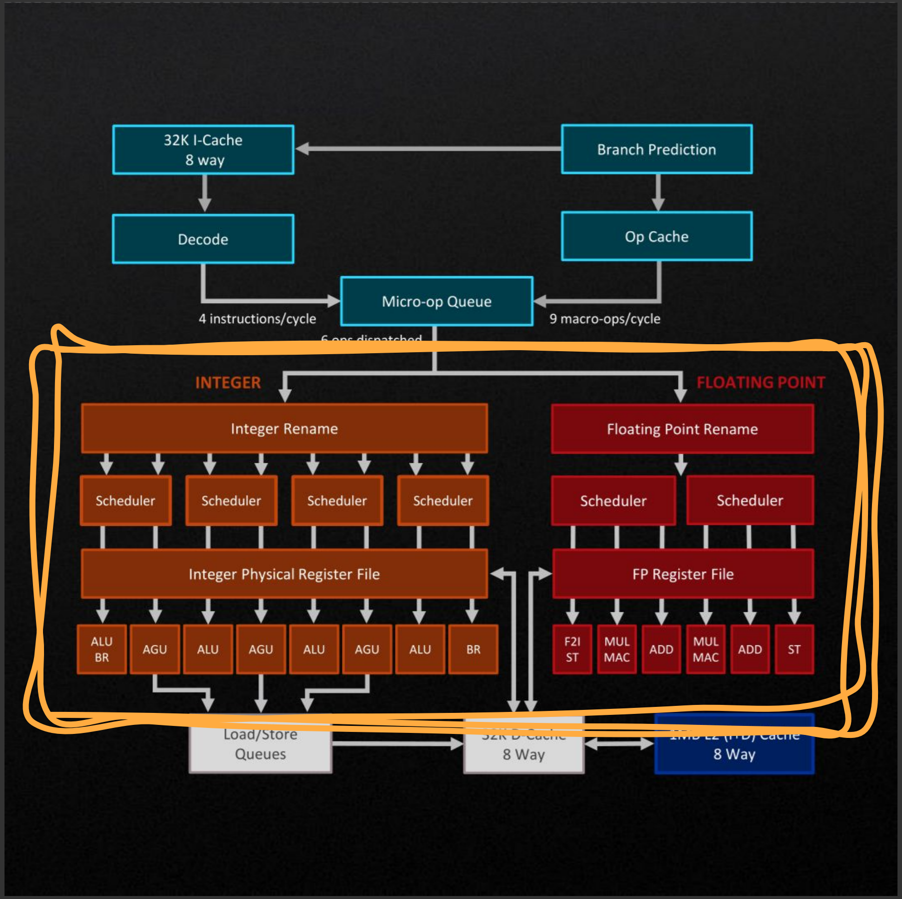
https://uops.info/html-instr/IDIV_R32.html



Execute

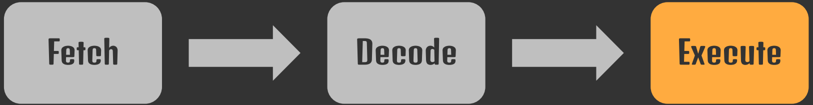
Carry out the operations specified by the instruction.

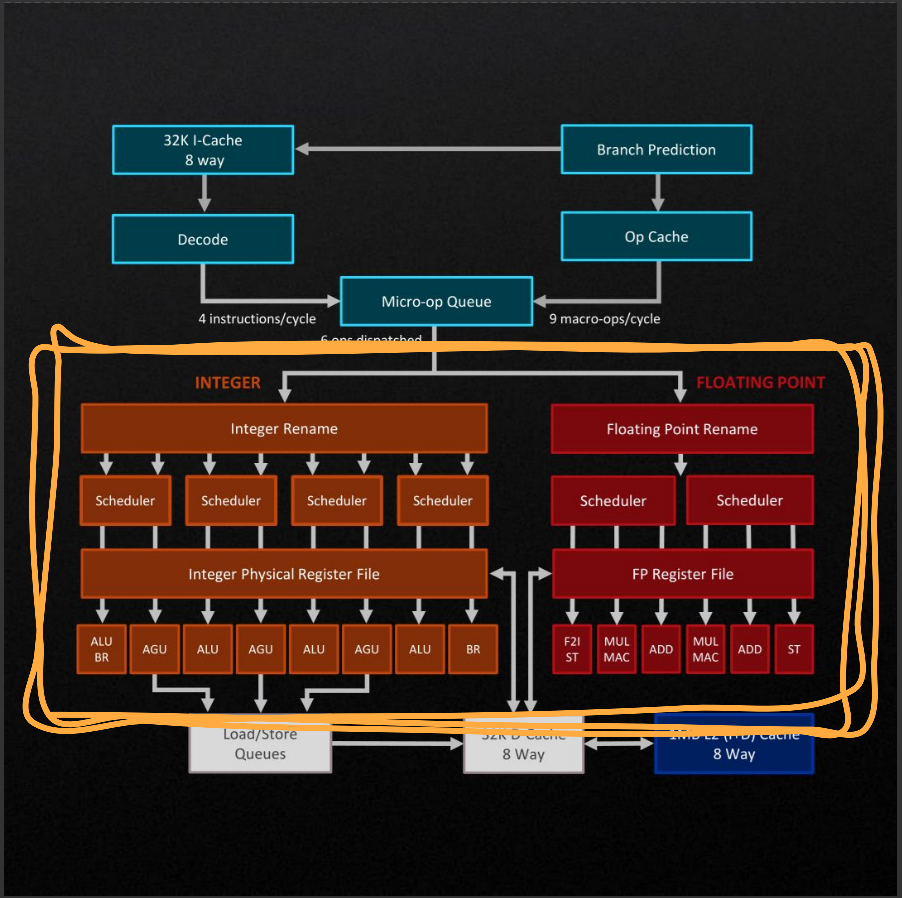




Execute

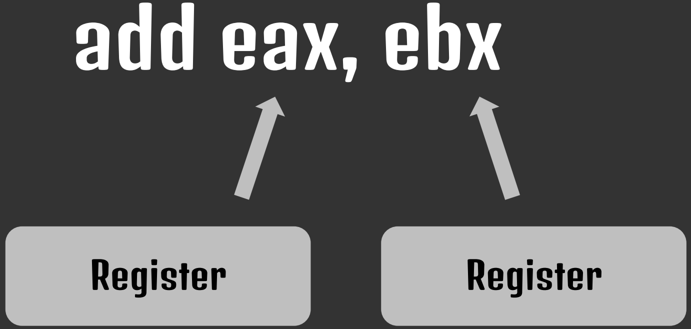
- Registers - circuits that temporarily store inputs and outputs
- Execution units - circuits that perform operations on data in registers





Registers

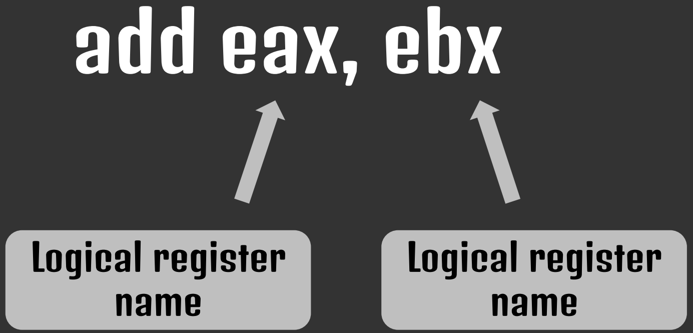
A register is a super fast storage location inside the CPU. It's like a workspace for the CPU that temporarily hold data for processing.



Register renaming

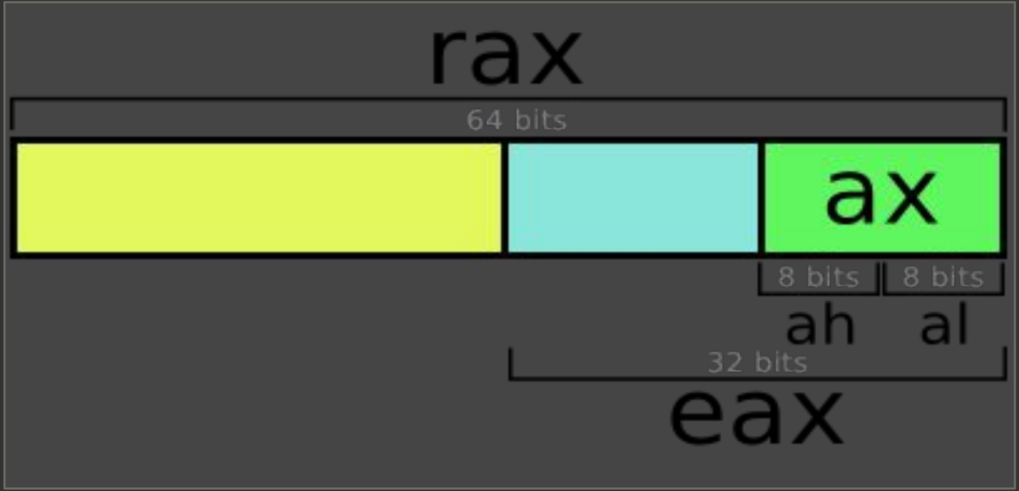
On Intel 8086 logical registers mapped directly to physical registers.

Modern CPUs have hundreds of physical registers and use register renaming.



Register names

Different logical register names can be used to access different parts of a register.

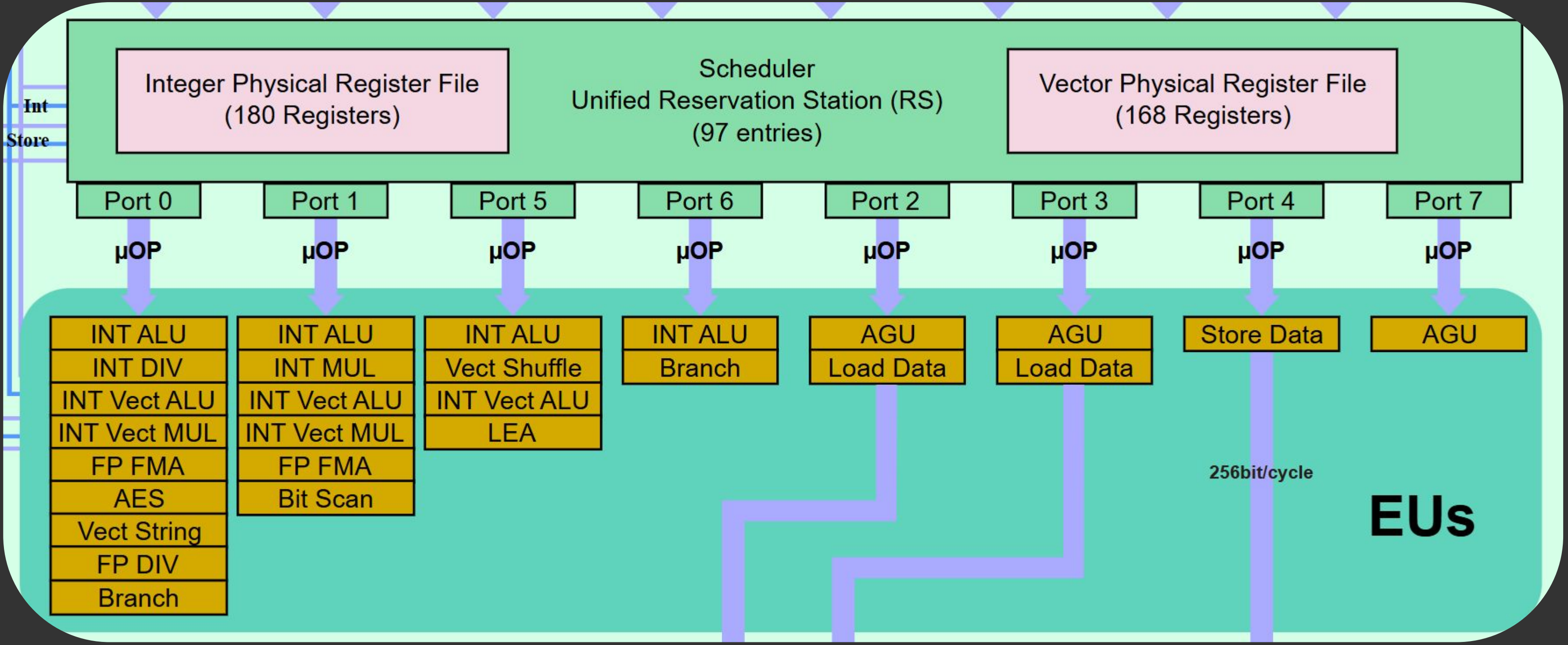


Execution units

Specialized components for executing different types of operations.

- Arithmetic logic unit (ALU)
- Address generation unit (AGU)
- Load-store unit (LSU)
- Branch execution unit (BEU)

Execution units



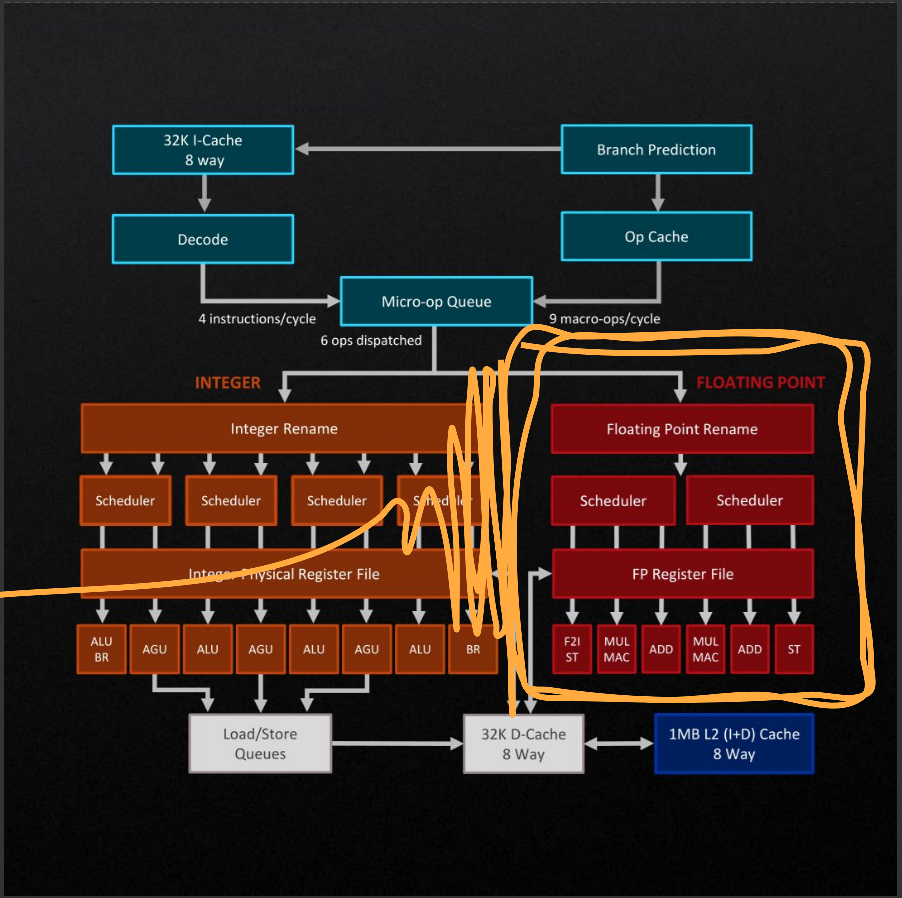
Where is the SIMD hardware?

Floating point path = SIMD path

Old Intel CPUs used to have a floating point coprocessor for which they designed an extension called x87.

Modern compilers will generate SSE instructions for scalar floating point operations*, though x87 is still supported.

SIMD hardware

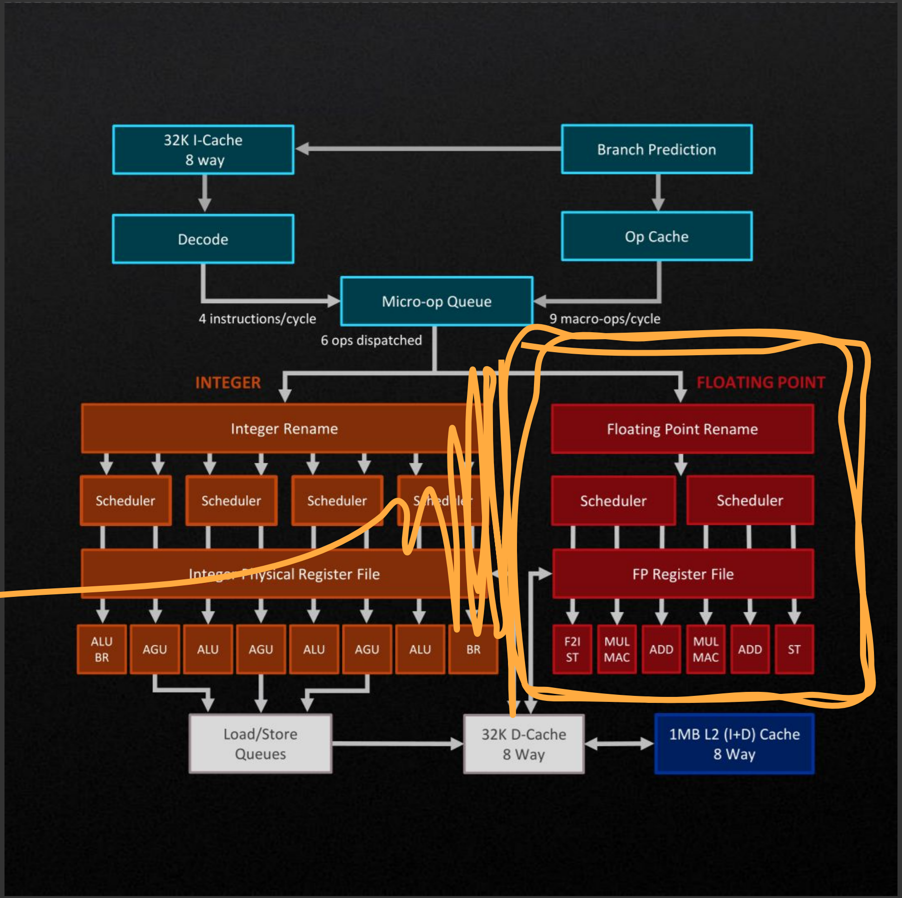


*I'm sure there are exceptions

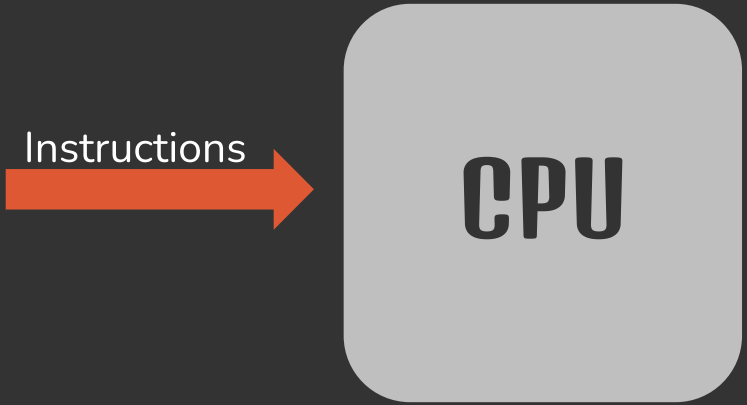
Wide hardware

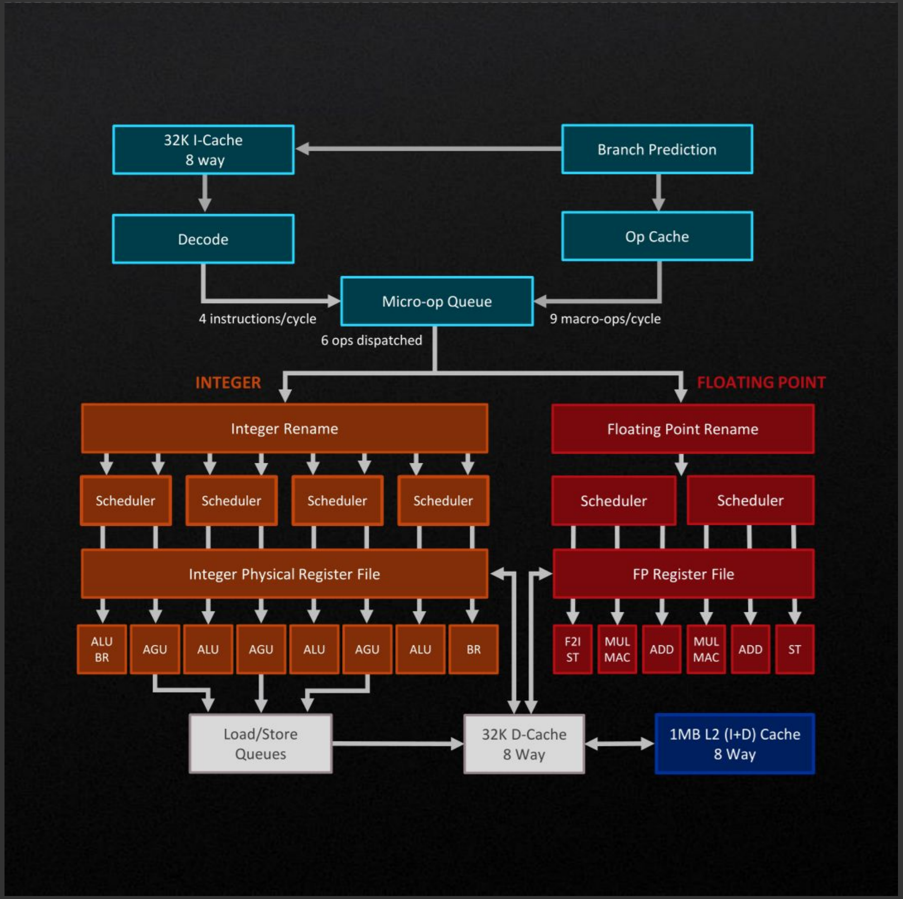
While components that handle scalar integer instructions are typically 64-bit wide, hardware that handles SIMD instructions is 256-bit (or even 512-bit) wide on modern CPUs.

SIMD hardware



CPU Architecture Overview



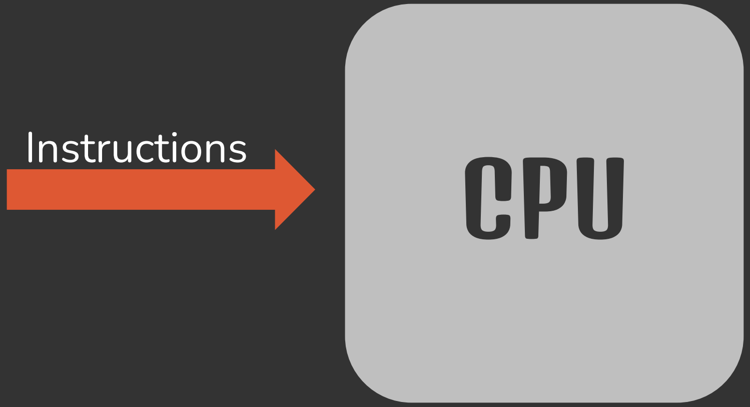


the NOTLANDERS

Performance

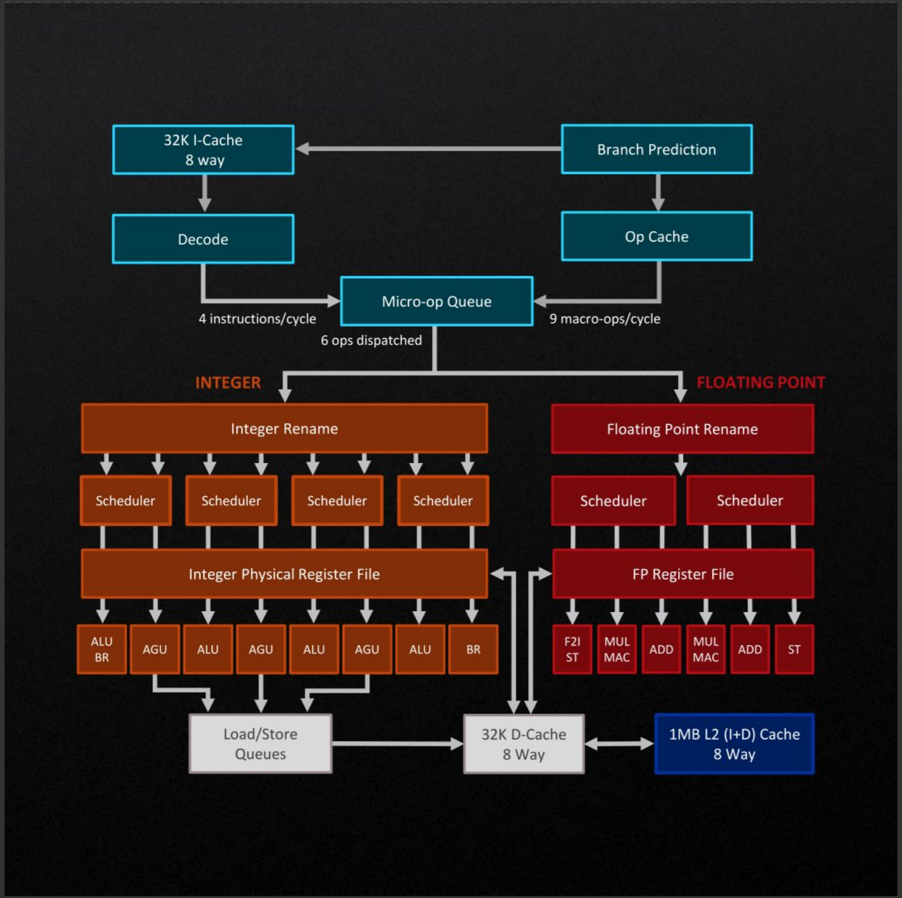
1. Number of instructions
   a. Algorithmic complexity
   b. Waste

2. Speed at which instructions go through the CPU
   a. Data locality
   b. Multi-threading
   c. SIMD



How does SIMD make my code faster?

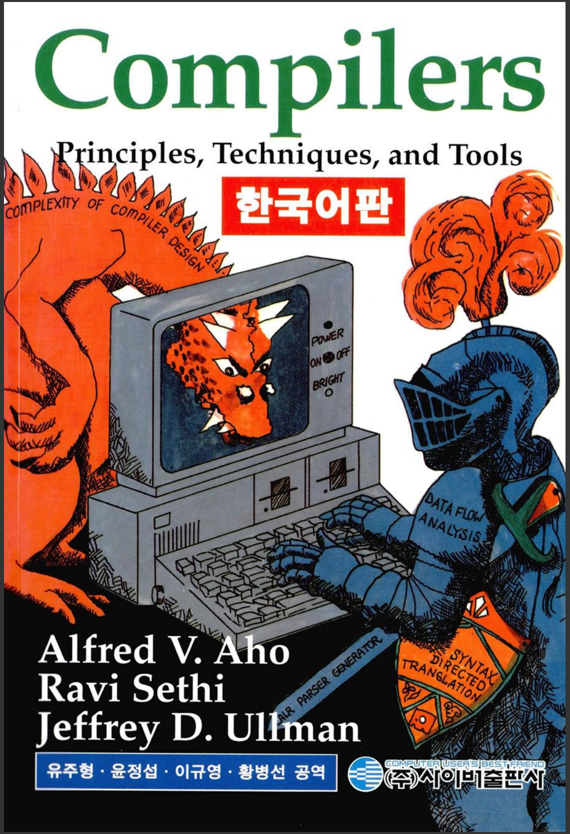
- Faster execution - do more useful work at the same time by utilizing vector hardware.
- Reduced cost of decoding - only one instruction to decode instead of multiple.
- Maximizes cache bandwidth - load more data from memory per cycle.



Auto-vectorization

If certain criteria are met, a compiler may be able to vectorize your code.

Free optimization? Yes, but compilers usually need some help to be able to auto-vectorize.


Compilers
Principles, Techniques, and Tools
한국어판
COMPLEXITY OF COMPILER DESIGN
POWER
ON OFF
BRIGHT
DATA FLOW ANALYSIS
LALR PARSER GENERATOR
SYNTAX DIRECTED TRANSLATION
Alfred V. Aho
Ravi Sethi
Jeffrey D. Ullman
유주형 · 윤정섭 · 이규영 · 황병선 공역
COMPUTER USER'S BEST FRIEND
(주)사이버출판사

Auto-vectorization

1. Loop vectorizer unrolls loops and writes them as SIMD
2. Block vectorizer (a.k.a. SLP) merges multiple scalars into a vector in a block of code

https://www.intel.com/content/dam/develop/external/us/en/documents/31848-compilerautovectorizationguide.pdf

Normal loop:

```
for (int i = 0; i < 1024; i++)
{
        a[i] = i;
}
```

Unrolled loop:

```
for (int i = 0; i < 256; i += 4)
{
        a[i] = i;
        a[i + 1] = i + 1;
        a[i + 2] = i + 2;
        a[i + 3] = i + 3;
}
```

Back to demo

Learn More

Articles:
http://const.me/articles/simd/simd.pdf
https://mcyoung.xyz/2023/11/27/simd-base64/

Talks:
https://gdcvault.com/play/1022248/SIMD-at-Insomniac-Games-How

Courses:
https://www.computerenhance.com/

Books:
https://www.bokus.com/bok/9780128203316/computer-organization-and-design-risc-v-edition/

Glossary

ARM

Auto-vectorization

AVX

CPU backend

CPU frontend

Execution engine

Execution unit

Fetch-Decode-Execute

Instruction set architecture

Microarchitecture

Register

Register renaming

SIMD

SSE

Vector

x86

Thanks for listening!

Please fill out the feedback form!

Presented by the NOTLANDERS