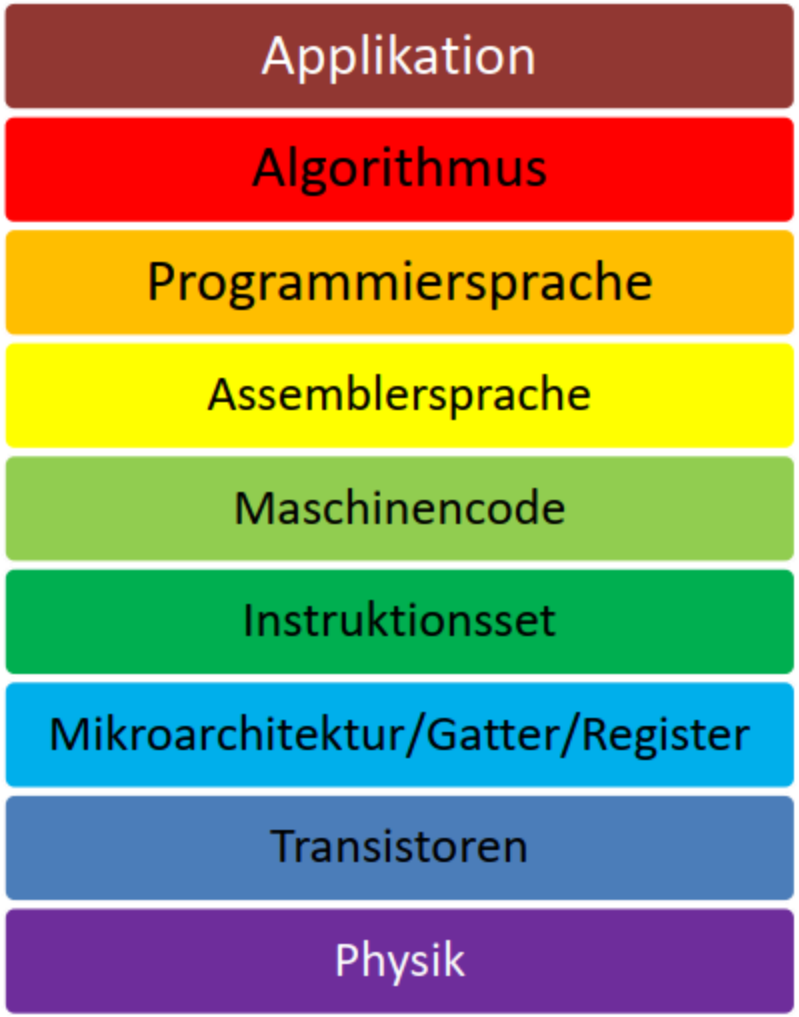
Microcomputertechnik

> "You don't have to be an engineer to be be a racing driver, but you do have to have Mechanical Sympathy."

-- vermutlich: Jackie Stewart, Rennfahrer

Überblick


Applikation
Algorithmus
Programmiersprache
Assemblersprache
Maschinencode
Instruktionsset
Mikroarchitektur/Gatter/Register
Transistoren
Physik

Computer Timescale vs Human Timescale

Task	Computer	Human
Add two Numbers	0.5ns	5s
Multiplication	2ns	20s
Division (32bit)	15ns	150s
Add (float)	2ns	20s
Multiply (float)	2ns	20s
Access L1 Cache	2ns	20s
Access L2 Cache	6ns	60s
Access L3 Cache	20ns	3min

Task	Computer	Human
Access RAM	100ns	15min
Read SSD	20us	2 days
Read HDD	10ms	3.1 years
Time (1 frame @ 60fps)	16ms	5.1 years
Local LAN Ping	400us	6 weeks
Next Google DNS	10ms	3.1 years
Next Google DNS	100ms	31 years

$ T_H = T_C*10^9 $

Quelle: https://www.youtube.com/watch?v=PpaQrzoDW2I

Alan Touring

- *1912 - †1954
- Begründer der «Computer Science»
- WWII: Massgeblich am Knacken der deutschen Enigma-Verschlüsselung beteiligt
- Verfolgt und «therapiert» wegen Homosexualität
- Vermutlich Selbstmord
- The Imitation Game, Benedict Cumberbatch

Turingmaschine

Die Turingmaschine hat ein Steuerwerk, in dem sich das Programm befindet, und besteht außerdem aus

- einem unendlich langen Speicherband mit unendlich vielen sequentiell angeordneten Feldern.
- einem programmgesteuerten Lese- und Schreibkopf, der sich auf dem Speicherband feldweise bewegen und die Zeichen
  verändern kann.
- Turing bewies, dass solch ein Gerät in der Lage ist, „jedes vorstellbare mathematische Problem zu lösen, sofern dieses
  auch durch einen Algorithmus gelöst werden kann“.

Turingvollständigkeit

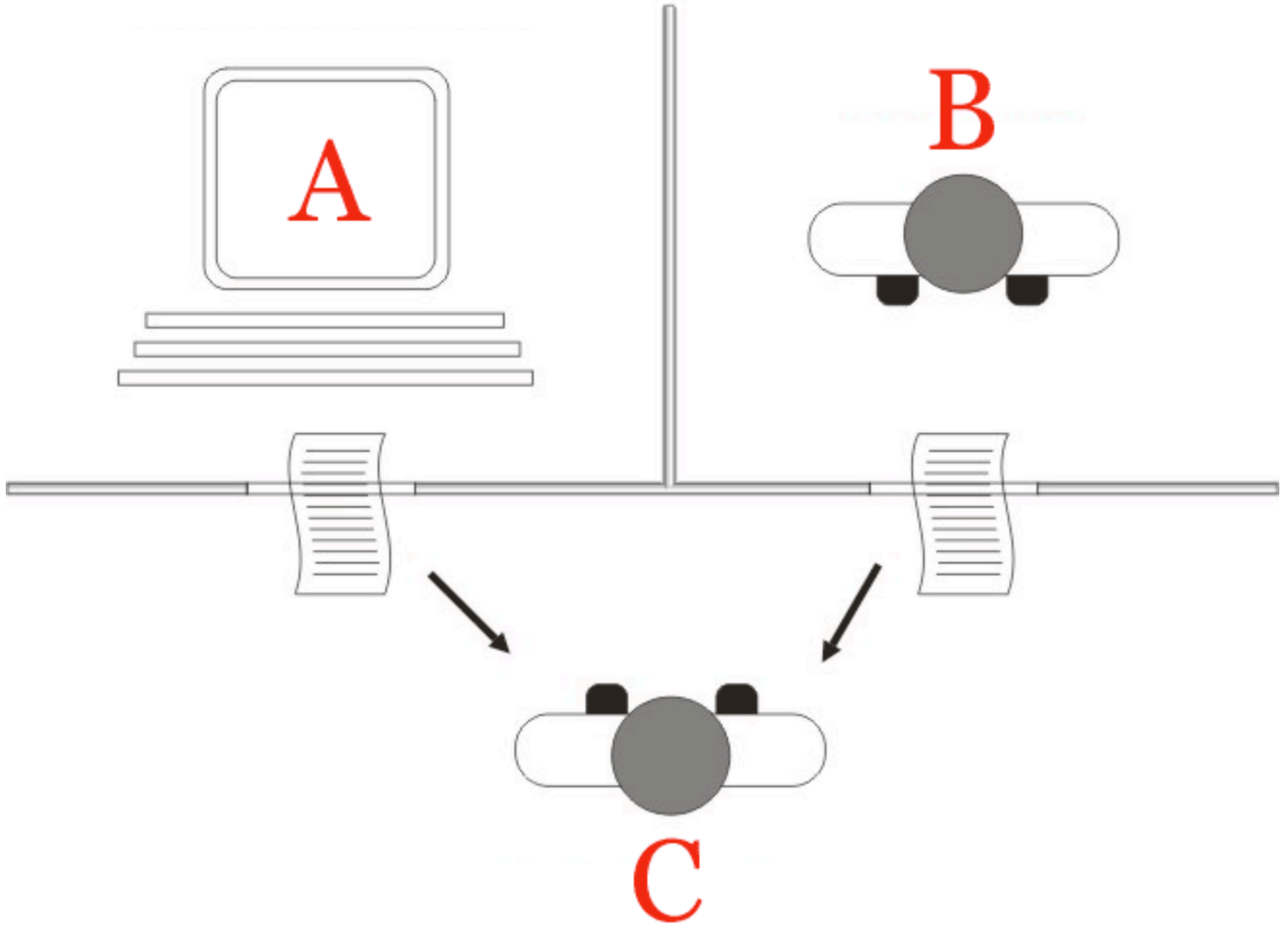
«Exakt ausgedrückt bezeichnet Turing-Vollständigkeit in der Berechenbarkeitstheorie die Eigenschaft einer
Programmiersprache oder eines anderen logischen Systems, sämtliche Funktionen berechnen zu können, die eine universelle
Turingmaschine berechnen kann.»
(https://de.wikipedia.org/wiki/Turing-Vollständigkeit)

Halteproblem

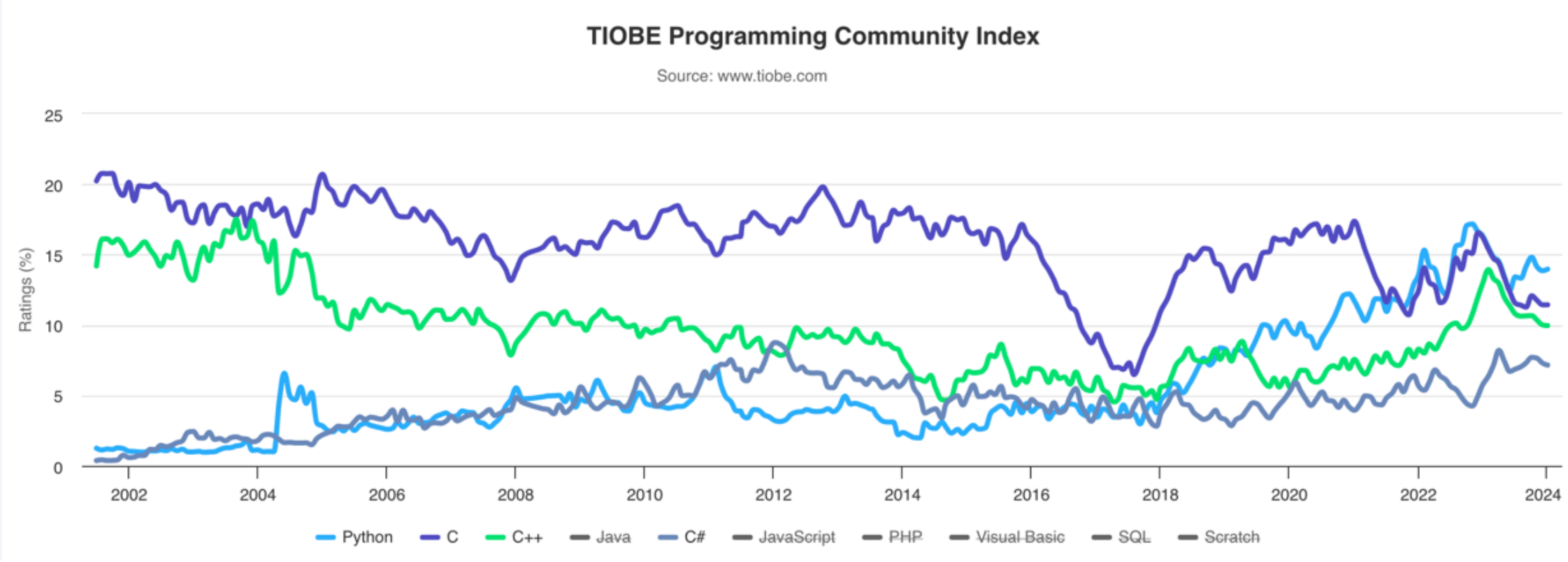
«Das Halteproblem beschreibt die Frage, ob die Ausführung eines Algorithmus zu einem Ende gelangt. Obwohl das für viele
Algorithmen leicht beantwortet werden kann, konnte der Mathematiker Alan Turing beweisen, dass es keinen Algorithmus
gibt, der diese Frage für alle möglichen Algorithmen und beliebige Eingaben beantwortet.»
(https://de.wikipedia.org/wiki/Halteproblem)

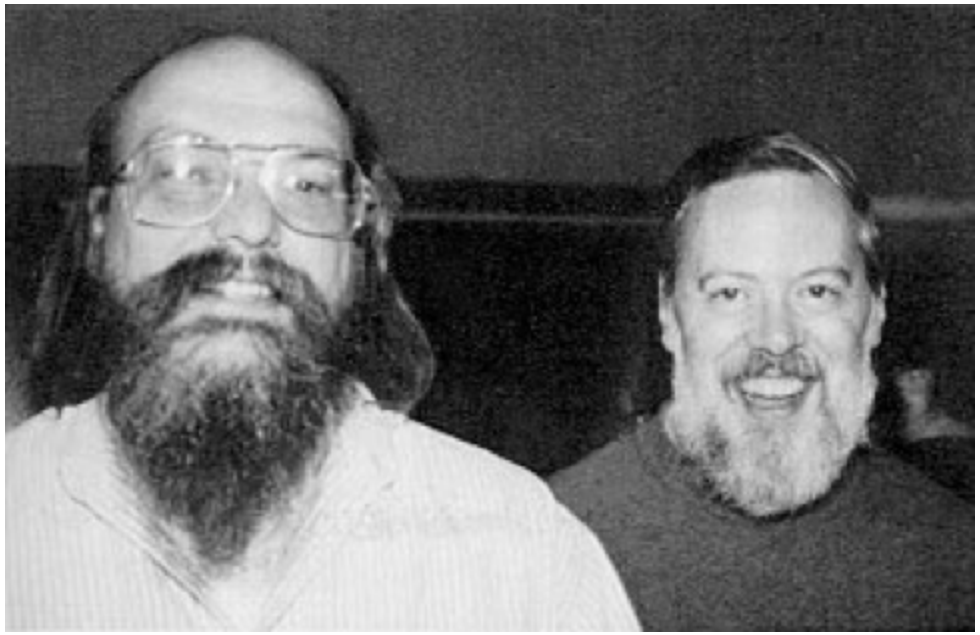
Wir müssen sicherstellen, dass unsere Programme nicht unabsichtlich endlos weiterlaufen!

Turing-Test



Software


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022
2024
Python
C
C++
Java
C#
JavaScript
PHP
Visual Basic
SQL
Scratch



Ken Thompson, Dennis Ritchie

C Keywords (Auswahl)

```
bool (C23)   extern       sizeof         default     return
false (C23)  float        static         do          volatile
break        for          struct         double      short
case         goto         switch         else        signed
char         if           true (C23)     unsigned    register
const        int          typedef        void        union
continue     long
```

Python Keywords

```
False       await       else        import      pass
None        break       except      in          raise
True        class       finally     is          return
and         continue    for         lambda      try
as          def         from        nonlocal    while
assert      del         global      not         with
async       elif        if          or          yield
```

Go Keywords

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

https://go.dev/ref/spec#Keywords

Hochsprache zu Maschinencode

4. Generation

- SQL
- Unix Shell
- LabVIEW
- Stata
- R
- MATLAB
- MaxMSP

Programmiersprachen der 3. Generation

- ALGOL
- Cobol
- Fortran
- C, C++
- C#
- Java
- Python
- Go
- Rust
- JavaScript
- etc.

Rust

```rust
pub fn square(num: i32) -> i32 {
    num * num
}
```

Assembler (2. Generation)

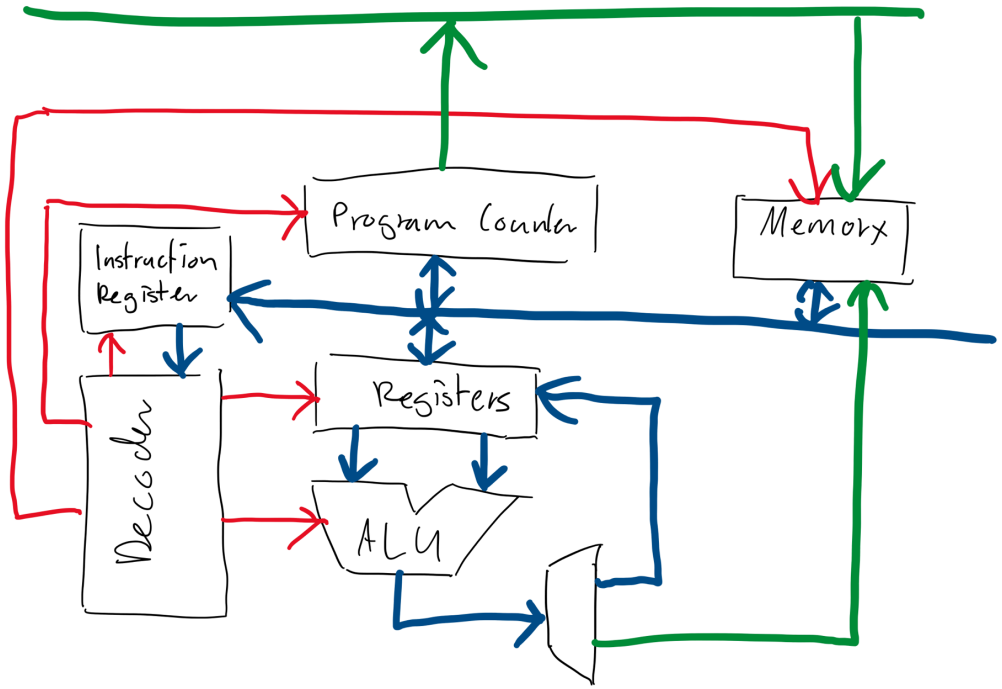
Assembler

```
square:
        push    {r7, lr}
        sub     sp, #8
        smull   r1, r0, r0, r0
        mov     r2, r1
        str     r2, [sp, #4]
        cmp.w   r0, r1, asr #31
        bne     .LBB0_2
        b       .LBB0_1
.LBB0_1:
        ldr     r0, [sp, #4]
        add     sp, #8
        pop     {r7, pc}
.LBB0_2:
        ldr     r0, .LCPI0_0
.LPC0_0:
        add     r0, pc
        ldr     r2, .LCPI0_1
.LPC0_1:
        add     r2, pc
        movs    r1, #33
        bl      core::panicking::panic
        .inst.n 0xdefe
.LCPI0_0:
        .long   str.0-(.LPC0_0+4)
.LCPI0_1:
        .long   .L__unnamed_1-(.LPC0_1+4)
.L__unnamed_2:
        .ascii  "/app/example.rs"
.L__unnamed_1:
        .long   .L__unnamed_2
        .asciz  "\017\000\000\000\013\000\000\000\005\000\000"

str.0:
        .ascii  "attempt to multiply with overflow"
```

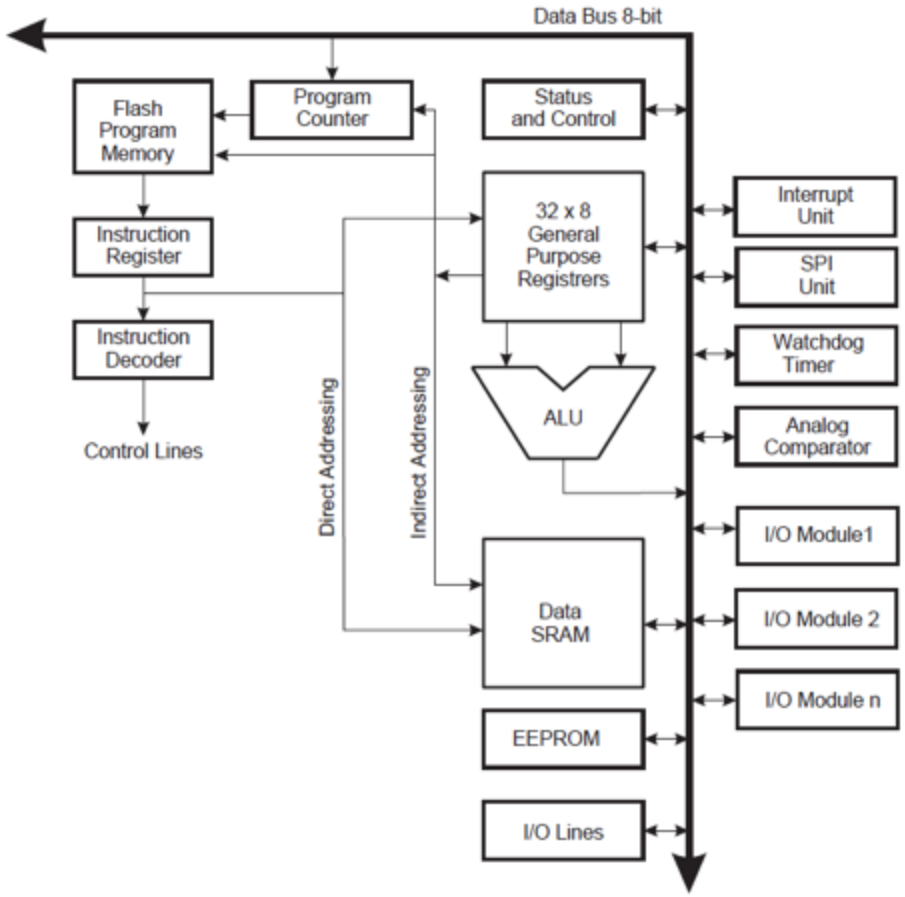
Maschinensprache (1. Generation)

Aufbau und Funktion eines Microprozessors

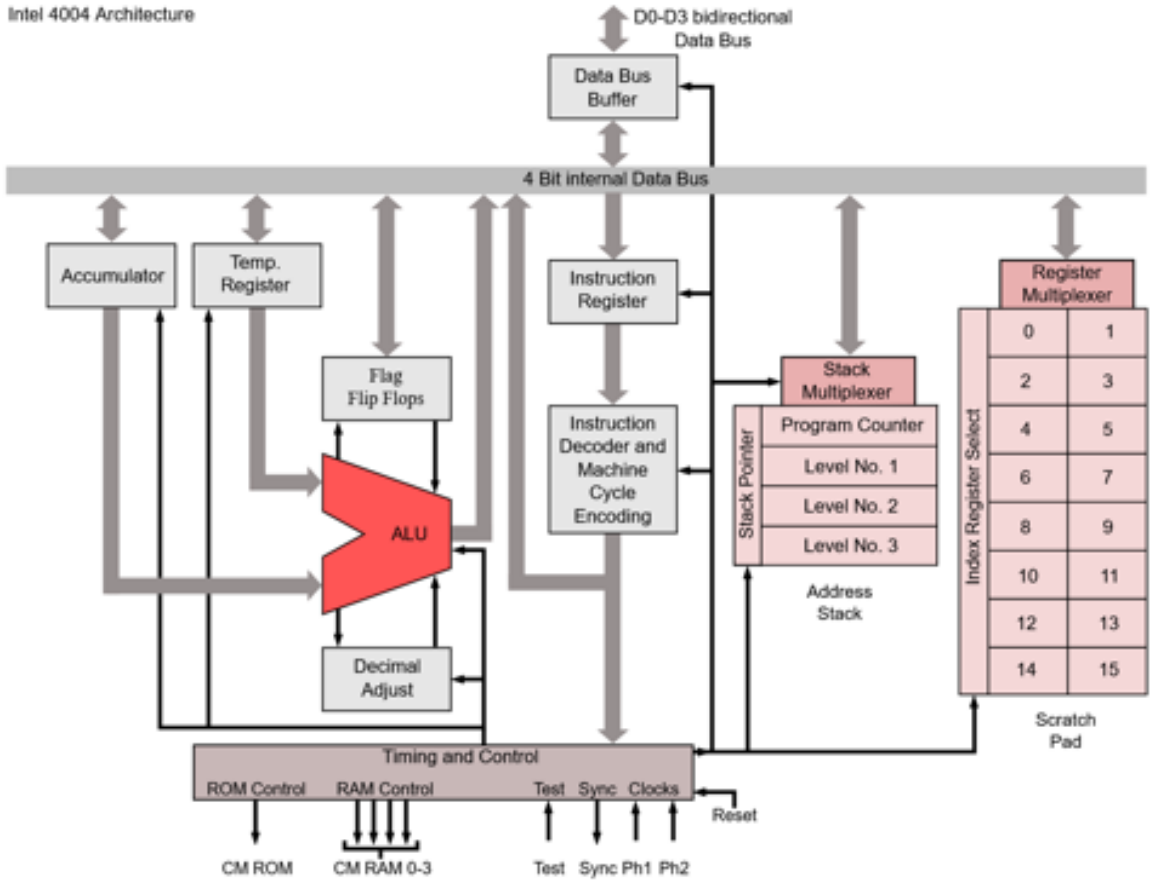


(vgl. https://erik-engheim.medium.com/how-does-a-microprocessor-run-a-program-11744ab47d04)

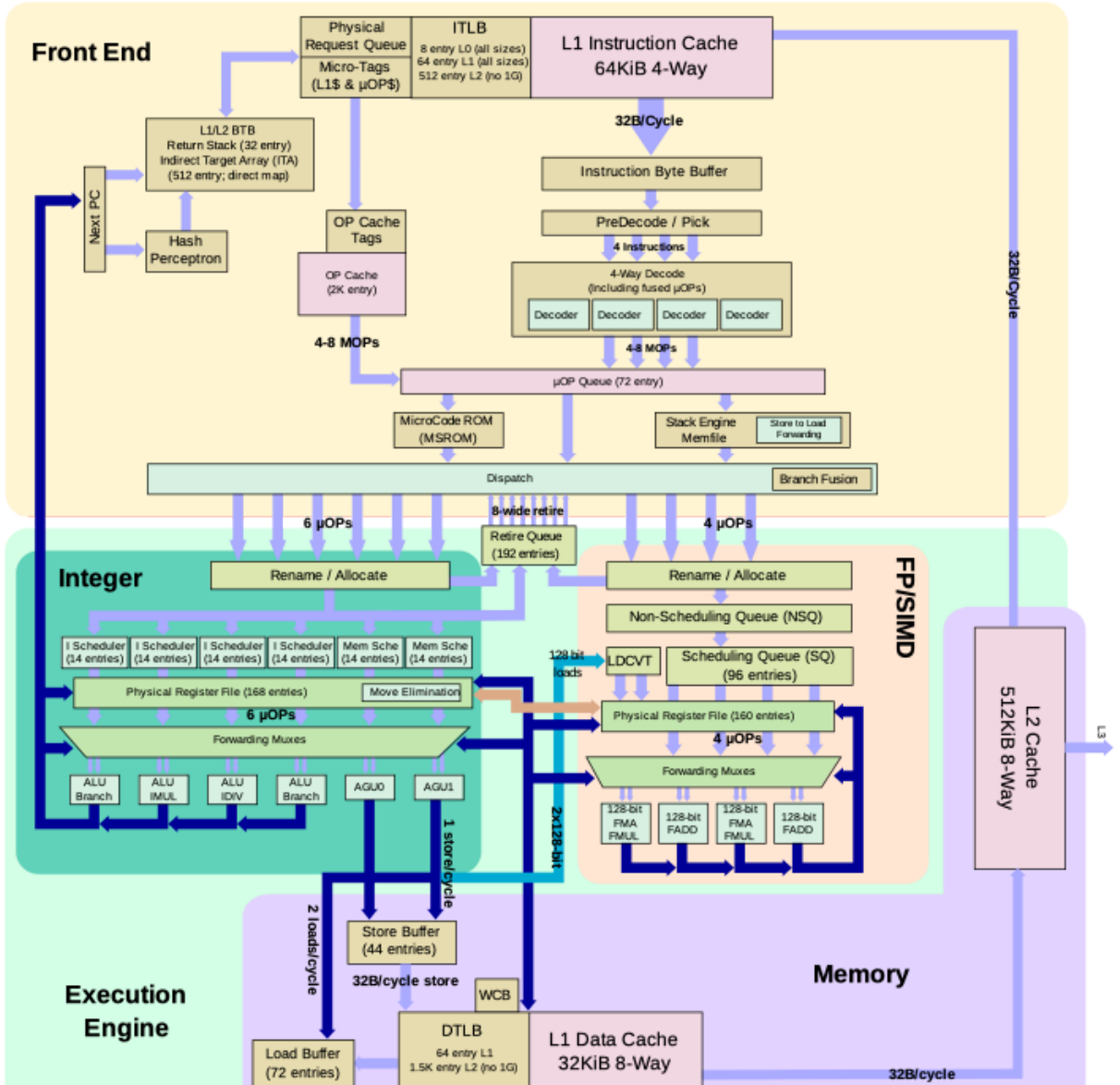
AVR Architektur Blockschaltbild



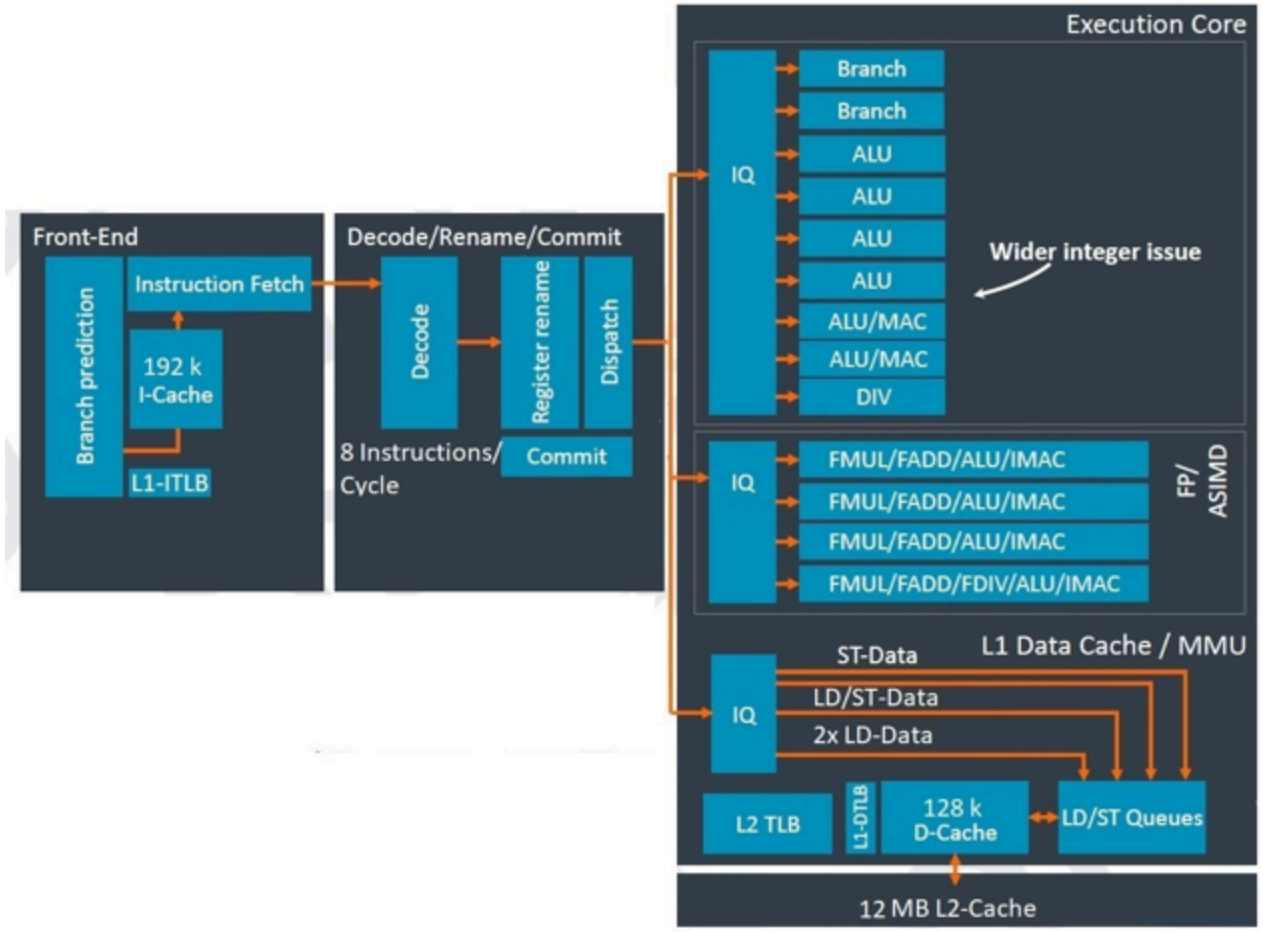
1971: Intel 4004



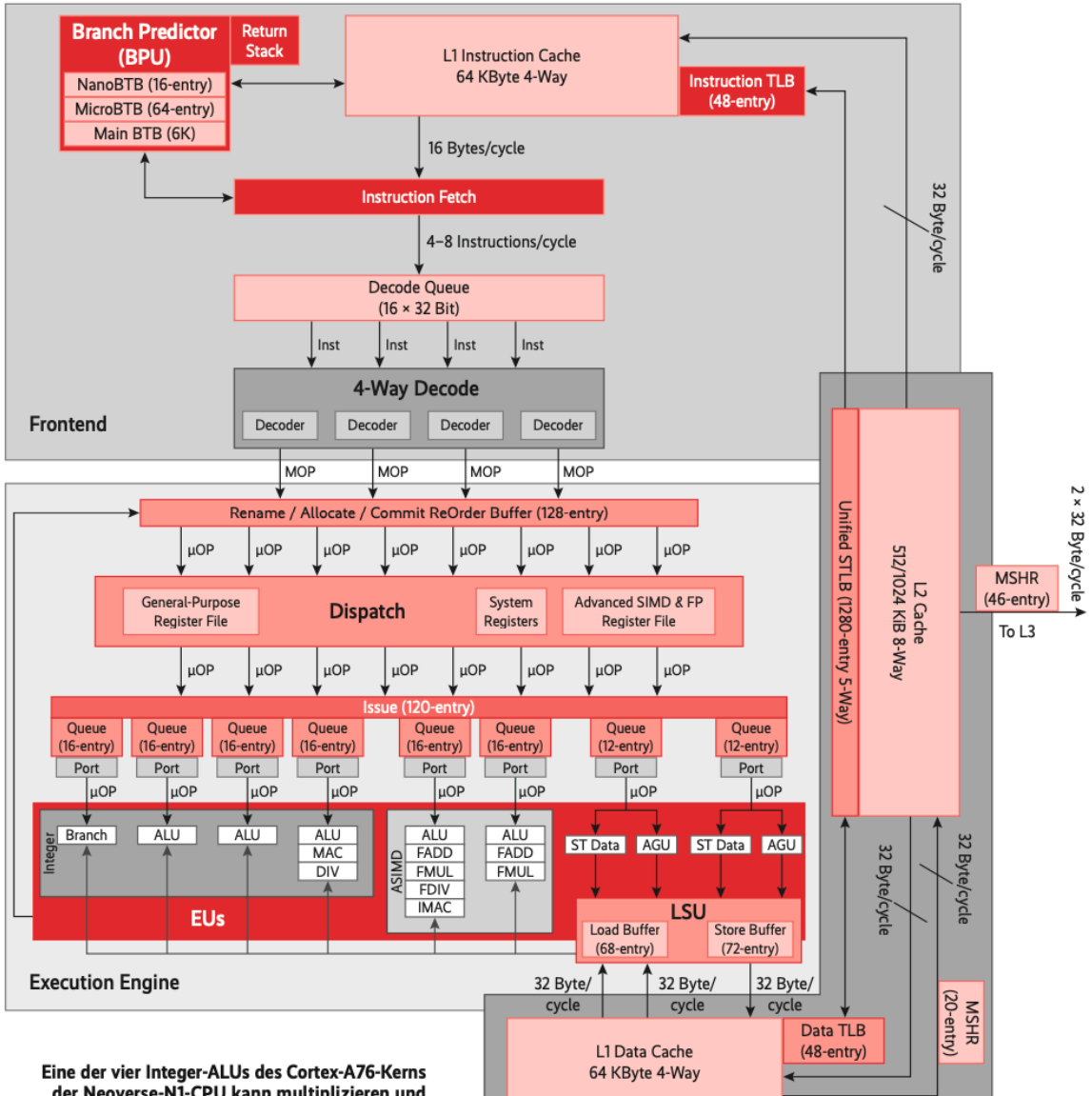
AMD Threadripper



Apple M1


Front-End
Branch prediction
Instruction Fetch
192 k
I-Cache
L1-ITLB
Decode/Rename/Commit
Decode
Register rename
Dispatch
8 Instructions/
Cycle
Commit
Execution Core
IQ
Branch
Branch
ALU
ALU
ALU
ALU
ALU/MAC
ALU/MAC
DIV
Wider integer issue
IQ
FMUL/FADD/ALU/IMAC
FMUL/FADD/ALU/IMAC
FMUL/FADD/ALU/IMAC
FMUL/FADD/FDIV/ALU/IMAC
FP/
ASIMD
L1 Data Cache / MMU
IQ
ST-Data
LD/ST-Data
2x LD-Data
L2 TLB
L1-DTLB
128 k
D-Cache
LD/ST Queues
12 MB L2-Cache

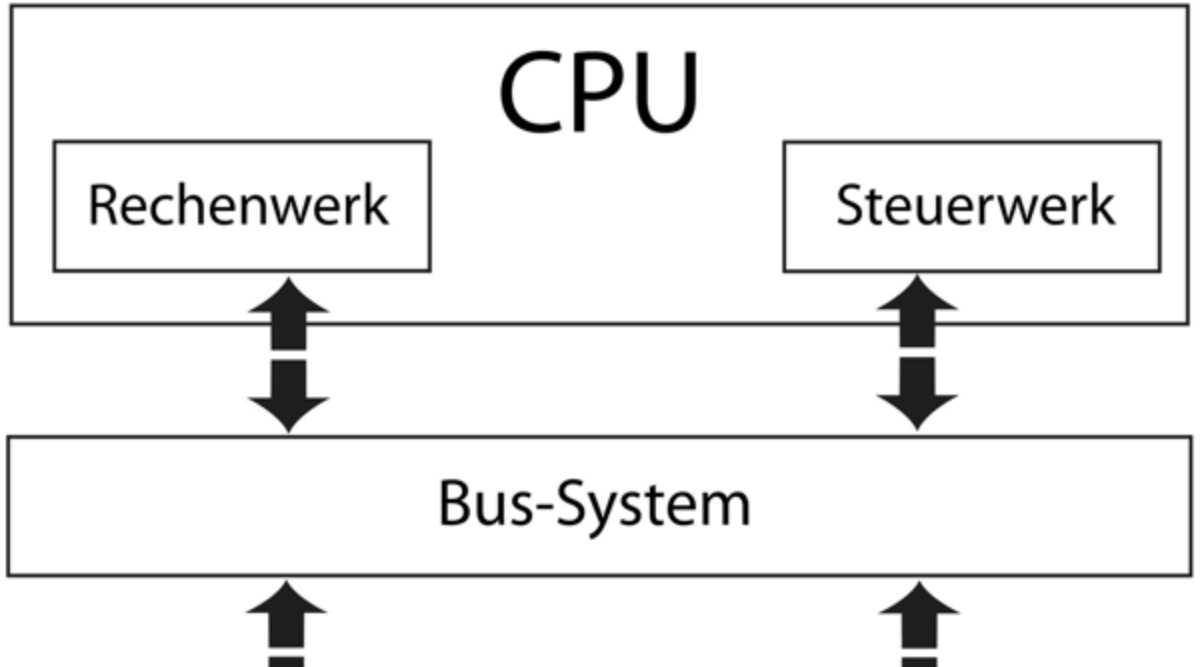
ARM Cortex A67



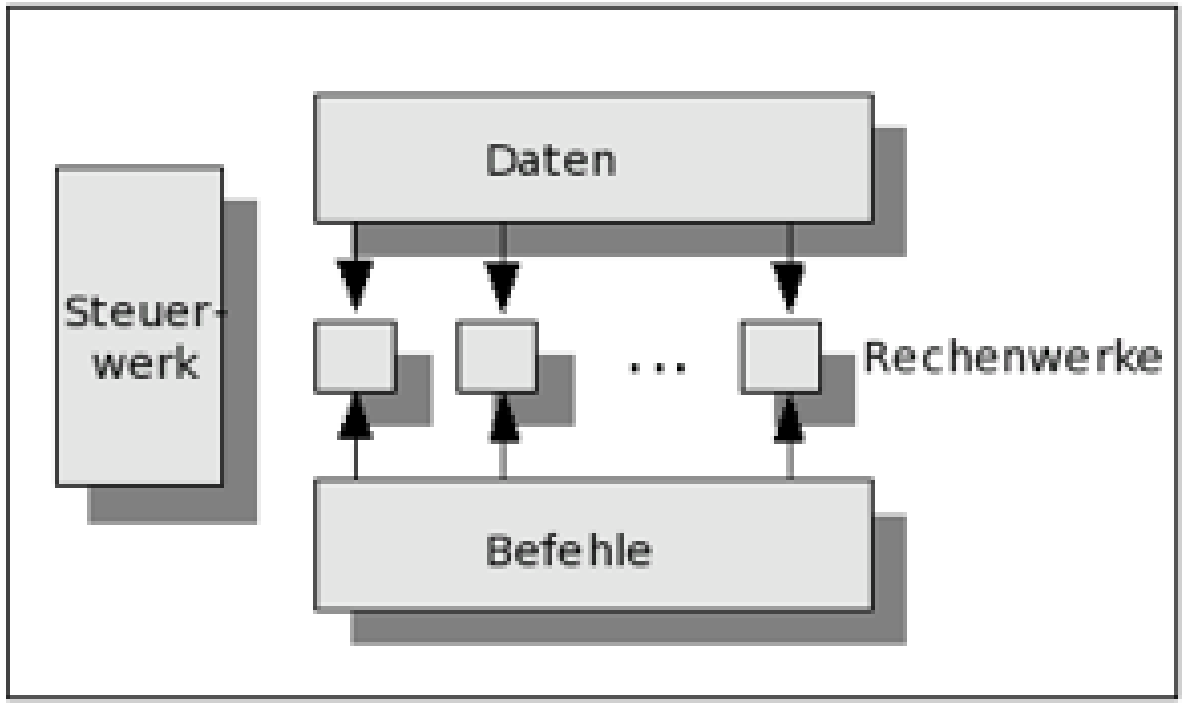
Eine der vier Integer-ALUs des Cortex-A76-Kerns der Neoverse-N1-CPU kann multiplizieren und

von Neumann Architektur

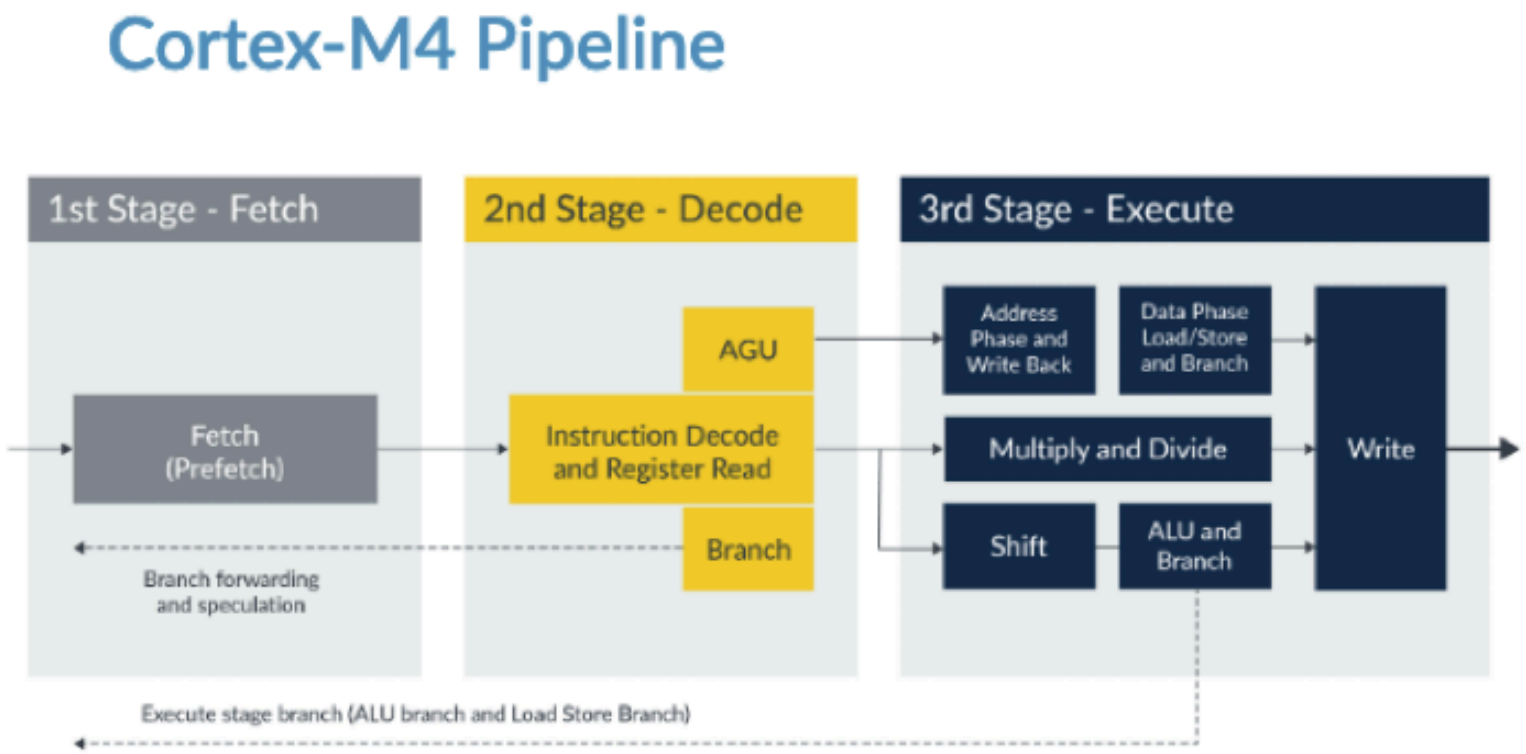
- Befehle werden aus einer Zelle des Speichers gelesen und dann ausgeführt.
- Normalerweise wird dann der Inhalt des Befehlszählers um Eins erhöht.
- Es gibt Verzweigungs-Befehle, die in Abhängigkeit vom Wert eines Entscheidungs-Bit den Befehlszähler um Eins erhöhen
  oder um einen anderen Wert verändern



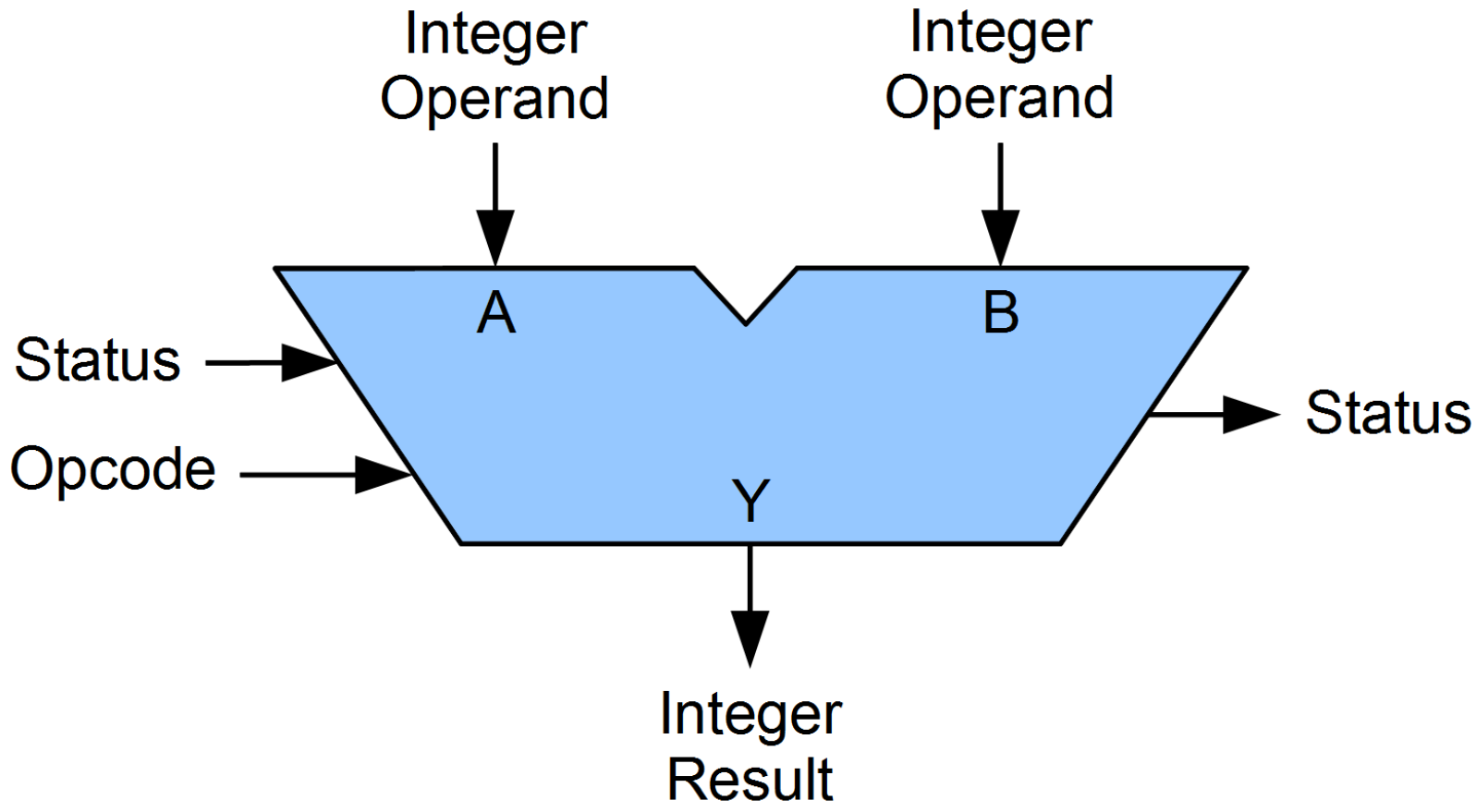
Harvard Architektur



Fetch - Decode - Execute



Arithmetic Logic Unit (ALU)



Mindestens:

- Addition (ADD)
- Negation (NOT)
- Konjunktion (AND)

Zusätzlich (Auswahl):

- Subtraktion
- Vergleich
- Multiplikationen / Division
- Oder
- Shift / Rotation

Instruction Set

http://lyons42.com/AVR/Opcodes/AVRAllOpcodes.html

A64 Instruction Set

C4.1 A64 instruction set encoding

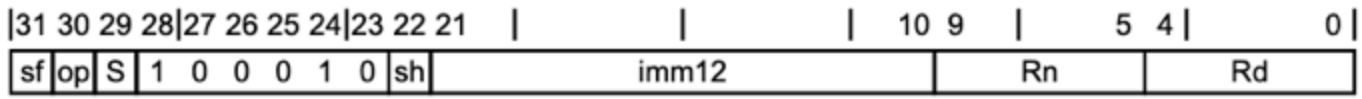
The A64 instruction encoding is:

Table C4-1 Main encoding table for the A64 instruction set

Decode fields op0	Decode group or instruction page
0000	*Reserved* on page C4-284.
0001	Unallocated.
0010	SVE instructions. See *The Scalable Vector Extension (SVE)* on page A2-110.
0011	Unallocated.
100x	*Data Processing -- Immediate* on page C4-284.
101x	*Branches, Exception Generating and System instructions* on page C4-289.
x1x0	*Loads and Stores* on page C4-298.
x101	*Data Processing -- Register* on page C4-332.
x111	*Data Processing -- Scalar Floating-Point and Advanced SIMD* on page C4-342.

Table C4-3 Encoding table for the Data Processing -- Immediate group

Decode fields op0	Decode group or instruction page
00x	*PC-rel. addressing* on page C4-285
010	*Add/subtract (immediate)* on page C4-285
011	*Add/subtract (immediate, with tags)* on page C4-286
100	*Logical (immediate)* on page C4-286
101	*Move wide (immediate)* on page C4-287
110	*Bitfield* on page C4-288
111	*Extract* on page C4-288



Decode fields			Instruction page
sf	op	S	
0	0	0	ADD (immediate) - 32-bit variant
0	0	1	ADDS (immediate) - 32-bit variant
0	1	0	SUB (immediate) - 32-bit variant
0	1	1	SUBS (immediate) - 32-bit variant
1	0	0	ADD (immediate) - 64-bit variant
1	0	1	ADDS (immediate) - 64-bit variant
1	1	0	SUB (immediate) - 64-bit variant
1	1	1	SUBS (immediate) - 64-bit variant

Instruction Set

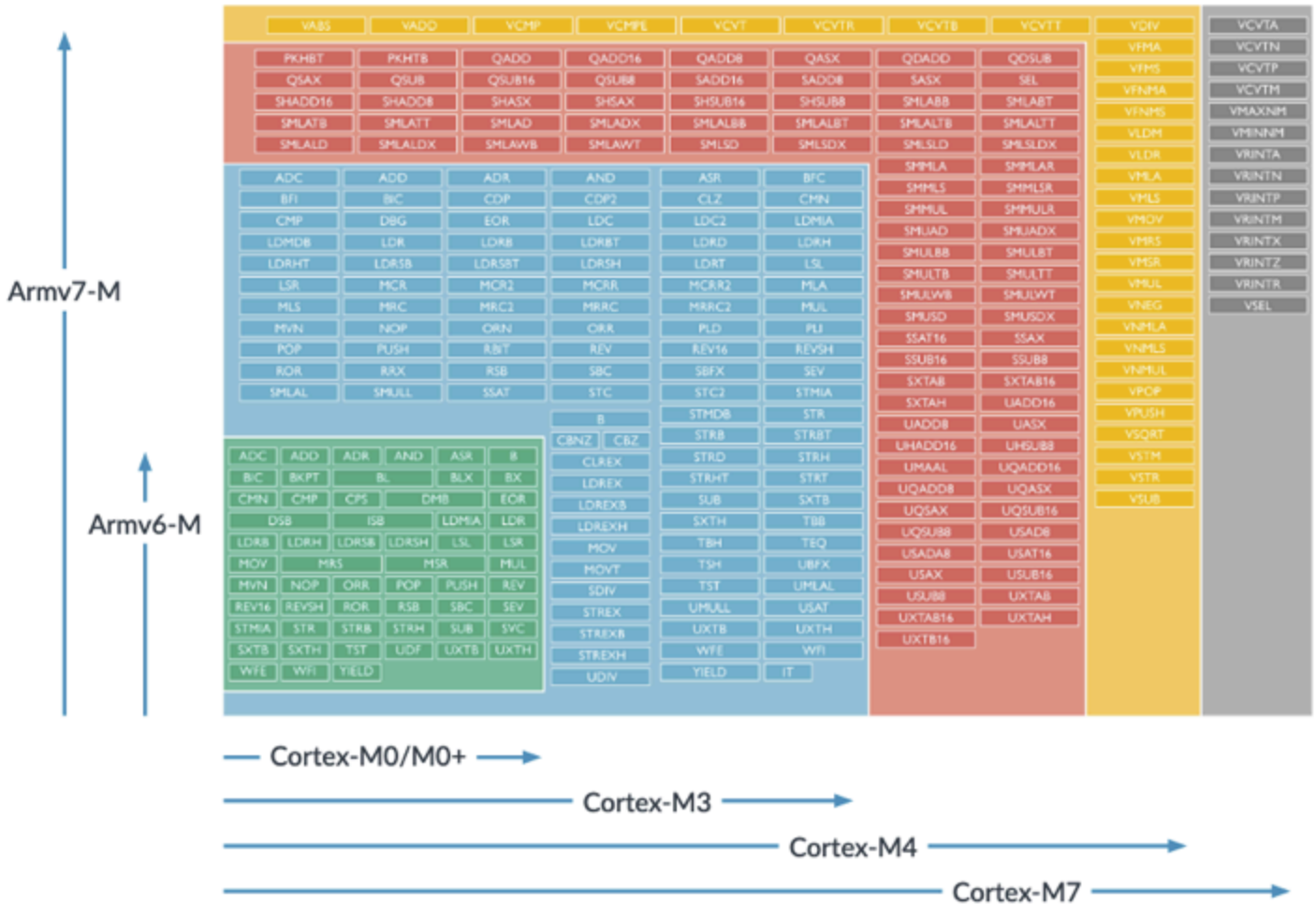


Figure 5: Instruction set

Reduced Instruction Set Computer (RISC)

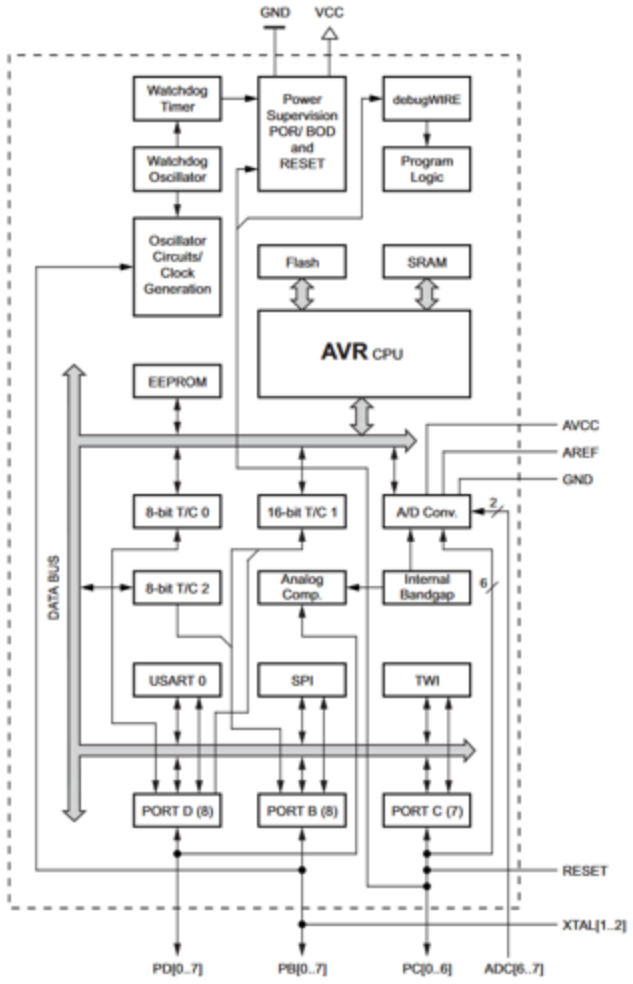
- Opcode hat eine feste Länge
- Meistens 1 Takt pro Operation
- Load/Store Architektur: Separate Lade und Speicher-Befehle
- Hohe Anzahl an Registern für Zwischenresultate
- Oft Harvard-Architektur
- Grundsätzlich: Einfachere Architektur, einfacher für Compiler
- Alles andere: **CISC**

Reduced Instruction Set Computer (RISC)

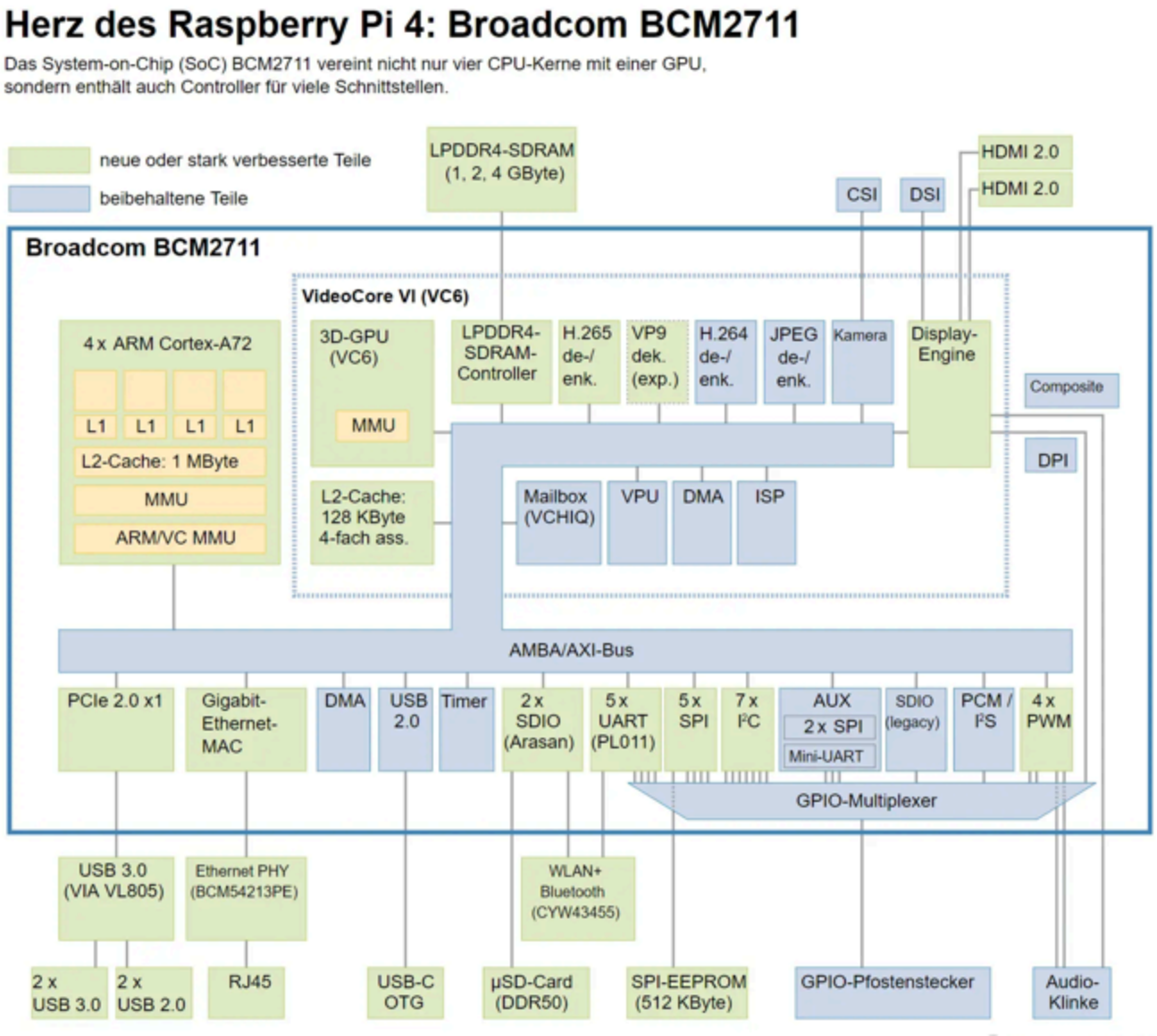
- Besser geeignet für "moderne" Compiler
- Intel / AMD haben lange den CPU Markt mit CISC CPUs dominiert
- Im mobile und embedded Bereich ist ARM (RISC) extrem verbreitet
- Seit 2020 gibt es auch im Desktop wieder RISC Systeme (Apple M1) mit grossen Vorteilen in der Effizienz
- Verschiedene Hersteller bieten auch für RISC Server-CPUs an die v.a. bei Cloud Anbietern (AWS, Google, etc)
  Verbreitung finden

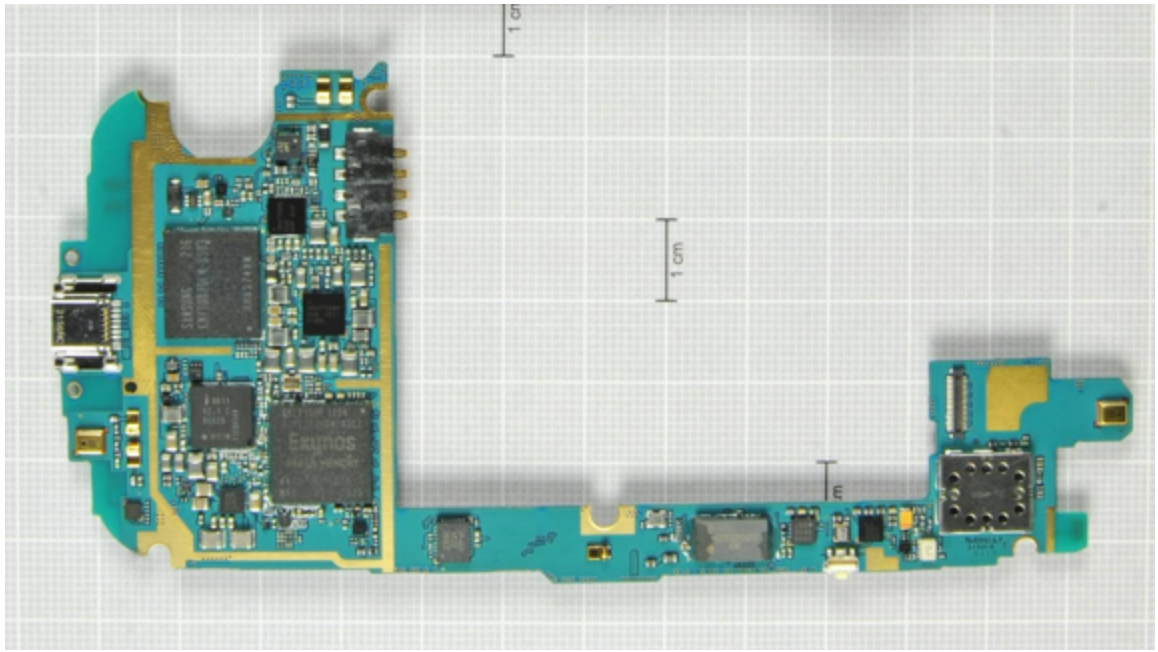
SoC vs Microprocessor vs Microcontroller

Microcontroller: ATmega328P

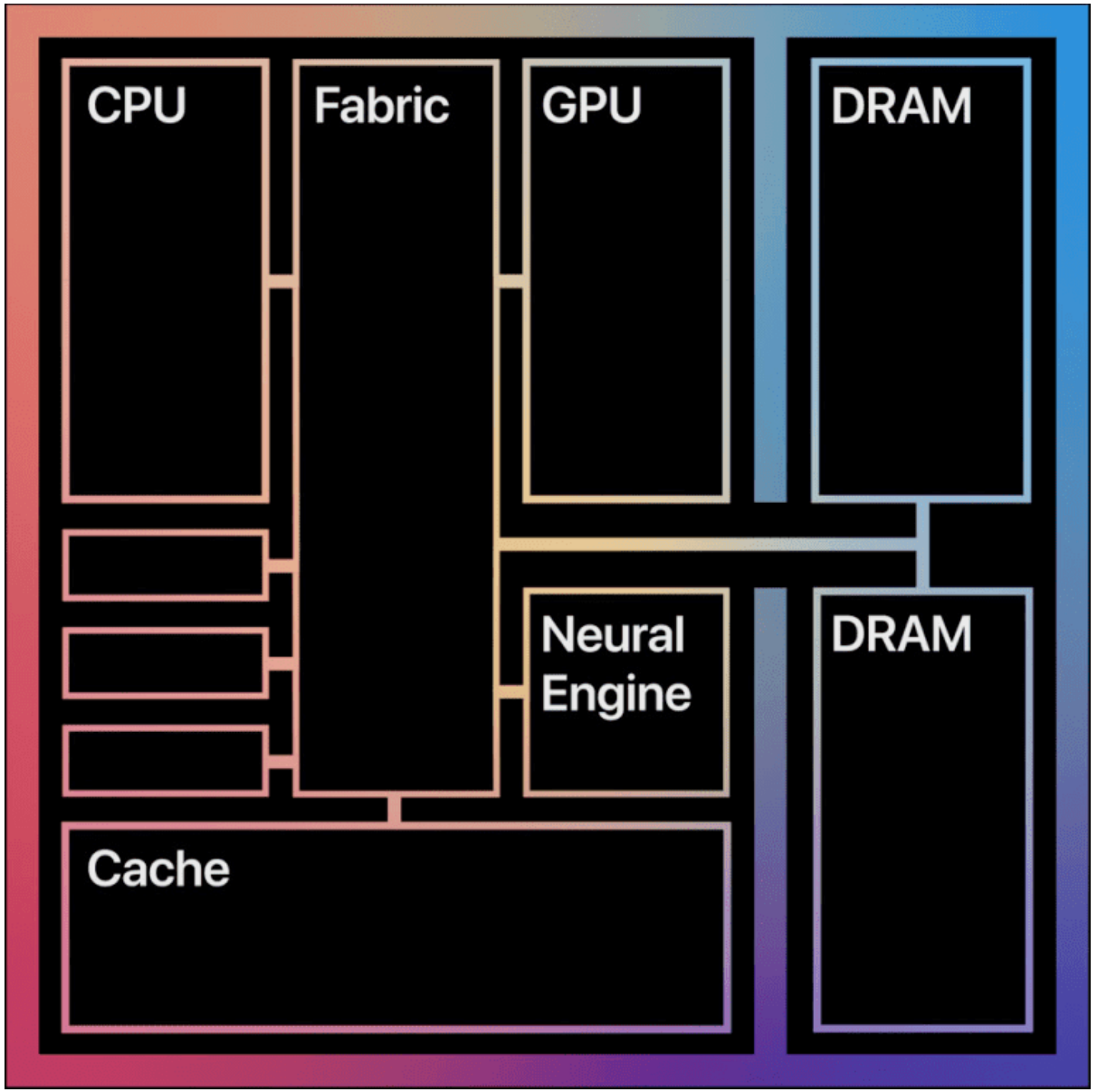


System on Chip (SoC)

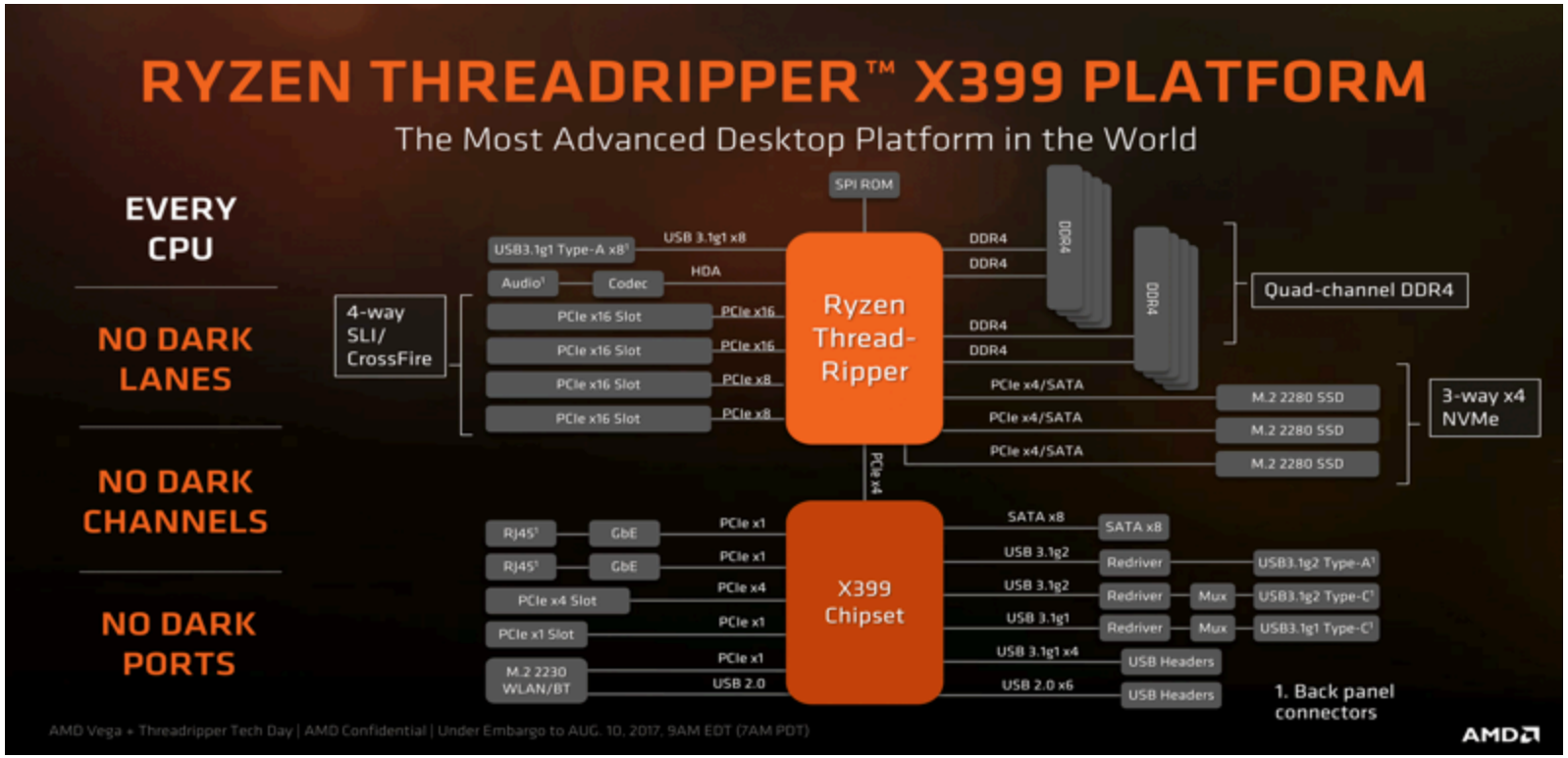


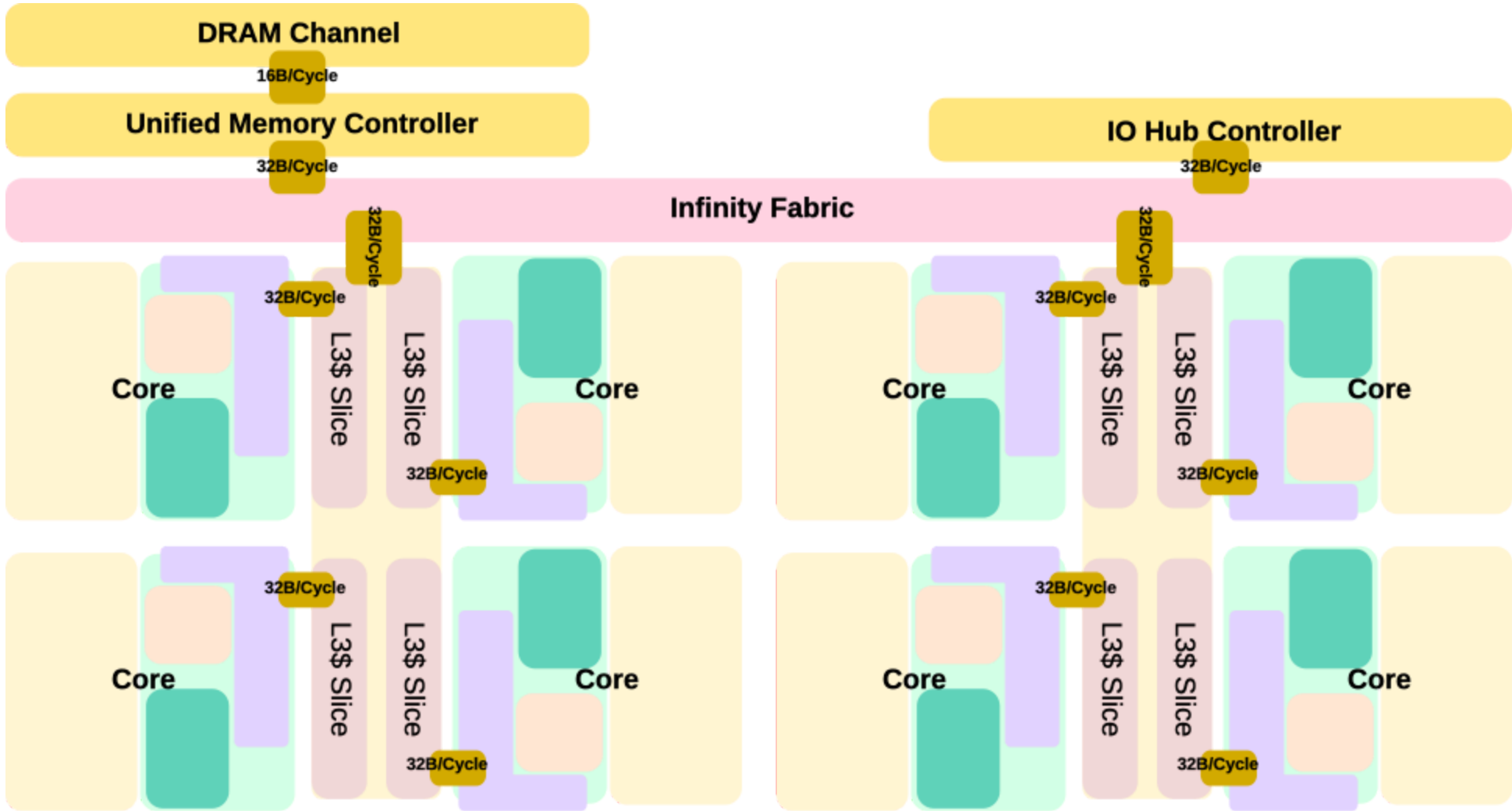


Samsung Galaxy S3


CPU
Fabric
GPU
DRAM
Neural Engine
DRAM
Cache

Microprocessor: AMD Ryzen Threadripper




DRAM Channel
16B/Cycle
Unified Memory Controller
32B/Cycle
IO Hub Controller
32B/Cycle
Infinity Fabric
32B/Cycle
32B/Cycle
L3$ Slice
L3$ Slice
32B/Cycle
Core
Core
32B/Cycle
32B/Cycle
L3$ Slice
L3$ Slice
32B/Cycle
Core
Core
32B/Cycle
32B/Cycle
L3$ Slice
L3$ Slice
32B/Cycle
Core
Core
32B/Cycle
32B/Cycle
L3$ Slice
L3$ Slice
32B/Cycle
Core
Core

Advanced RISC Machine (ARM)

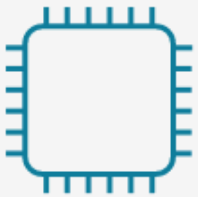
> "Arm licenses processor designs to semiconductor companies that incorporate the technology into their computer chips.
> Licensees pay an up-front fee to gain access to our technology, and a royalty on every chip that uses one of our
> technology designs.
> Typically, the royalty is based on the selling price of the chip."

(https://group.softbank/en/ir/financials/annual_reports/2021/message/segars, 08.01.2024)

Company Highlights

70%

of the world's population uses Arm-based products

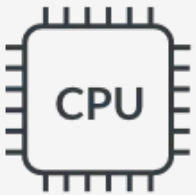


270Bn+

Arm-based chips shipped to date

99%

of smartphones run on Arm-based processors



50%

of all chips with processors are Arm-based

Moore's Law

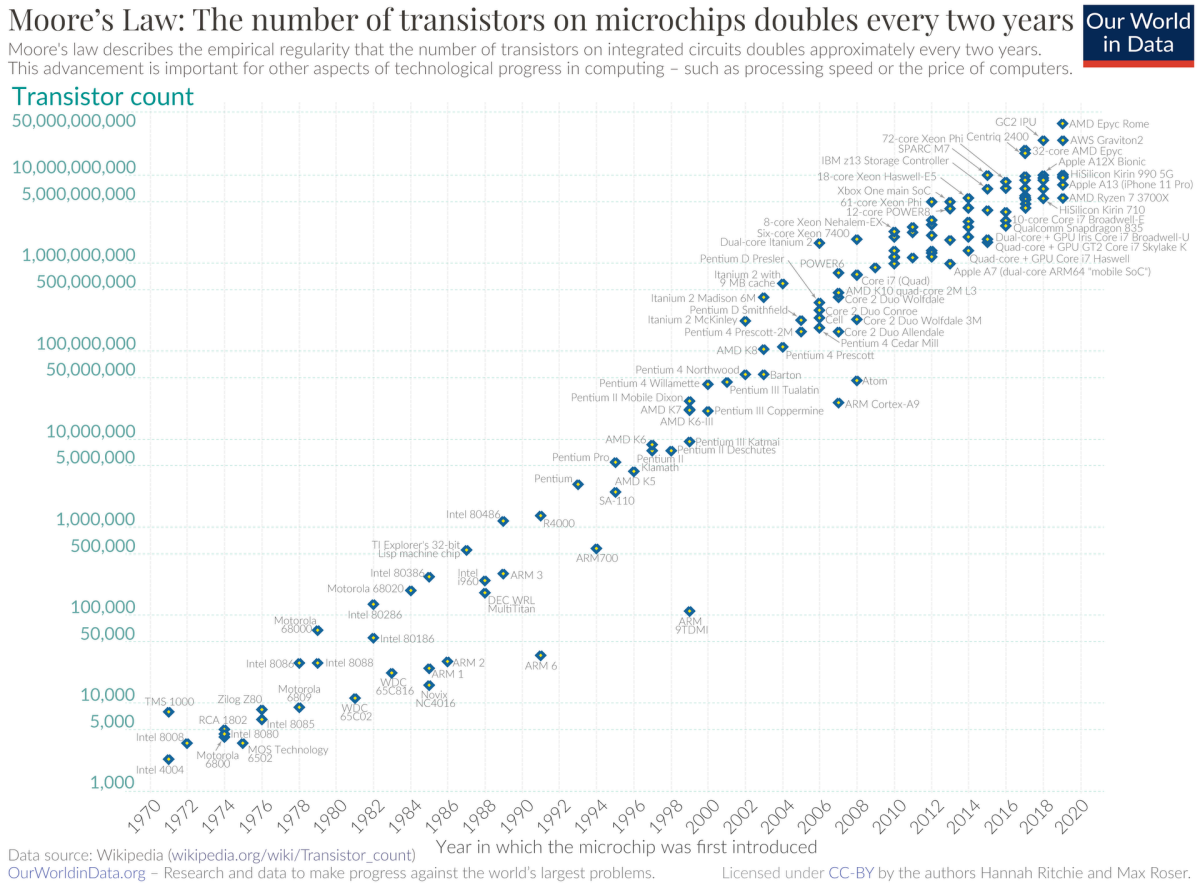
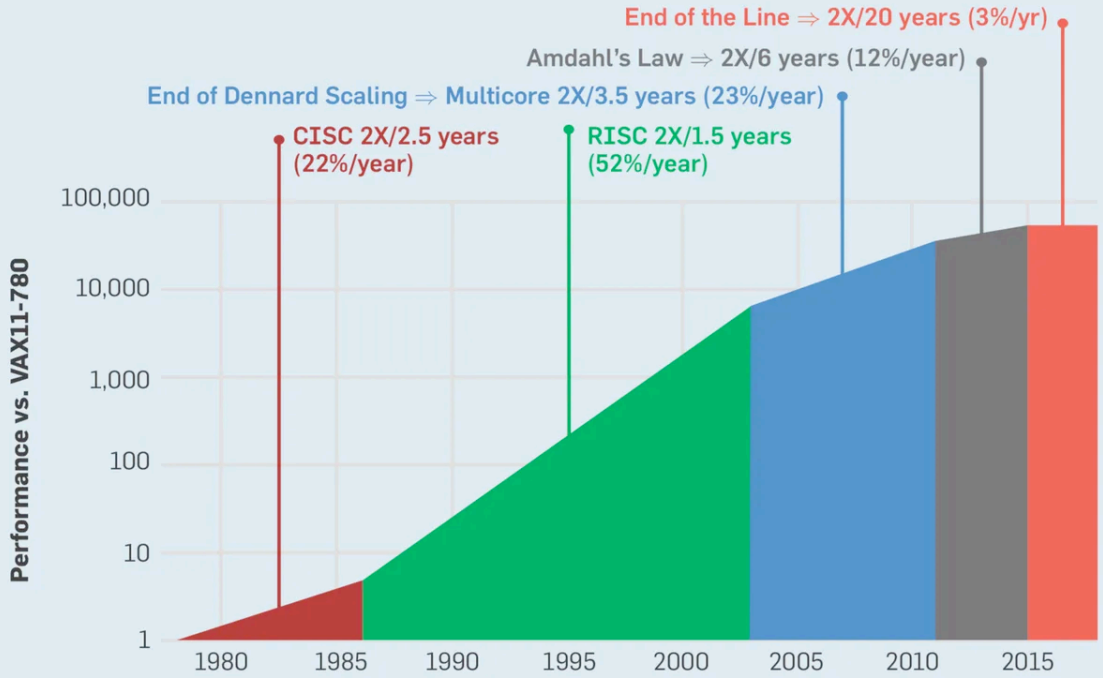

Moore's Law: The number of transistors on microchips doubles every two years
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years.
This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.
Our World in Data
Transistor count
50,000,000,000
10,000,000,000
5,000,000,000
1,000,000,000
500,000,000
100,000,000
50,000,000
10,000,000
5,000,000
1,000,000
500,000
100,000
50,000
10,000
5,000
1,000
GC2 IPU
AMD Epyc Rome
72-core Xeon Phi
Centriq 2400
AWS Graviton2
SPARC M7
32-core AMD Epyc
IBM z13 Storage Controller
Apple A12X Bionic
18-core Xeon Haswell-E5
HiSilicon Kirin 990 5G
Xbox One main SoC
Apple A13 (iPhone 11 Pro)
61-core Xeon Phi
AMD Ryzen 7 3700X
12-core POWER8
HiSilicon Kirin 710
8-core Xeon Nehalem-EX
10-core Core i7 Broadwell-E
Six-core Xeon 7400
Qualcomm Snapdragon 835
Dual-core Itanium 2
Dual-core + GPU Iris Core i7 Broadwell-U
Quad-core + GPU GT2 Core i7 Skylake K
Pentium D Presler
POWER6
Quad-core + GPU Core i7 Haswell
Apple A7 (dual-core ARM64 "mobile SoC")
Itanium 2 with 9 MB cache
Core i7 (Quad)
AMD K10 quad-core 2M L3
Itanium 2 Madison 6M
Core 2 Duo Wolfdale
Pentium D Smithfield
Core 2 Duo Conroe
Itanium 2 McKinley
Cell
Core 2 Duo Wolfdale 3M
Pentium 4 Prescott-2M
Core 2 Duo Allendale
Pentium 4 Cedar Mill
AMD K8
Pentium 4 Prescott
Pentium 4 Northwood
Barton
Pentium 4 Willamette
Atom
Pentium III Tualatin
Pentium II Mobile Dixon
ARM Cortex-A9
AMD K7
Pentium III Coppermine
AMD K6-III
AMD K6
Pentium III Katmai
Pentium II Deschutes
Pentium Pro
Pentium III
Klamath
Pentium
AMD K5
SA-110
Intel 80486
R4000
TI Explorer's 32-bit Lisp machine chip
ARM700
Intel 80386
Intel i960
ARM 3
Motorola 68020
DEC WRL MultiTitan
Intel 80286
ARM 9TDMI
Motorola 68000
Intel 80186
Intel 8086
Intel 8088
ARM 2
ARM 1
ARM 6
WDC 65C816
Motorola 6809
Novix NC4016
TMS 1000
Zilog Z80
WDC 65C02
RCA 1802
Intel 8085
Intel 8008
Intel 8080
MOS Technology 6502
Motorola 6800
Intel 4004
1970
1972
1974
1976
1978
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
Year in which the microchip was first introduced
Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world's largest problems.
Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

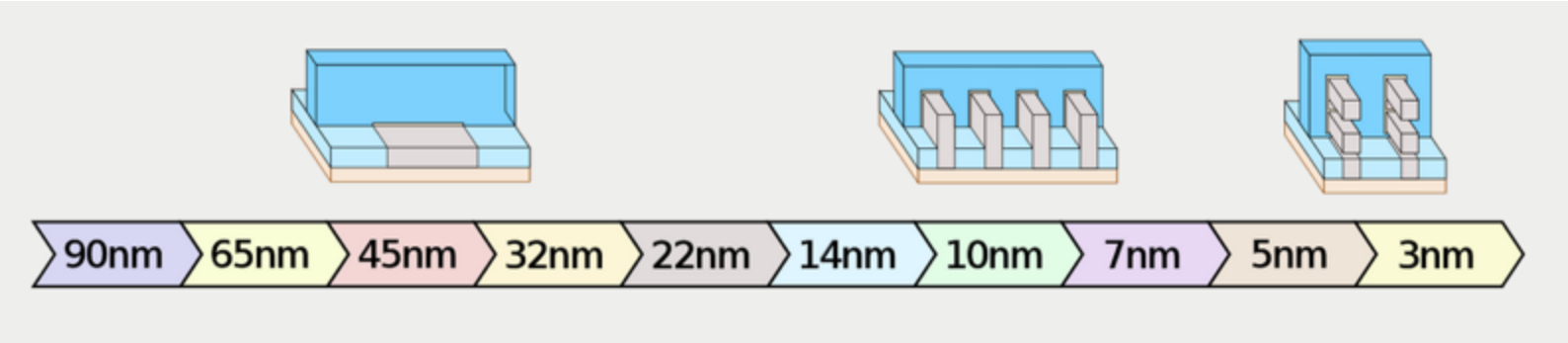
Figure 6. Growth of computer performance using integer programs (SPECintCPU).



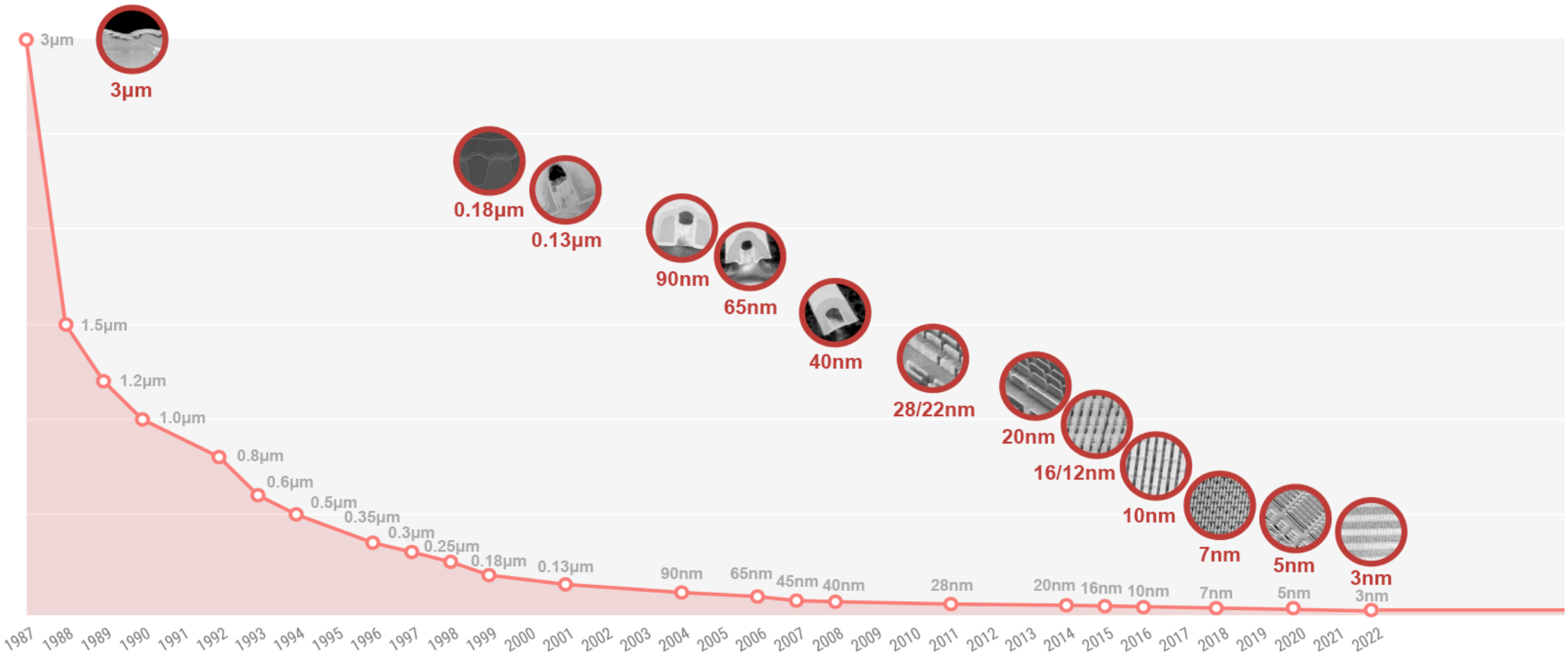
https://www.zdnet.com/article/ai-is-changing-the-entire-nature-of-compute/

(Patterson, Hennessy, 2014, S.44)

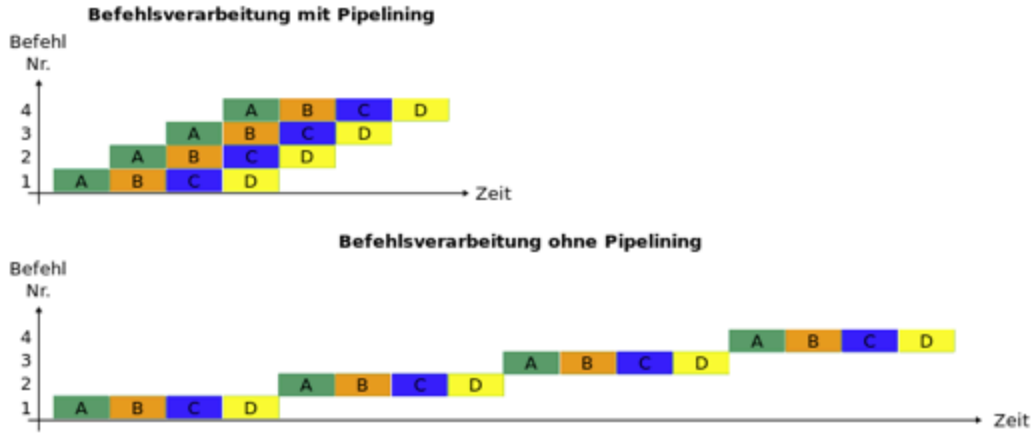
Strukturgrösse



TSMC


3μm
1.5μm
1.2μm
1.0μm
0.8μm
0.6μm
0.5μm
0.35μm
0.3μm
0.25μm
0.18μm
0.13μm
90nm
65nm
45nm
40nm
28nm
20nm
16nm
10nm
7nm
5nm
3nm
3μm
0.18μm
0.13μm
90nm
65nm
40nm
28/22nm
20nm
16/12nm
10nm
7nm
5nm
3nm
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022

Pipelining



A – Befehlscode laden (IF, Instruction Fetch)

In der Befehlsbereitstellungsphase wird der Befehl, der durch den Befehlszähler adressiert ist, aus dem Arbeitsspeicher geladen. Der Befehlszähler wird anschließend hochgezählt.

B – Instruktion dekodieren und Laden der Daten (ID, Instruction Decoding)

In der Dekodier- und Ladephase wird der geladene Befehl dekodiert (1. Takthälfte) und die notwendigen Daten aus dem Arbeitsspeicher und dem Registersatz geladen (2. Takthälfte).

C – Befehl ausführen (EX, Execution)

In der Ausführungsphase wird der dekodierte Befehl ausgeführt. Das Ergebnis wird durch den Pipeline-latch gepuffert.

D – Ergebnisse zurückgeben (WB, Write Back)

In der Resultatspeicherphase wird das Ergebnis in den Arbeitsspeicher oder in den Registersatz zurückgeschrieben.

Co-Processors

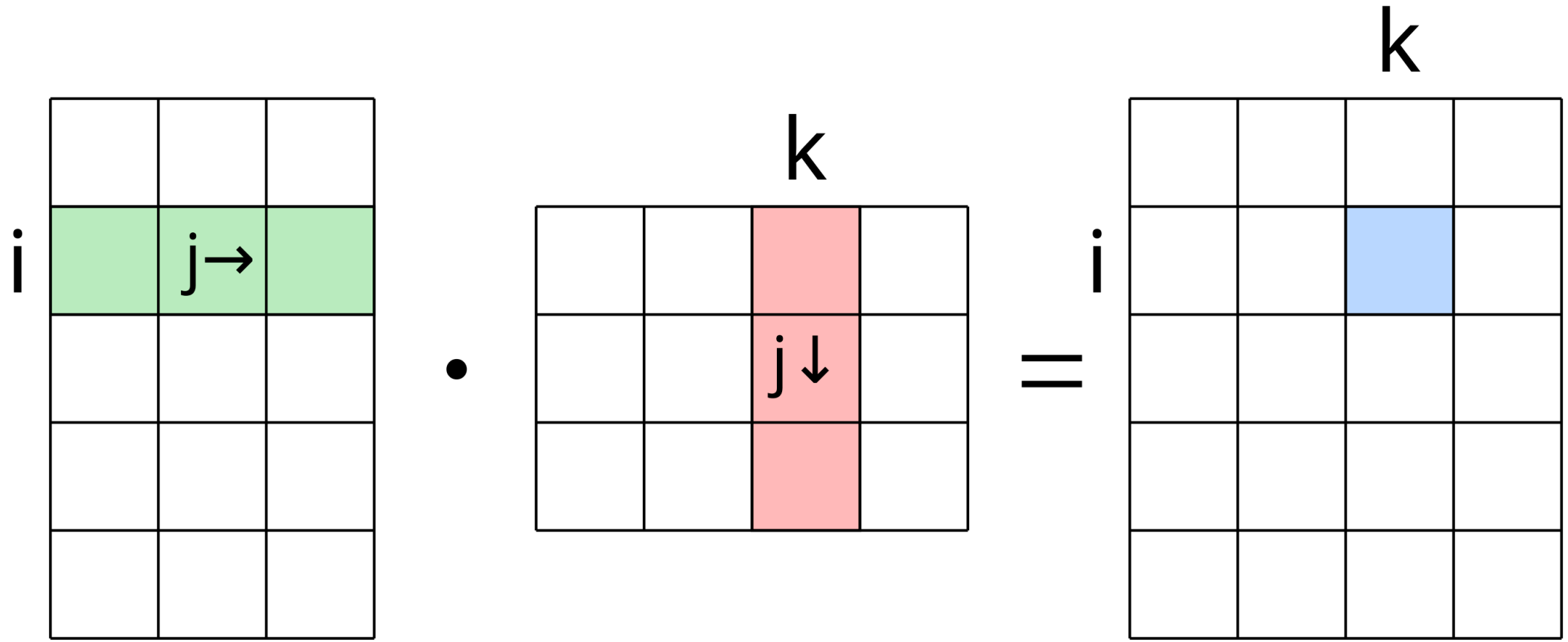
Vektoren

$$a \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = ab_1 + ab_2 + ab_3$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1b_1 + a_2b_2 + a_3b_3$$

Matrizen

$$C = A \cdot B, c_{ik} = \sum_{j=1}^{n} a_{ij} \cdot b_{jk}$$

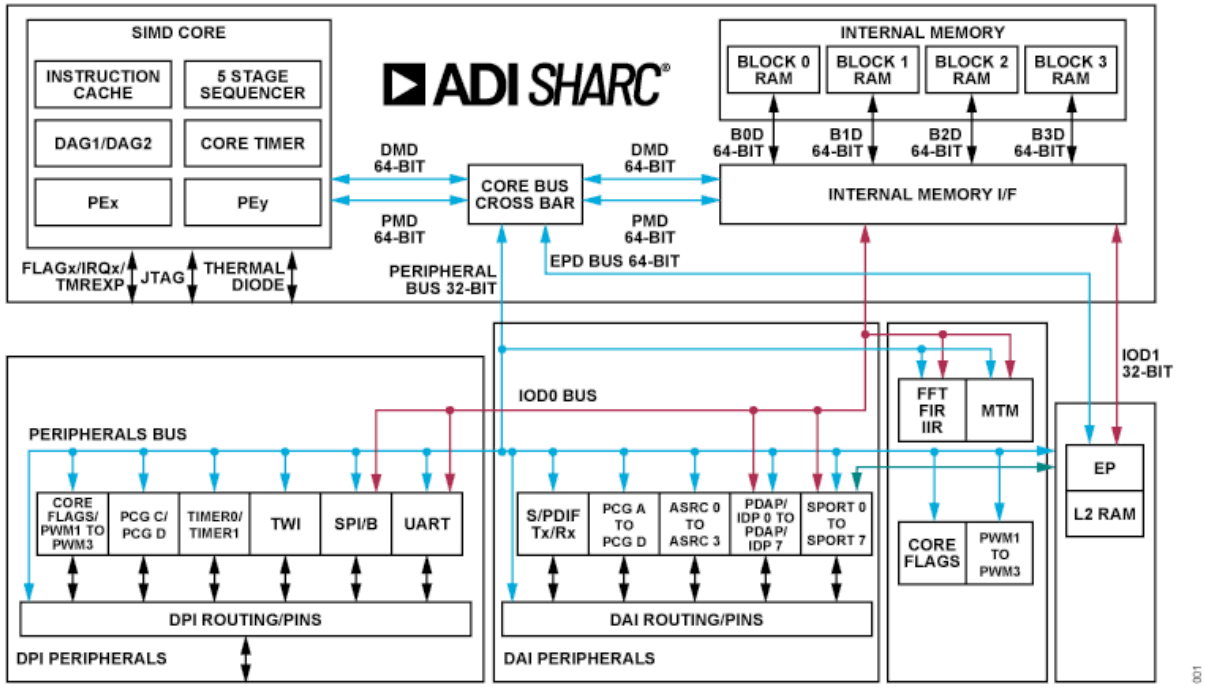


MAC

$$a = a + (b \cdot c)$$

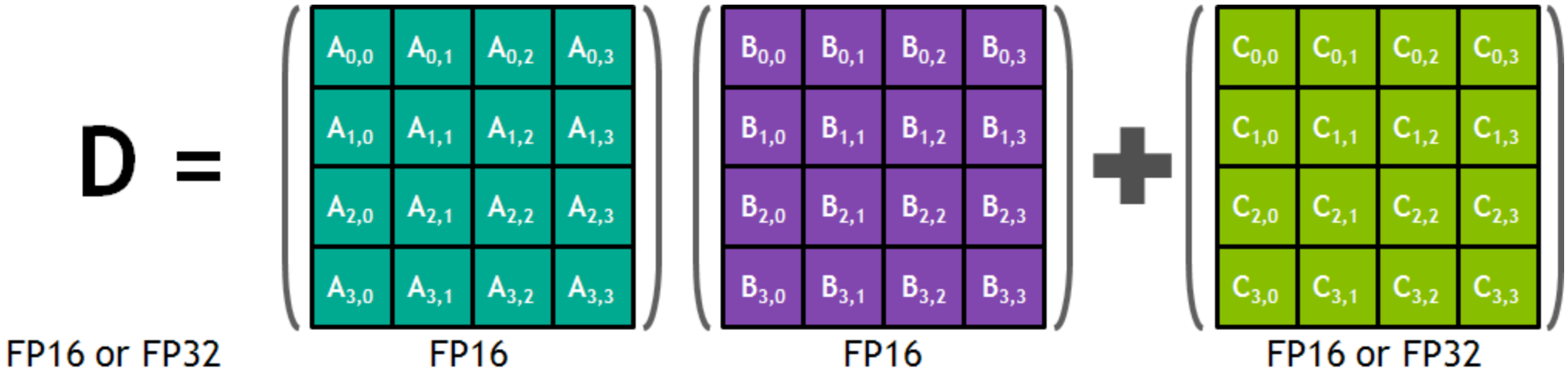
Digital Signal Processors: Vektorprozessoren

- FIR: $y[n] = \sum_{i=1}^{N} b_i \cdot x[n-i]$
- FFT: $\hat{a} = W \cdot a, a = (a_0, \ldots, a_{N-1}), W[k,j] = e^{-w\pi i \frac{jk}{N}}$ (Matrix-Vektor Multiplikation)

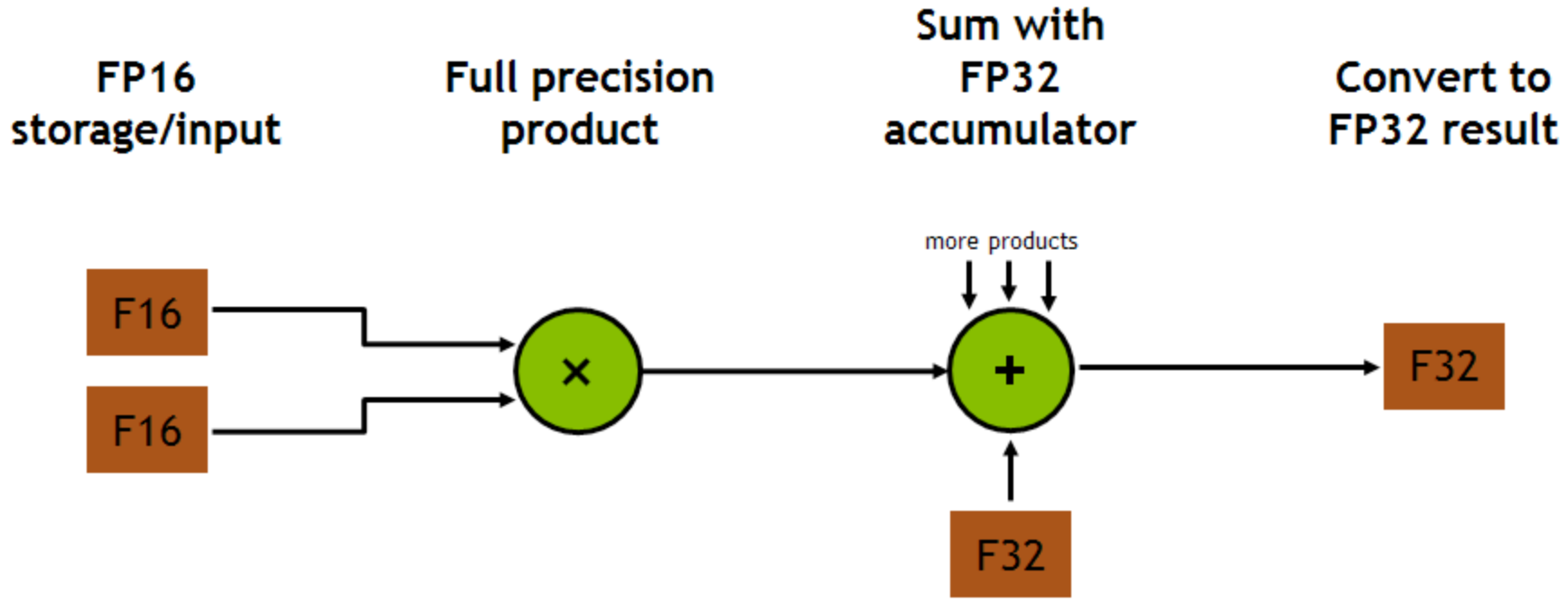


Tensor Cores: Matrixprozessoren

NVIDIA V100 Tensor Cores are programmable matrix-multiply-and-accumulate units



https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/



https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/

GH100 Streaming Multiprocessor (SM)

- INT32: cores for integer math (32-Bit)
- 128 FP32 Cores per SM (32-Bit Floating Point)
- 64 FP64 Cores per SM (64-Bit Floating Point)
- 64 INT32 Cores per SM (32-Bit Integer)
- 4 Tensor Cores per SM

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#architecture-8-x


PCI Express 5.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
L2 Cache
L2 Cache
Memory Controller
HBM3
High-Speed Hub
NVLink

The full implementation of the GH100 GPU includes the following units:

- 8 GPCs, 72 TPCs (9 TPCs/GPC), 2 SMs/TPC, 144 SMs per full GPU
- 18432 FP32 CUDA Cores per full GPU
- 576 Tensor Cores per full GPU
- 6 HBM3 or HBM2e stacks, 12 512-bit memory controllers
- 60 MB L2 cache
- Fourth-generation NVLink and PCIe Gen 5

https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

Durchsatz

Number of Results per Clock Cycle per Multiprocessor

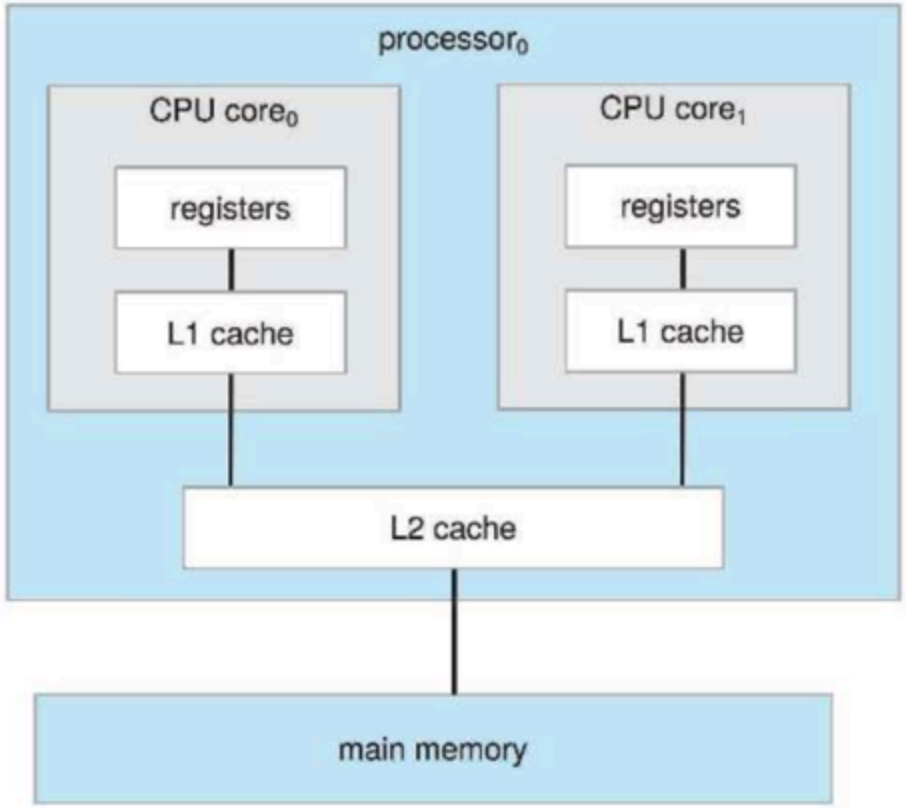
32-bit floating-point add, multiply, multiply-add: 128

D.h. 18'432 FP32 Operation pro Takt.

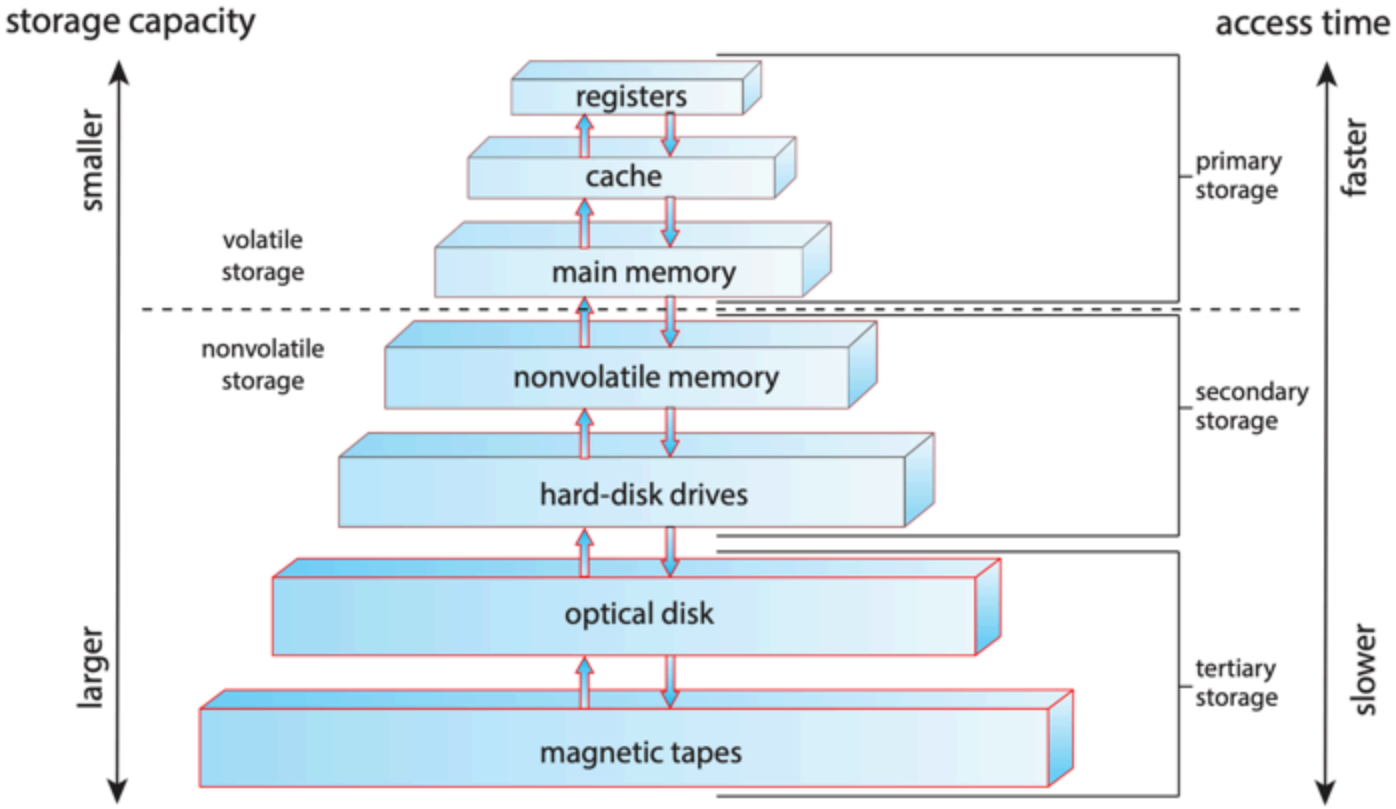
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#arithmetic-instructions

Speicher

Cache



(Silberschatz, 2019)



(Silberschatz, 2019)

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Figure 1.14 Characteristics of various types of storage.

(Silberschatz, 2019)

Zahlendarstellung und Datentypen

- Binäre Zahlen: Für Maschinen einfach darstellbar (2 mögliche Zustände, idR. Spannungen)

Datentypen in Go (Auswahl)

`bool` boolean, 1-bit, true or false

`int8` 8-bit signed integer (-128 bis 127)

`int16` 16-bit signed integer (-32'768 bis 32'767)

`int32` 32-bit signed integer (–2'147'483'648 bis 2'147'483'647)

`uint8` 8-bit unsigned integer (0 bis 255)

`float32` 32-bit IEEE 754 floating-point number (1.2E–38 bis 3.4E38)

`string` "Sequence of Unicode code points"

Statische Typisierung

- Zur Laufzeit hat jedes Objekt einen (Daten)typ
- Im Programmtext hat jeder Ausdruck einen Typ → Der Typ ist zum Zeitpunkt der Kompilierung bekannt
- Vorteile
  - Fehler können früher erkannt werden
  - Effizientere Programme, da keine Typprüfung während der Laufzeit
  - Mehr Optimierungsmöglichkeiten durch Compiler
- statisch typisierte Sprachen: Java, Kotlin, C#, C, Go, Rust

```
final Crossroad crossroad = new Crossroad();
final CrossroadController crossroadController =

final Scene scene = new Scene(crossroadControll
```

Datentypen in Python (Auswahl)

- `str`
- `int` (Kein Limit)
- `float` (64Bit IEEE 754))
- `complex`
- `bool`

Dynamische Typisierung

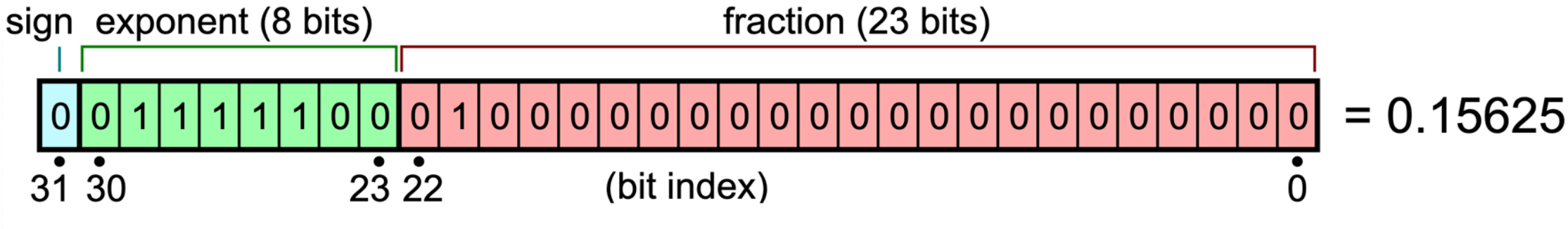
- Zur Laufzeit hat jedes Objekt einen Typ
- Der Typ wird zur Laufzeit geprüft
- Duck Typing: “When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird
  a duck.”
- Vorteile
  - Einfachere Programmierung
- Durch Typehints kann die IDE uns bei der Entwicklung dennoch unterstützen
  - `def greeting(name: str) -> str:`
- dynamisch typisierte Sprachen: PHP, Python, Ruby, JavaScript

Integer

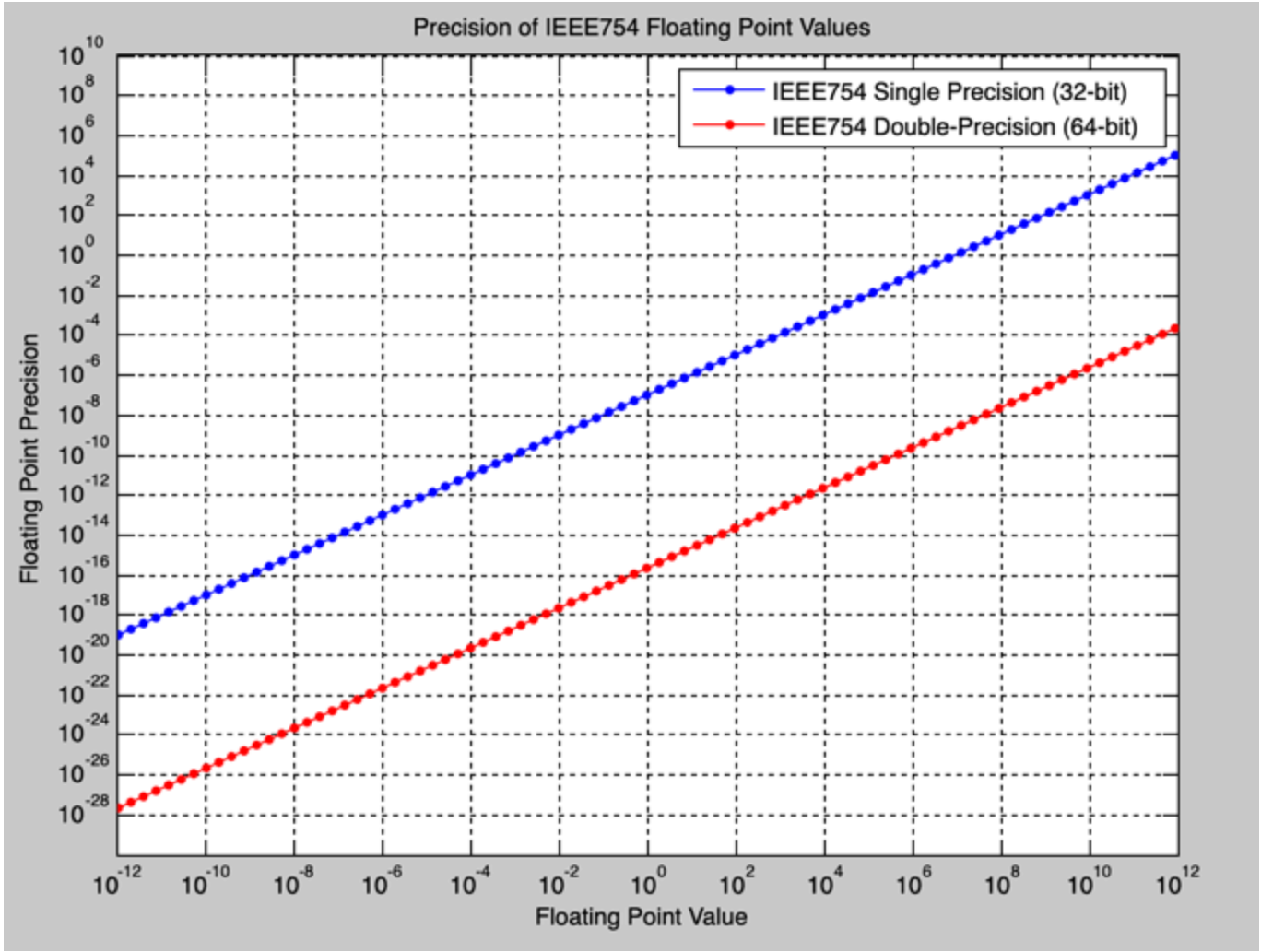
- Ganze Zahlen
- Natürliche Zahlen (Negativ): Das MSB (most significant bit) wird für das Vorzeichen verwendet

Fliesskommazahlen

- Normiert in IEEE 754
- `x = s • m • b^e`
  - Vorzeichen s
  - Mantisse m
  - Basis b (b=2)
  - Exponent e



Floating Point: Präzision



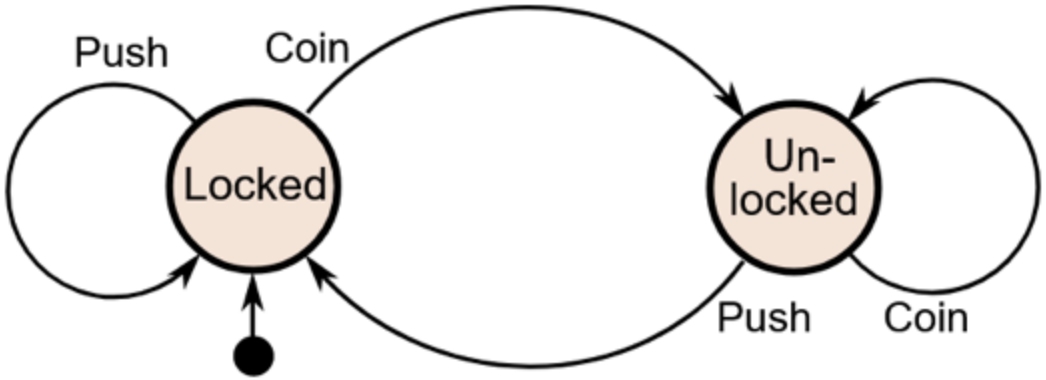
Strings: Unicode

Array von Buchstaben (Char)

	U+002F	/	47	057	Slash (Solidus)	0016
ASCII Digits	U+0030	0	48	060	Digit Zero	0017
	U+0031	1	49	061	Digit One	0018
	U+0032	2	50	062	Digit Two	0019
	U+0033	3	51	063	Digit Three	0020
	U+0034	4	52	064	Digit Four	0021
	U+0035	5	53	065	Digit Five	0022
	U+0036	6	54	066	Digit Six	0023
	U+0037	7	55	067	Digit Seven	0024
	U+0038	8	56	070	Digit Eight	0025
	U+0039	9	57	071	Digit Nine	0026
ASCII Punctuation & Symbols	U+003A	:	58	072	Colon	0027
	U+003B	;	59	073	Semicolon	0028
	U+003C	<	60	074	Less-than sign	0029
	U+003D	=	61	075	Equal sign	0030
	U+003E	>	62	076	Greater-than sign	0031
	U+003F	?	63	077	Question mark	0032
	U+0040	@	64	0100	At sign	0033
	U+0041	A	65	0101	Latin Capital letter A	0034
	U+0042	B	66	0102	Latin Capital letter B	0035
	U+0043	C	67	0103	Latin Capital letter C	0036
	U+0044	D	68	0104	Latin Capital letter D	0037

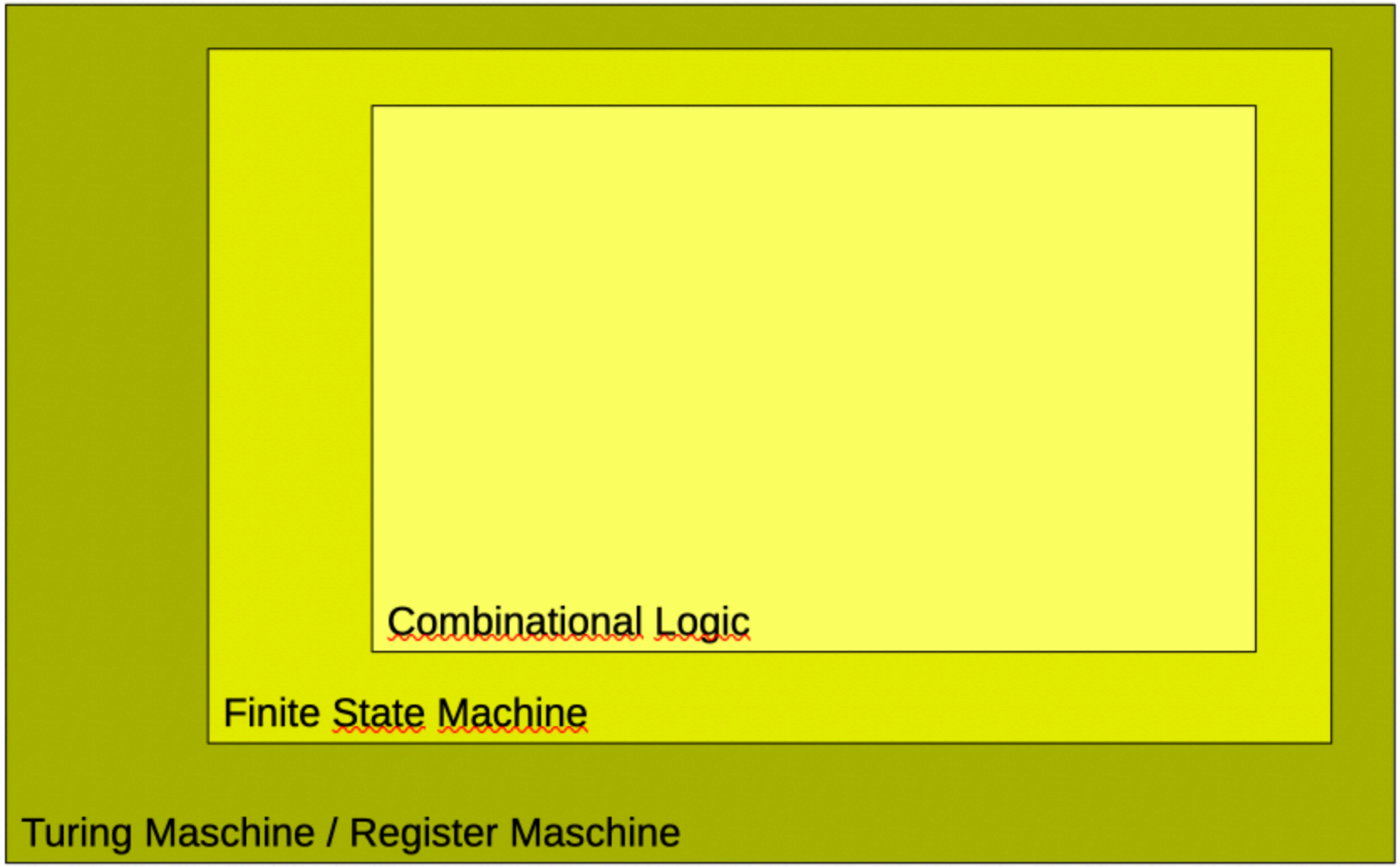
U+1F60x	😀	😁	😂	😃	😄	😅	😆	😇	😈	😉	😊	😋	😌	😍	😎	😏
U+1F61x	😐	😑	😒	😓	😔	😕	😖	😗	😘	😙	😚	😛	😜	😝	😞	😟
U+1F62x	😠	😡	😢	😣	😤	😥	😦	😧	😨	😩	😪	😫	😬	😭	😮	😯
U+1F63x	😰	😱	😲	😳	😴	😵	😶	😷	😸	😹	😺	😻	😼	😽	😾	😿
U+1F64x	🙀	🙁	🙂	🙃	🙄	🙅	🙆	🙇	🙈	🙉	🙊	🙋	🙌	🙍	🙎	🙏
U+1F68x	🚀	🚁	🚂	🚃	🚄	🚅	🚆	🚇	🚈	🚉	🚊	🚋	🚌	🚍	🚎	🚏

Finite State Machine

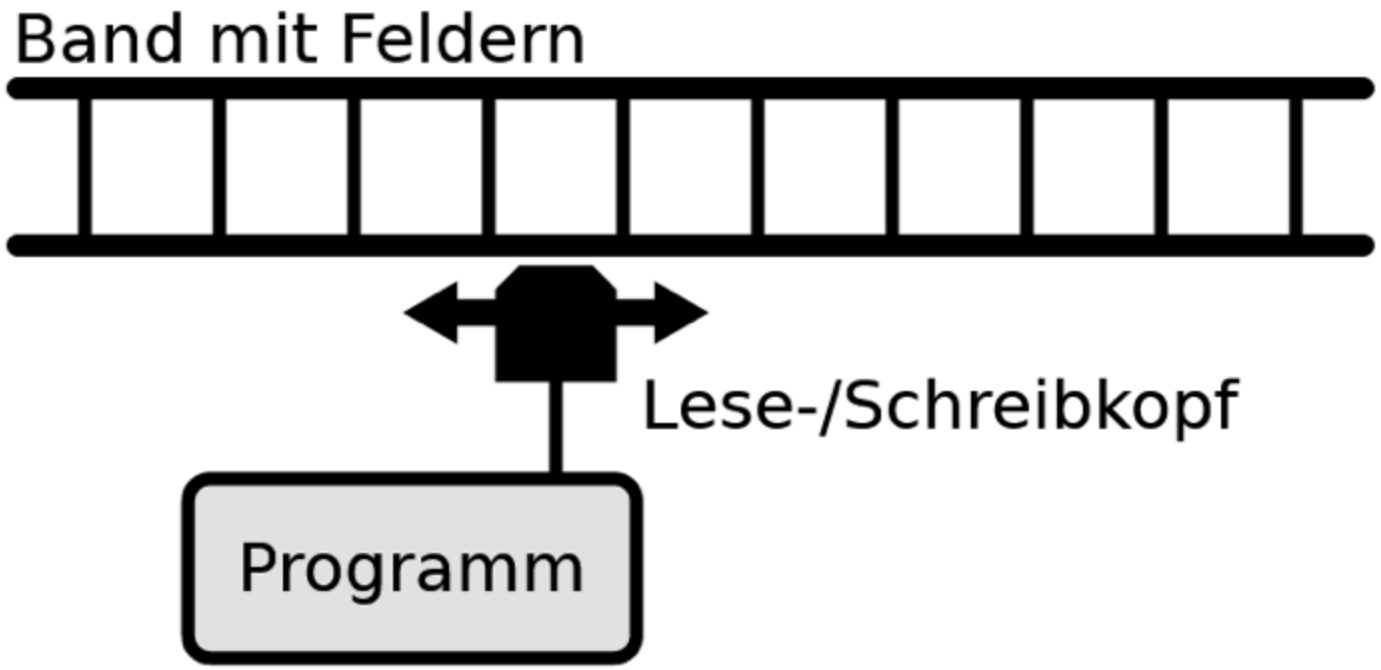


Current State	Input	Next State	Output
Locked	coin	Unlocked	Unlocks the turnstile so that the customer can push through.
	push	Locked	None
Unlocked	coin	Unlocked	None
	push	Locked	When the customer has pushed through, locks the turnstile.

Automatentheorie



Turing-Maschine



Quellen

Silberschatz, 2019
: A.Silberschatz, P.B.Galvin, G. Gagne (2019): Operating System Concepts, Global Edition, Wiley

Patterson, Hennessy, 2014
: D.A.Patterson, J.L.Hennesy (2014): Computer Organization and Design - The Hardware / Software Interface, Fifth
Edition, Morgan Kaufmann