Three Papers on Takums

Laslo Hunhold[1]

[1]Parallel and Distributed Systems Group, University of Cologne, Germany

15th November 2025

Reflections

Reflections

- Computer science is a very young field

Reflections

- Computer science is a very young field
- QUEVEDO invented floating-point numbers in 1913

Reflections

- Computer science is a very young field
- QUEVEDO invented floating-point numbers in 1913
- ZUSE built the first general-purpose computer in 1941 (Z3), merely 85 years ago

Reflections

- Computer science is a very young field
- QUEVEDO invented floating-point numbers in 1913
- ZUSE built the first general-purpose computer in 1941 (Z3), merely 85 years ago

- IEEE 754 is just 40 years old

Reflections

- Computer science is a very young field
- QUEVEDO invented floating-point numbers in 1913
- ZUSE built the first general-purpose computer in 1941 (Z3), merely 85 years ago

- IEEE 754 is just 40 years old
- `bfloat16` is 6 years old, implemented in a lot of hardware

Reflections

- Computer science is a very young field
- QUEVEDO invented floating-point numbers in 1913
- ZUSE built the first general-purpose computer in 1941 (Z3), merely 85 years ago

- IEEE 754 is just 40 years old
- `bfloat16` is 6 years old, implemented in a lot of hardware
- OFP8 is 2 years old, already included in Intel's AVX10.2

Reflections

- Computer science is a very young field
- Quevedo invented floating-point numbers in 1913
- Zuse built the first general-purpose computer in 1941 (Z3), merely 85 years ago

- IEEE 754 is just 40 years old
- `bfloat16` is 6 years old, implemented in a lot of hardware
- OFP8 is 2 years old, already included in Intel's AVX10.2

Fundamental innovations and improvements are not only possible, but the norm.

Demand for new Number Formats

Demand for new Number Formats

Memory Wall

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM
- Halfing the size of a number doubles memory throughput

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM
- Halfing the size of a number doubles memory throughput $\rightarrow$ usually doubles computation speed

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM
- Halfing the size of a number doubles memory throughput $\rightarrow$ usually doubles computation speed
- The age of '`float64` everywhere' is over.

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM
- Halfing the size of a number doubles memory throughput $\rightarrow$ usually doubles computation speed
- The age of '`float64` everywhere' is over.
- IEEE 754 `float32` is okay, but `float16` is useless for general-purpose computations (dynamic range 10^{-8} to 10^{4})

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM
- Halfing the size of a number doubles memory throughput $\rightarrow$ usually doubles computation speed
- The age of '`float64` everywhere' is over.
- IEEE 754 `float32` is okay, but `float16` is useless for general-purpose computations (dynamic range 10^{-8} to 10^{4})

AI Revolution

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM
- Halfing the size of a number doubles memory throughput $\rightarrow$ usually doubles computation speed
- The age of '`float64` everywhere' is over.
- IEEE 754 `float32` is okay, but `float16` is useless for general-purpose computations (dynamic range 10^{-8} to 10^{4})

AI Revolution

- LLMs push the boundaries of computation

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM
- Halfing the size of a number doubles memory throughput $\rightarrow$ usually doubles computation speed
- The age of '`float64` everywhere' is over.
- IEEE 754 `float32` is okay, but `float16` is useless for general-purpose computations (dynamic range 10^{-8} to 10^{4})

AI Revolution

- LLMs push the boundaries of computation
- Heavily overparametrised, the individual weight doesn't matter in the big picture

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM
- Halfing the size of a number doubles memory throughput $\rightarrow$ usually doubles computation speed
- The age of '`float64` everywhere' is over.
- IEEE 754 `float32` is okay, but `float16` is useless for general-purpose computations (dynamic range 10^{-8} to 10^{4})

AI Revolution

- LLMs push the boundaries of computation
- Heavily overparametrised, the individual weight doesn't matter in the big picture
- Super-low precisions per weight (down to five, four, three or even two bits)

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM
- Halfing the size of a number doubles memory throughput $\rightarrow$ usually doubles computation speed
- The age of '`float64` everywhere' is over.
- IEEE 754 `float32` is okay, but `float16` is useless for general-purpose computations (dynamic range 10^{-8} to 10^{4})

AI Revolution

- LLMs push the boundaries of computation
- Heavily overparametrised, the individual weight doesn't matter in the big picture
- Super-low precisions per weight (down to five, four, three or even two bits)
- Main driver of new arithmetics design (AI-tailored arithmetics)

Demand for new Number Formats

Memory Wall

- Computers are not bound by processing speed, but memory bandwidth
- Can perform tens or even hundreds of FLOPs in the time it takes to transmit one single floating-point number from/to RAM
- Halfing the size of a number doubles memory throughput $\rightarrow$ usually doubles computation speed
- The age of '`float64` everywhere' is over.
- IEEE 754 `float32` is okay, but `float16` is useless for general-purpose computations (dynamic range 10^{-8} to 10^{4})

AI Revolution

- LLMs push the boundaries of computation
- Heavily overparametrised, the individual weight doesn't matter in the big picture
- Super-low precisions per weight (down to five, four, three or even two bits)
- Main driver of new arithmetics design (AI-tailored arithmetics)
- Not limited to 8, 16, 32, 64, etc. bits of precision

New Number Formats

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019)

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` $((n_e, n_f) = (8, 7))$

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` $((n_e, n_f) = (8, 7))$
  - Tailored towards AI

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` $((n_e, n_f) = (8, 7))$
  - Tailored towards AI
  - More dynamic range than `float16`

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` $((n_e, n_f) = (8, 7))$
  - Tailored towards AI
  - More dynamic range than `float16`
  - Widespread use in hardware, default format for storing LLM weights

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` $((n_e, n_f) = (8, 7))$
  - Tailored towards AI
  - More dynamic range than `float16`
  - Widespread use in hardware, default format for storing LLM weights
- Open Compute Project (OCP) 8-bit Floating-Point (OFP8)[2] (2023)

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` ($(n_e, n_f) = (8, 7)$)
  - Tailored towards AI
  - More dynamic range than `float16`
  - Widespread use in hardware, default format for storing LLM weights
- Open Compute Project (OCP) 8-bit Floating-Point (OFP8)[2] (2023): Two 8-bit types E4M3 and E5M2 with (n_e, n_f) set at $(4, 3)$ and $(5, 2)$, respectively

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` ($(n_e, n_f) = (8, 7)$)
  - Tailored towards AI
  - More dynamic range than `float16`
  - Widespread use in hardware, default format for storing LLM weights
- Open Compute Project (OCP) 8-bit Floating-Point (OFP8)[2] (2023): Two 8-bit types E4M3 and E5M2 with (n_e, n_f) set at $(4, 3)$ and $(5, 2)$, respectively
  - Tailored towards AI

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` ($(n_e, n_f) = (8, 7)$)
  - Tailored towards AI
  - More dynamic range than `float16`
  - Widespread use in hardware, default format for storing LLM weights
- Open Compute Project (OCP) 8-bit Floating-Point (OFP8)[2] (2023): Two 8-bit types E4M3 and E5M2 with (n_e, n_f) set at $(4, 3)$ and $(5, 2)$, respectively
  - Tailored towards AI
  - E5M2 is a truncated `float16`

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` ($(n_e, n_f) = (8, 7)$)
  - Tailored towards AI
  - More dynamic range than `float16`
  - Widespread use in hardware, default format for storing LLM weights
- Open Compute Project (OCP) 8-bit Floating-Point (OFP8)[2] (2023): Two 8-bit types E4M3 and E5M2 with (n_e, n_f) set at $(4, 3)$ and $(5, 2)$, respectively
  - Tailored towards AI
  - E5M2 is a truncated `float16`
  - Used in Intel's AVX10.2

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` ($(n_e, n_f) = (8, 7)$)
  - Tailored towards AI
  - More dynamic range than `float16`
  - Widespread use in hardware, default format for storing LLM weights
- Open Compute Project (OCP) 8-bit Floating-Point (OFP8)[2] (2023): Two 8-bit types E4M3 and E5M2 with (n_e, n_f) set at $(4, 3)$ and $(5, 2)$, respectively
  - Tailored towards AI
  - E5M2 is a truncated `float16`
  - Used in Intel's AVX10.2
- General trend: Reparametrise IEEE 754, however:

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` ($(n_e, n_f) = (8, 7)$)
  - Tailored towards AI
  - More dynamic range than `float16`
  - Widespread use in hardware, default format for storing LLM weights
- Open Compute Project (OCP) 8-bit Floating-Point (OFP8)[2] (2023): Two 8-bit types E4M3 and E5M2 with (n_e, n_f) set at $(4, 3)$ and $(5, 2)$, respectively
  - Tailored towards AI
  - E5M2 is a truncated `float16`
  - Used in Intel's AVX10.2
- General trend: Reparametrise IEEE 754, however:
  - IEEE 754 has redundant representations, costly at low precisions

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` ($(n_e, n_f) = (8, 7)$)
  - Tailored towards AI
  - More dynamic range than `float16`
  - Widespread use in hardware, default format for storing LLM weights
- Open Compute Project (OCP) 8-bit Floating-Point (OFP8)[2] (2023): Two 8-bit types E4M3 and E5M2 with (n_e, n_f) set at $(4, 3)$ and $(5, 2)$, respectively
  - Tailored towards AI
  - E5M2 is a truncated `float16`
  - Used in Intel's AVX10.2
- General trend: Reparametrise IEEE 754, however:
  - IEEE 754 has redundant representations, costly at low precisions
  - IEEE 754 has a huge hardware overhead (energy, speed)

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

New Number Formats

- All widespread new number formats are based on IEEE 754
- bfloat16[1] (2019): 16-bit format by Google Brain, truncated `float32` ($(n_e, n_f) = (8, 7)$)
  - Tailored towards AI
  - More dynamic range than `float16`
  - Widespread use in hardware, default format for storing LLM weights
- Open Compute Project (OCP) 8-bit Floating-Point (OFP8)[2] (2023): Two 8-bit types E4M3 and E5M2 with (n_e, n_f) set at $(4, 3)$ and $(5, 2)$, respectively
  - Tailored towards AI
  - E5M2 is a truncated `float16`
  - Used in Intel's AVX10.2
- General trend: Reparametrise IEEE 754, however:
  - IEEE 754 has redundant representations, costly at low precisions
  - IEEE 754 has a huge hardware overhead (energy, speed)
  - this creates a zoo of formats, which is complicated for hardware implementers and users

[1] S. Wang and P. Kanwar. 'BFloat16: The secret to high performance on Cloud TPUs'. 2019.
[2] Open Compute Project. '(OCP) 8-bit Floating Point Specification (OFP8)'. 2023

Tapered Precision

Tapered Precision

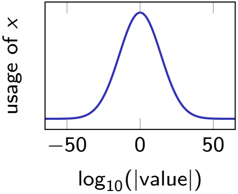
Observation:

Tapered Precision

Observation: Numbers in magnitude close to one are used much more frequently than very large or very small numbers; IEEE 754 (and derived) formats do not reflect this

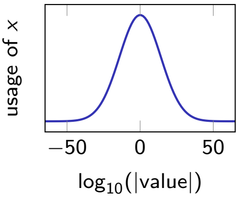
Tapered Precision

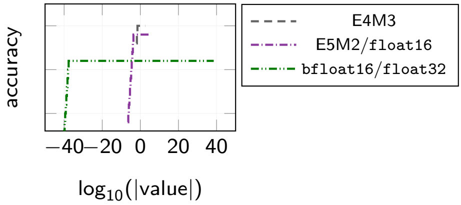
Observation: Numbers in magnitude close to one are used much more frequently than very large or very small numbers; IEEE 754 (and derived) formats do not reflect this



Tapered Precision

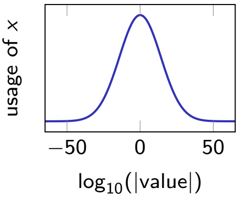
Observation: Numbers in magnitude close to one are used much more frequently than very large or very small numbers; IEEE 754 (and derived) formats do not reflect this

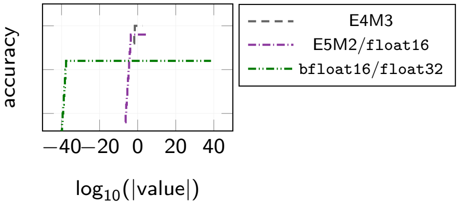




Tapered Precision

Observation: Numbers in magnitude close to one are used much more frequently than very large or very small numbers; IEEE 754 (and derived) formats do not reflect this

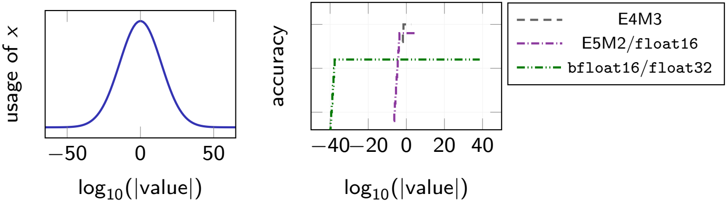




Motivation:

Tapered Precision

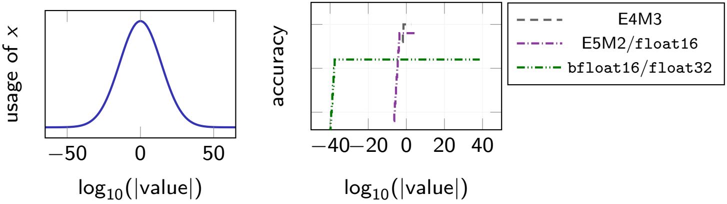
Observation: Numbers in magnitude close to one are used much more frequently than very large or very small numbers; IEEE 754 (and derived) formats do not reflect this



Motivation: Want higher density of numbers (i.e. more fraction bits, precision) for exponent values close to zero

Tapered Precision

Observation: Numbers in magnitude close to one are used much more frequently than very large or very small numbers; IEEE 754 (and derived) formats do not reflect this

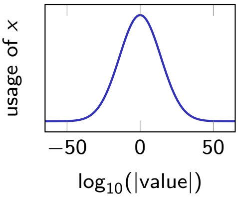


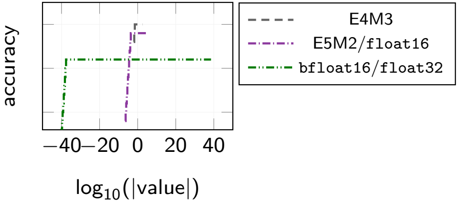
Motivation: Want higher density of numbers (i.e. more fraction bits, precision) for exponent values close to zero

Angle of approach: exponent bits

Tapered Precision

Observation: Numbers in magnitude close to one are used much more frequently than very large or very small numbers; IEEE 754 (and derived) formats do not reflect this





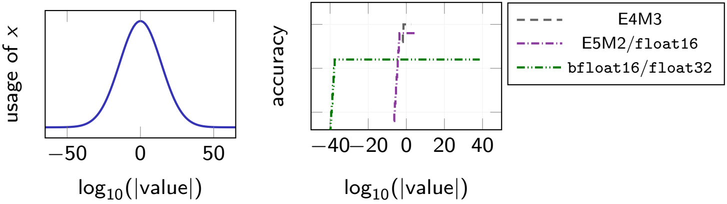
Motivation: Want higher density of numbers (i.e. more fraction bits, precision) for exponent values close to zero

Angle of approach: exponent bits

- Sign and fraction bits already contain maximum information

Tapered Precision

Observation: Numbers in magnitude close to one are used much more frequently than very large or very small numbers; IEEE 754 (and derived) formats do not reflect this



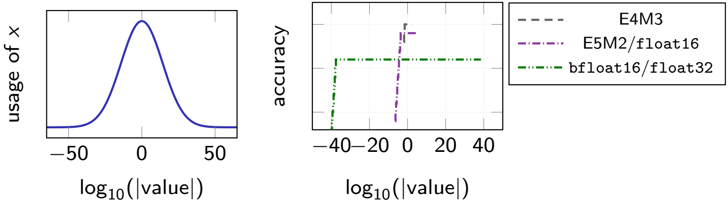
Motivation: Want higher density of numbers (i.e. more fraction bits, precision) for exponent values close to zero

Angle of approach: exponent bits

- Sign and fraction bits already contain maximum information
- Solution: variable-length exponent encoding

Tapered Precision

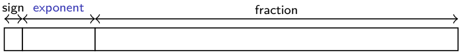
Observation: Numbers in magnitude close to one are used much more frequently than very large or very small numbers; IEEE 754 (and derived) formats do not reflect this



Motivation: Want higher density of numbers (i.e. more fraction bits, precision) for exponent values close to zero

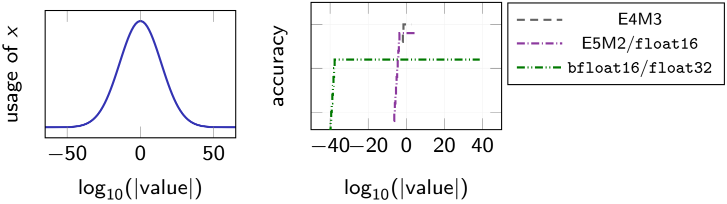
Angle of approach: exponent bits

- Sign and fraction bits already contain maximum information
- Solution: variable-length exponent encoding
    - Small magnitude: shorter exponent, longer fraction, higher density



Tapered Precision

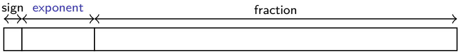
Observation: Numbers in magnitude close to one are used much more frequently than very large or very small numbers; IEEE 754 (and derived) formats do not reflect this



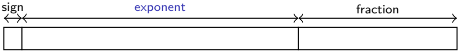
Motivation: Want higher density of numbers (i.e. more fraction bits, precision) for exponent values close to zero

Angle of approach: exponent bits

- Sign and fraction bits already contain maximum information
- Solution: variable-length exponent encoding
  - Small magnitude: shorter exponent, longer fraction, higher density



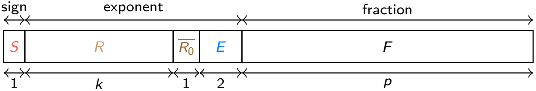
  - Large magnitude: longer exponent, shorter fraction, lower density



Posits

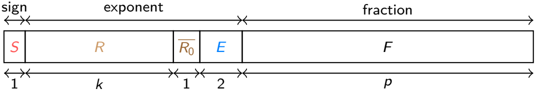
[3] John L. Gustafson et al. ‘Standard for Posit Arithmetic (2022)’. Mar. 2022

Posits



[3] John L. Gustafson et al. ‘Standard for Posit Arithmetic (2022)’. Mar. 2022

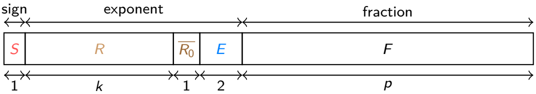
Posits



- State of the art tapered machine number format, in standardisation[3]

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

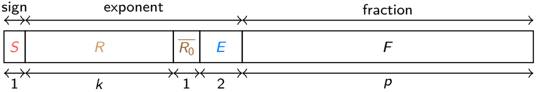
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

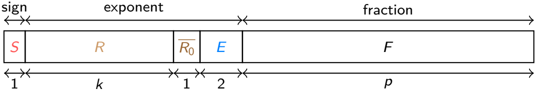
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC ‘TUNGA’ (2024)

[3] John L. Gustafson et al. ‘Standard for Posit Arithmetic (2022)’. Mar. 2022

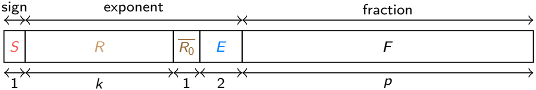
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC 'TUNGA' (2024)
- Exponent coding

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

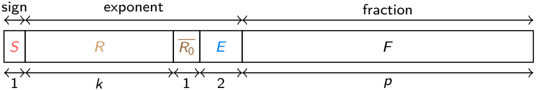
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC 'TUNGA' (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

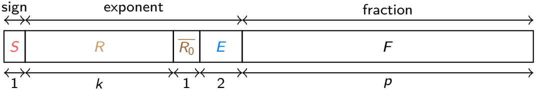
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC 'TUNGA' (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \mathrm{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \mathrm{uint}(E) & \overline{R_0} = 1 \end{cases}$

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

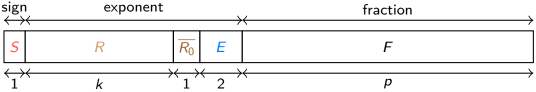
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC 'TUNGA' (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \operatorname{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \operatorname{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

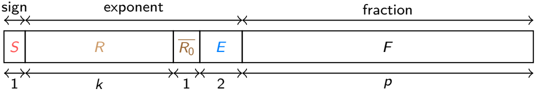
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC 'TUNGA' (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \mathrm{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \mathrm{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)
  - Inefficient for large exponents

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

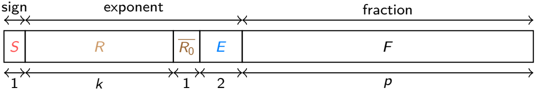
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC ‘TUNGA’ (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \mathrm{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \mathrm{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)
  - Inefficient for large exponents (e.g. 48-bit $111111111111111111111111111111111111111111111011 \equiv 183$)

[3] John L. Gustafson et al. ‘Standard for Posit Arithmetic (2022)’. Mar. 2022

Posits

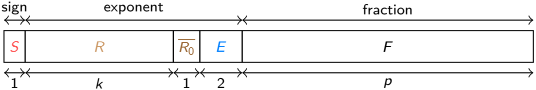


- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC 'TUNGA' (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \mathrm{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \mathrm{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)
  - Inefficient for large exponents (e.g. 48-bit $111111111111111111111111111111111111111111111011 \equiv 183$)
  - R can cover the whole length of the bit string, assume non-specified bits to be zero ('ghost bits')

---

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

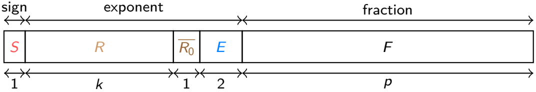
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC 'TUNGA' (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \mathrm{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \mathrm{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)
  - Inefficient for large exponents (e.g. 48-bit $111111111111111111111111111111111111111111111011 \equiv 183$)
  - R can cover the whole length of the bit string, assume non-specified bits to be zero ('ghost bits')
- Properties

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

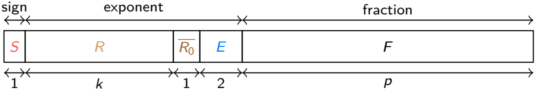
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC ‘TUNGA’ (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \mathrm{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \mathrm{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)
  - Inefficient for large exponents (e.g. 48-bit $11111111111111111111111111111111111111111111011 \equiv 183$)
  - R can cover the whole length of the bit string, assume non-specified bits to be zero (‘ghost bits’)
- Properties
  - No redundant representations

[3] John L. Gustafson et al. ‘Standard for Posit Arithmetic (2022)’. Mar. 2022

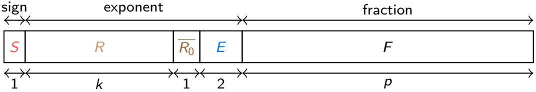
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC ‘TUNGA’ (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \mathrm{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \mathrm{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)
  - Inefficient for large exponents (e.g. 48-bit $111111111111111111111111111111111111111111111011 \equiv 183$)
  - R can cover the whole length of the bit string, assume non-specified bits to be zero (‘ghost bits’)
- Properties
  - No redundant representations
  - Symmetry with two’s complement integers

[3] John L. Gustafson et al. ‘Standard for Posit Arithmetic (2022)’. Mar. 2022

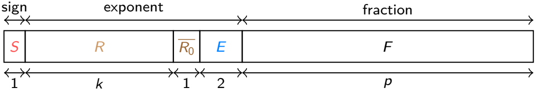
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC ‘TUNGA’ (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \mathrm{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \mathrm{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)
  - Inefficient for large exponents (e.g. 48-bit $1111111111111111111111111111111111111111111111011 \equiv 183$)
  - R can cover the whole length of the bit string, assume non-specified bits to be zero (‘ghost bits’)
- Properties
  - No redundant representations
  - Symmetry with two’s complement integers (unsigned zero of all zero bits

[3] John L. Gustafson et al. ‘Standard for Posit Arithmetic (2022)’. Mar. 2022

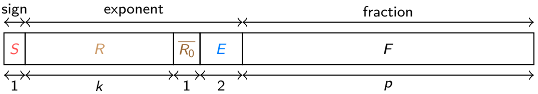
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC 'TUNGA' (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \operatorname{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \operatorname{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)
  - Inefficient for large exponents (e.g. 48-bit $111111111111111111111111111111111111111111111011 \equiv 183$)
  - R can cover the whole length of the bit string, assume non-specified bits to be zero ('ghost bits')
- Properties
  - No redundant representations
  - Symmetry with two's complement integers (unsigned zero of all zero bits, negation

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

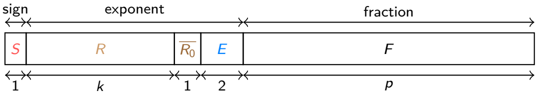
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC ‘TUNGA’ (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \mathrm{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \mathrm{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)
  - Inefficient for large exponents (e.g. 48-bit $1111111111111111111111111111111111111111111111011 \equiv 183$)
  - R can cover the whole length of the bit string, assume non-specified bits to be zero (‘ghost bits’)
- Properties
  - No redundant representations
  - Symmetry with two’s complement integers (unsigned zero of all zero bits, negation, ordering

[3] John L. Gustafson et al. ‘Standard for Posit Arithmetic (2022)’. Mar. 2022

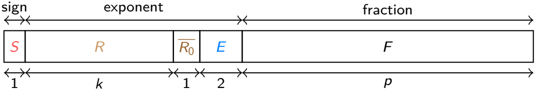
Posits



- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC 'TUNGA' (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \operatorname{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \operatorname{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. $1010 \equiv 2$, $0111 \equiv -1$)
  - Inefficient for large exponents (e.g. 48-bit $111111111111111111111111111111111111111111111011 \equiv 183$)
  - R can cover the whole length of the bit string, assume non-specified bits to be zero ('ghost bits')
- Properties
  - No redundant representations
  - Symmetry with two's complement integers (unsigned zero of all zero bits, negation, ordering, sign)

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

Posits

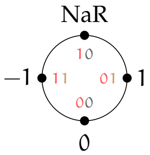


- State of the art tapered machine number format, in standardisation[3]
- Active research field (hundreds of publications since 2017)
- Numerous implementations, latest by Calligo with RISC-V SoC 'TUNGA' (2024)
- Exponent coding
  - R is run of k zeros or ones, followed by one or zero ($\overline{R_0}$) (prefix code)
  - Coded exponent is $e = \begin{cases} -4k + \mathrm{uint}(E) & \overline{R_0} = 0 \\ 4(k-1) + \mathrm{uint}(E) & \overline{R_0} = 1 \end{cases}$
  - Efficient for small exponents (e.g. 1010 $\equiv 2$, 0111 $\equiv -1$)
  - Inefficient for large exponents (e.g. 48-bit 111111111111111111111111111111111111111111111011 $\equiv 183$)
  - R can cover the whole length of the bit string, assume non-specified bits to be zero ('ghost bits')
- Properties
  - No redundant representations
  - Symmetry with two's complement integers (unsigned zero of all zero bits, negation, ordering, sign)
  - Defined for all n, conversion between lengths simple rounding or expansion

[3] John L. Gustafson et al. 'Standard for Posit Arithmetic (2022)'. Mar. 2022

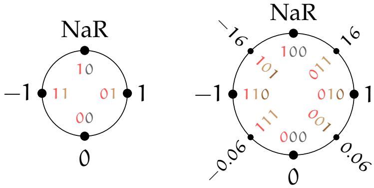
Posit Examples

Posit Examples



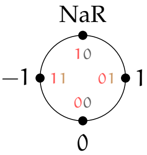
(a) $n = 2$

Posit Examples

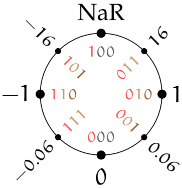


(a) $n = 2$ (b) $n = 3$

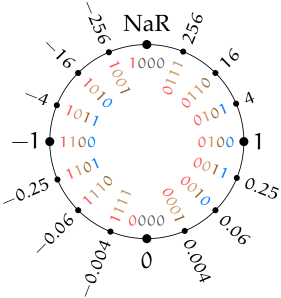
Posit Examples



(a) $n = 2$

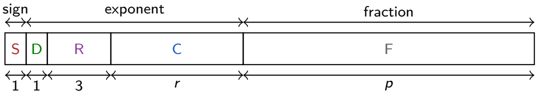


(b) $n = 3$

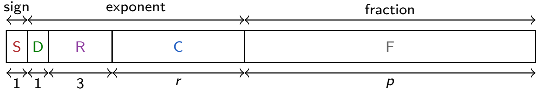


(c) $n = 4$

Takums

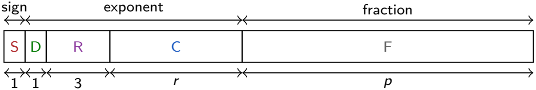


Takums



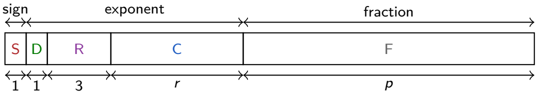
- Goals

Takums



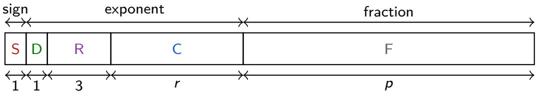
- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)

Takums



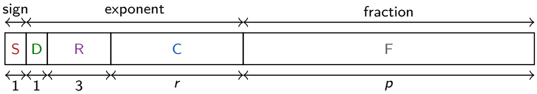
- Goals
  - Preserve useful posit properties (two’s complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic ‘takmarkað umfang’, meaning ‘limited range’)

Takums



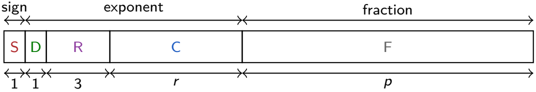
- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic 'takmarkað umfang', meaning 'limited range')
- Exponent coding

Takums



- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic 'takmarkað umfang', meaning 'limited range')
- Exponent coding
  - 3-bit regime (0-7) encodes length, followed by 0 to 7 exponent bits.

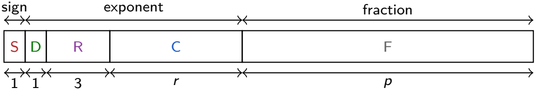
Takums



- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic 'takmarkað umfang', meaning 'limited range')
- Exponent coding
  - 3-bit regime (0-7) encodes length, followed by 0 to 7 exponent bits.
  - Exponent value has implicit leading 1, subtract 1 for range 0-254

value	0	1	2	3	4	...	254
value bits	1	10	11	100	101	...	11111111
encoding	000	0010	0011	01000	01001	...	1111111111

Takums

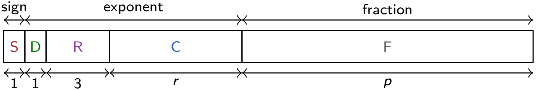


- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic 'takmarkað umfang', meaning 'limited range')
- Exponent coding
  - 3-bit regime (0-7) encodes length, followed by 0 to 7 exponent bits.
  - Exponent value has implicit leading 1, subtract 1 for range 0-254

value	0	1	2	3	4	...	254
value bits	1	10	11	100	101	...	11111111
encoding	000	0010	0011	01000	01001	...	1111111111

  - Separate 'direction' bit D for exponent value sign

Takums

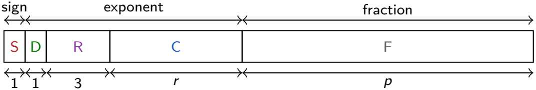


- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic 'takmarkað umfang', meaning 'limited range')
- Exponent coding
  - 3-bit regime (0-7) encodes length, followed by 0 to 7 exponent bits.
  - Exponent value has implicit leading 1, subtract 1 for range 0-254

value	0	1	2	3	4	...	254
value bits	1	10	11	100	101	...	11111111
encoding	000	0010	0011	01000	01001	...	1111111111

  - Separate 'direction' bit D for exponent value sign
- Improvements over posits

Takums

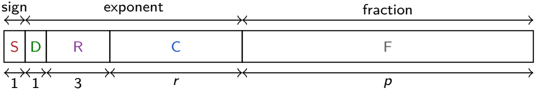


- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic 'takmarkað umfang', meaning 'limited range')
- Exponent coding
  - 3-bit regime (0-7) encodes length, followed by 0 to 7 exponent bits.
  - Exponent value has implicit leading 1, subtract 1 for range 0-254

value	0	1	2	3	4	...	254
value bits	1	10	11	100	101	...	11111111
encoding	000	0010	0011	01000	01001	...	1111111111

  - Separate 'direction' bit D for exponent value sign
- Improvements over posits
  - Exponent is at most 11 bits long ($\rightarrow$ simpler hardware)

Takums

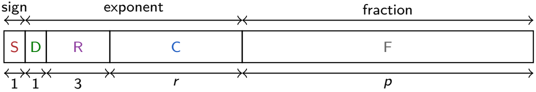


- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic 'takmarkað umfang', meaning 'limited range')
- Exponent coding
  - 3-bit regime (0-7) encodes length, followed by 0 to 7 exponent bits.
  - Exponent value has implicit leading 1, subtract 1 for range 0-254

value	0	1	2	3	4	...	254
value bits	1	10	11	100	101	...	11111111
encoding	000	0010	0011	01000	01001	...	1111111111

  - Separate 'direction' bit D for exponent value sign
- Improvements over posits
  - Exponent is at most 11 bits long ($\rightarrow$ simpler hardware)
  - Machine precision bounded, same as `float64`, permits numerical analysis

Takums

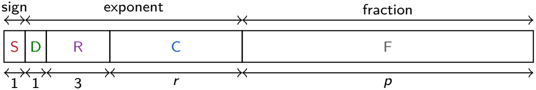


- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic 'takmarkað umfang', meaning 'limited range')
- Exponent coding
  - 3-bit regime (0-7) encodes length, followed by 0 to 7 exponent bits.
  - Exponent value has implicit leading 1, subtract 1 for range 0-254

value	0	1	2	3	4	...	254
value bits	1	10	11	100	101	...	11111111
encoding	000	0010	0011	01000	01001	...	1111111111

  - Separate 'direction' bit D for exponent value sign
- Improvements over posits
  - Exponent is at most 11 bits long ($\rightarrow$ simpler hardware)
  - Machine precision bounded, same as `float64`, permits numerical analysis
  - Interlude: Comparison for value 254 (posit $\rightarrow$ takum):

Takums



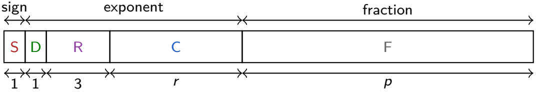
- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic 'takmarkað umfang', meaning 'limited range')
- Exponent coding
  - 3-bit regime (0-7) encodes length, followed by 0 to 7 exponent bits.
  - Exponent value has implicit leading 1, subtract 1 for range 0-254

value	0	1	2	3	4	...	254
value bits	1	10	11	100	101	...	11111111
encoding	000	0010	0011	01000	01001	...	1111111111

  - Separate 'direction' bit D for exponent value sign
- Improvements over posits
  - Exponent is at most 11 bits long ($\rightarrow$ simpler hardware)
  - Machine precision bounded, same as `float64`, permits numerical analysis
  - Interlude: Comparison for value 254 (posit $\rightarrow$ takum):

1111111111111111111111111111111111111111111111111111111111111111010 (67b)

Takums



- Goals
  - Preserve useful posit properties (two's complement symmetry, etc.)
  - Bounded, high dynamic range, attained already at low precisions (Icelandic 'takmarkað umfang', meaning 'limited range')
- Exponent coding
  - 3-bit regime (0-7) encodes length, followed by 0 to 7 exponent bits.
  - Exponent value has implicit leading 1, subtract 1 for range 0-254

value	0	1	2	3	4	...	254
value bits	1	10	11	100	101	...	11111111
encoding	000	0010	0011	01000	01001	...	1111111111

  - Separate 'direction' bit D for exponent value sign
- Improvements over posits
  - Exponent is at most 11 bits long ($\rightarrow$ simpler hardware)
  - Machine precision bounded, same as `float64`, permits numerical analysis
  - Interlude: Comparison for value 254 (posit $\rightarrow$ takum):

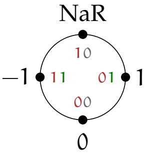
1111111111111111111111111111111111111111111111111111111111111111010 (67b)

$\downarrow$

11111111111 (11b)

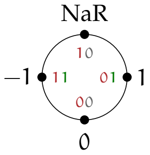
Takum Examples

Takum Examples

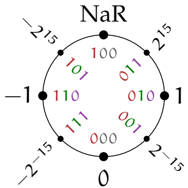


(a) $n = 2$

Takum Examples

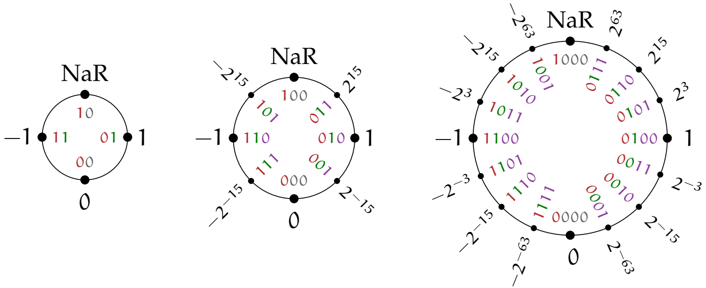


(a) $n = 2$



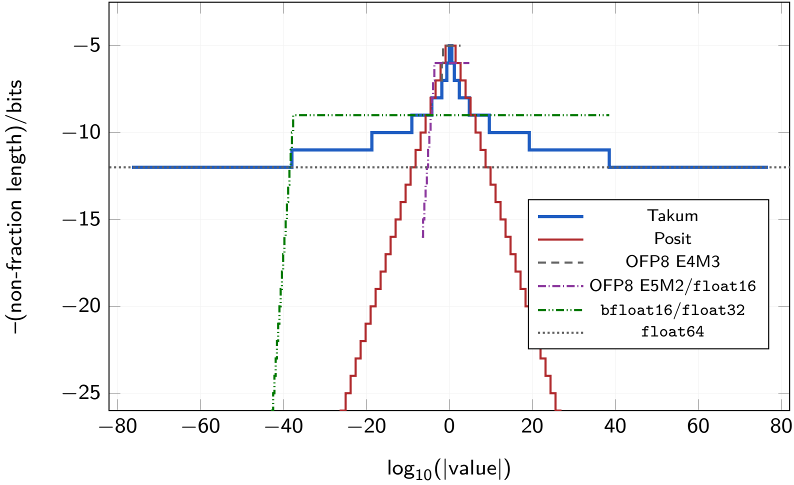
(b) $n = 3$

Takum Examples



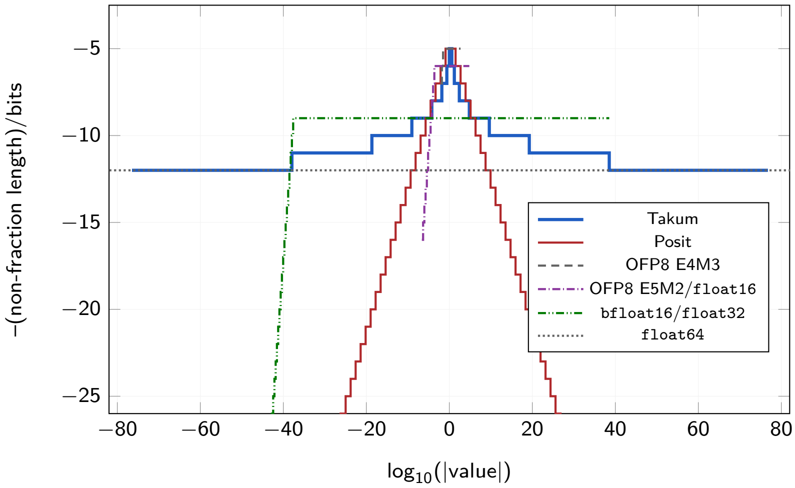
(a) $n = 2$ (b) $n = 3$ (c) $n = 4$

Tapering



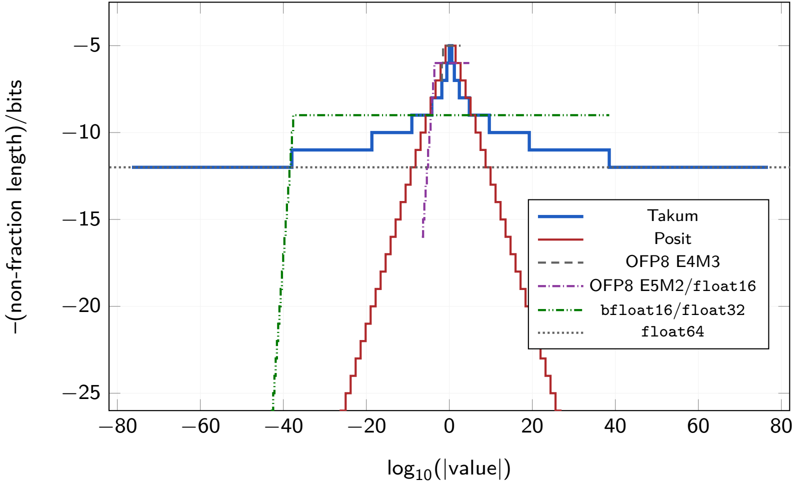
−5
−10
−15
−20
−25
−(non-fraction length)/bits
−80
−60
−40
−20
0
20
40
60
80
log10(|value|)
Takum
Posit
OFP8 E4M3
OFP8 E5M2/float16
bfloat16/float32
float64

Tapering



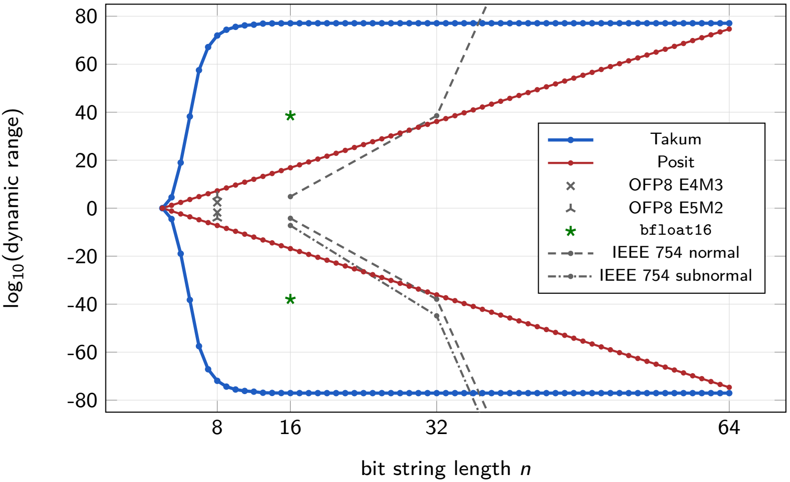
Posit: Linear tapering, drop-off at the edges

Tapering

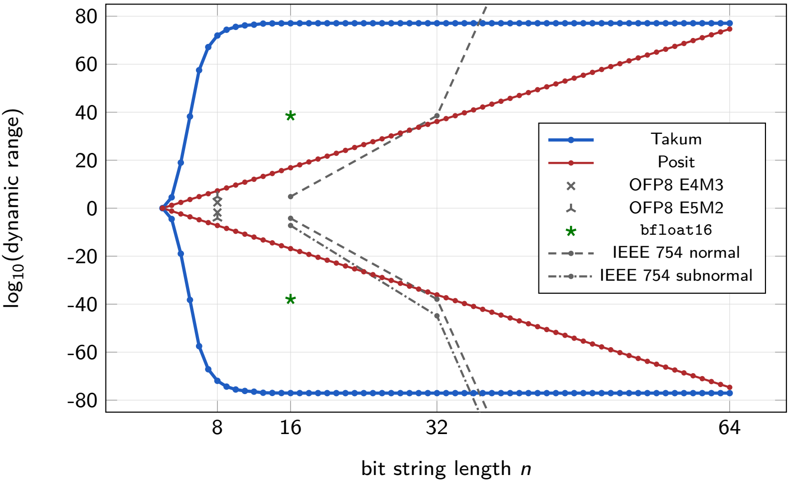


Posit: Linear tapering, drop-off at the edges
Takum: Logarithmic tapering, no drop-off at the edges

Dynamic Range

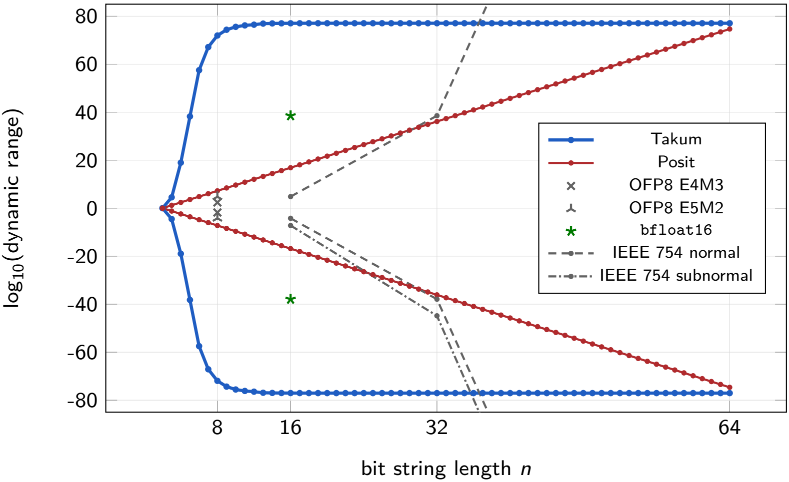


Dynamic Range



Takum: Reducing precision only reduces accuracy, not dynamic range;

Dynamic Range



Takum: Reducing precision only reduces accuracy, not dynamic range;
dynamic range is general-purpose, fully available at 8 bits

First Paper

Integer Representations in IEEE 754, OFP8, Bfloat16, Posit, and Takum Arithmetics

Integer Representations

Motivation and Approach

Integer Representations

Motivation and Approach

- Floats are often used to represent integers (e.g. JavaScript, file formats, implicit use)

Integer Representations

Motivation and Approach

- Floats are often used to represent integers (e.g. JavaScript, file formats, implicit use)
- Like using a wrench as a hammer

Integer Representations

Motivation and Approach

- Floats are often used to represent integers (e.g. JavaScript, file formats, implicit use)
- Like using a wrench as a hammer
- Quantity of interest: Largest consecutive integer (no rounding gaps)

Integer Representations

Motivation and Approach

- Floats are often used to represent integers (e.g. JavaScript, file formats, implicit use)
- Like using a wrench as a hammer
- Quantity of interest: Largest consecutive integer (no rounding gaps)
- Empirically derived formula in posits standard (`pIntMax`)

Integer Representations

Motivation and Approach

- Floats are often used to represent integers (e.g. JavaScript, file formats, implicit use)
- Like using a wrench as a hammer
- Quantity of interest: Largest consecutive integer (no rounding gaps)
- Empirically derived formula in posits standard (`pIntMax`)
- No prior works

Integer Representations

Motivation and Approach

- Floats are often used to represent integers (e.g. JavaScript, file formats, implicit use)
- Like using a wrench as a hammer
- Quantity of interest: Largest consecutive integer (no rounding gaps)
- Empirically derived formula in posits standard (`pIntMax`)
- No prior works
- Plan: Examine for IEEE 754, posits and takums and compare formats

Integer Representations

Motivation and Approach

- Floats are often used to represent integers (e.g. JavaScript, file formats, implicit use)
- Like using a wrench as a hammer
- Quantity of interest: Largest consecutive integer (no rounding gaps)
- Empirically derived formula in posits standard (`pIntMax`)
- No prior works
- Plan: Examine for IEEE 754, posits and takums and compare formats
- Are posits, takums drop-in solutions for IEEE 754 in this regard?

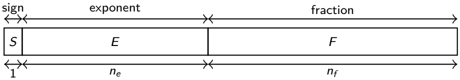
Integer Representations

Motivation and Approach

- Floats are often used to represent integers (e.g. JavaScript, file formats, implicit use)
- Like using a wrench as a hammer
- Quantity of interest: Largest consecutive integer (no rounding gaps)
- Empirically derived formula in posits standard (`pIntMax`)
- No prior works
- Plan: Examine for IEEE 754, posits and takums and compare formats
- Are posits, takums drop-in solutions for IEEE 754 in this regard?
- Different question: Given an integer, how many bits do I need to represent it as a posit or takum?

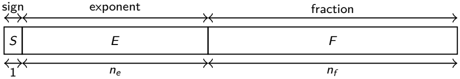
Integer Representations

IEEE 754 Floating-Point Numbers



Integer Representations

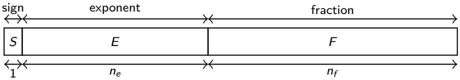
IEEE 754 Floating-Point Numbers



- Assume IEEE 754 floating-point number with n_e exponent and n_f fraction bits

Integer Representations

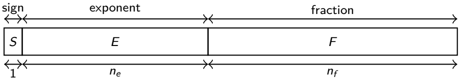
IEEE 754 Floating-Point Numbers



- Assume IEEE 754 floating-point number with n_e exponent and n_f fraction bits
- Condition: n_e sufficiently large such that the exponent can be $n_f + 1$

Integer Representations

IEEE 754 Floating-Point Numbers



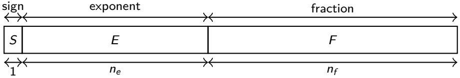
- Assume IEEE 754 floating-point number with n_e exponent and n_f fraction bits
- Condition: n_e sufficiently large such that the exponent can be $n_f + 1$
- All integers $m \in \mathbb{Z}$ with

$$|m| \leq 2^{n_f+1}$$

  are representable

Integer Representations

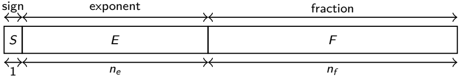
IEEE 754 Floating-Point Numbers



- Assume IEEE 754 floating-point number with n_e exponent and n_f fraction bits
- Condition: n_e sufficiently large such that the exponent can be $n_f + 1$
- All integers $m \in \mathbb{Z}$ with
$$|m| \leq 2^{n_f+1}$$
are representable
- Proof: Extremely simple
  - We can store $n_f + 1$ fraction bits (one implicit), so all integers up to $2^{n_f+1} - 1$

Integer Representations

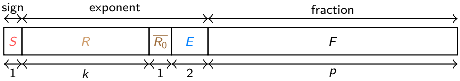
IEEE 754 Floating-Point Numbers



- Assume IEEE 754 floating-point number with n_e exponent and n_f fraction bits
- Condition: n_e sufficiently large such that the exponent can be $n_f + 1$
- All integers $m \in \mathbb{Z}$ with
$$|m| \leq 2^{n_f+1}$$
  are representable
- Proof: Extremely simple
  - We can store $n_f + 1$ fraction bits (one implicit), so all integers up to $2^{n_f+1} - 1$
  - We can also store 2^{n_f+1}, its successor $\square$

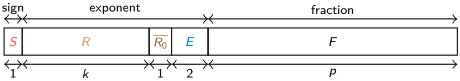
Integer Representations

Posits



Integer Representations

Posits



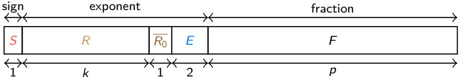
- Let $m \in \mathbb{Z} \setminus \{0\}$ with $v := 1 + \lfloor \log_2(|m|) \rfloor$ bits and $w := \max_{i \in \mathbb{N}_0} \left(2^i \mid m\right)$ trailing zeros in $|m|$'s binary representation. There exists an $M \in \{0,1\}^\ell$ with $\pi(M) = m$ and

$$\ell := \left\lfloor \frac{5(v+3)}{4} - w \right\rfloor - (w = v - 1) \cdot \begin{cases} 3 & v \in 4\mathbb{N}_0 + 1 \\ 1 & v \in 4\mathbb{N}_0 + 3, \end{cases}$$

which is the shortest possible representation.

Integer Representations

Posits



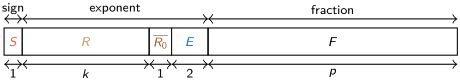
- Let $m \in \mathbb{Z} \setminus \{0\}$ with $v := 1 + \lfloor \log_2(|m|) \rfloor$ bits and $w := \max_{i \in \mathbb{N}_0} \left(2^i \mid m\right)$ trailing zeros in $|m|$'s binary representation. There exists an $M \in \{0,1\}^{\ell}$ with $\pi(M) = m$ and

$$\ell := \left\lfloor \frac{5(v+3)}{4} - w \right\rfloor - (w = v - 1) \cdot \begin{cases} 3 & v \in 4\mathbb{N}_0 + 1 \\ 1 & v \in 4\mathbb{N}_0 + 3, \end{cases}$$

  which is the shortest possible representation.
- Proof: Encode integer as posit, check reducibility (trailing zeros). $\square$

Integer Representations

Posits



- Let $m \in \mathbb{Z} \setminus \{0\}$ with $v := 1 + \lfloor \log_2(|m|) \rfloor$ bits and $w := \max_{i \in \mathbb{N}_0} \left(2^i \mid m \right)$ trailing zeros in $|m|$'s binary representation. There exists an $M \in \{0,1\}^\ell$ with $\pi(M) = m$ and

$$\ell := \left\lfloor \frac{5(v+3)}{4} - w \right\rfloor - (w = v - 1) \cdot \begin{cases} 3 & v \in 4\mathbb{N}_0 + 1 \\ 1 & v \in 4\mathbb{N}_0 + 3, \end{cases}$$

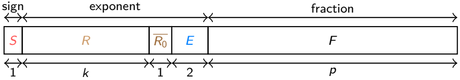
which is the shortest possible representation.

- Proof: Encode integer as posit, check reducibility (trailing zeros). □
- Let $n \in \mathbb{N}_3$. It holds for all $m \in \mathbb{Z}$ with

$$|m| \leq 2^{\left\lfloor \frac{4(n-3)}{5} \right\rfloor}$$

that there exists an $M \in \{0,1\}^n$ with $m = \pi(M)$.

Integer Representations
Posits



- Let $m \in \mathbb{Z} \setminus \{0\}$ with $v := 1 + \lfloor \log_2(|m|) \rfloor$ bits and $w := \max_{i \in \mathbb{N}_0} \left(2^i \mid m\right)$ trailing zeros in $|m|$'s binary representation. There exists an $M \in \{0,1\}^\ell$ with $\pi(M) = m$ and

$$\ell := \left\lfloor \frac{5(v+3)}{4} - w \right\rfloor - (w = v - 1) \cdot \begin{cases} 3 & v \in 4\mathbb{N}_0 + 1 \\ 1 & v \in 4\mathbb{N}_0 + 3, \end{cases}$$

which is the shortest possible representation.

- Proof: Encode integer as posit, check reducibility (trailing zeros). □

- Let $n \in \mathbb{N}_3$. It holds for all $m \in \mathbb{Z}$ with

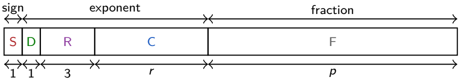
$$|m| \leq 2^{\left\lfloor \frac{4(n-3)}{5} \right\rfloor}$$

that there exists an $M \in \{0,1\}^n$ with $m = \pi(M)$.

- Proof: Insert ℓ into $\ell \leq n$, rearrange for v, consider saturated $m = 2^v - 1$ and check successors $m + 1$ (fits) and $m + 2$ (doesn't fit). □

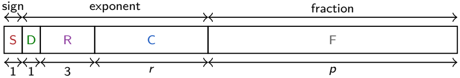
Integer Representations

Takums



Integer Representations

Takums



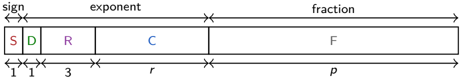
- Let $m \in \mathbb{Z} \setminus \{0\}$ with $|m| \leq 2^{254}$, $v := 1 + \lfloor \log_2(|m|) \rfloor$ bits and $w := \max_{i \in \mathbb{N}_0} \left(2^i \mid m\right)$ trailing zeros in $|m|$'s binary representation. There exists an $M \in \{0,1\}^\ell$ with $\tau(M) = m$ and

$$\begin{aligned}\ell := \lfloor 4 + v + \log_2(v) - w \rfloor - &\\ &(w = v - 1) \cdot \max_{i \in \{0,\dots,\lfloor \log_2(v) \rfloor\}} \left(2^i \mid v - 2^{\lfloor \log_2(v) \rfloor}\right) - \\ &(v \in 2^{\mathbb{N}_0}) \cdot \max_{i \in \{0,\dots,3\}} \left(2^i \mid \lfloor \log_2(v) \rfloor\right),\end{aligned}$$

which is the shortest possible representation.

Integer Representations

Takums

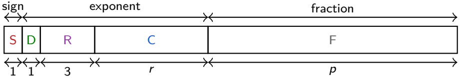


- Let $m \in \mathbb{Z} \setminus \{0\}$ with $|m| \leq 2^{254}$, $v := 1 + \lfloor \log_2(|m|) \rfloor$ bits and $w := \max_{i \in \mathbb{N}_0} \left(2^i \mid m\right)$ trailing zeros in $|m|$'s binary representation. There exists an $M \in \{0,1\}^{\ell}$ with $\tau(M) = m$ and

$$\begin{aligned}\ell := \lfloor 4 + v + \log_2(v) - w \rfloor - \\ &(w = v - 1) \cdot \max_{i \in \{0,\dots,\lfloor \log_2(v) \rfloor\}} \left(2^i \mid v - 2^{\lfloor \log_2(v) \rfloor}\right) - \\ &(v \in 2^{\mathbb{N}_0}) \cdot \max_{i \in \{0,\dots,3\}} \left(2^i \mid \lfloor \log_2(v) \rfloor\right),\end{aligned}$$

  which is the shortest possible representation.
- Proof: Just like for posits, only harder. $\square$

Integer Representations
Takums



- Let $m \in \mathbb{Z} \setminus \{0\}$ with $|m| \leq 2^{254}$, $v := 1 + \lfloor \log_2(|m|) \rfloor$ bits and $w := \max_{i \in \mathbb{N}_0} \left(2^i \mid m\right)$ trailing zeros in $|m|$'s binary representation. There exists an $M \in \{0,1\}^\ell$ with $\tau(M) = m$ and

$$\begin{aligned}\ell := &\lfloor 4 + v + \log_2(v) - w \rfloor - \\ &(w = v - 1) \cdot \max_{i \in \{0,\dots,\lfloor \log_2(v) \rfloor\}} \left(2^i \mid v - 2^{\lfloor \log_2(v) \rfloor}\right) - \\ &(v \in 2^{\mathbb{N}_0}) \cdot \max_{i \in \{0,\dots,3\}} \left(2^i \mid \lfloor \log_2(v) \rfloor\right),\end{aligned}$$

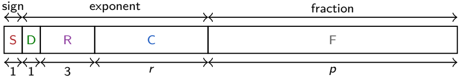
  which is the shortest possible representation.
- Proof: Just like for posits, only harder. $\square$
- Let $n \in \mathbb{N}_5$. It holds for all $m \in \mathbb{Z}$ with $|m| \leq 2^{254}$ and

$$|m| \leq 2^{\left\lceil \frac{W_0\left(2^{n-3}\ln(2)\right)}{\ln(2)} - 1 \right\rceil},$$

  where W_0 is the principal branch of the Lambert W function, that there exists an $M \in \{0,1\}^n$ with $m = \tau(M)$.

Integer Representations

Takums



- Let $m \in \mathbb{Z} \setminus \{0\}$ with $|m| \leq 2^{254}$, $v := 1 + \lfloor \log_2(|m|) \rfloor$ bits and $w := \max_{i \in \mathbb{N}_0} \left(2^i \mid m\right)$ trailing zeros in $|m|$'s binary representation. There exists an $M \in \{0,1\}^\ell$ with $\tau(M) = m$ and

$$\begin{aligned}\ell := \lfloor 4 + v + \log_2(v) - w \rfloor - \\ & (w = v - 1) \cdot \max_{i \in \{0,\dots,\lfloor \log_2(v) \rfloor\}} \left(2^i \mid v - 2^{\lfloor \log_2(v) \rfloor}\right) - \\ & (v \in 2^{\mathbb{N}_0}) \cdot \max_{i \in \{0,\dots,3\}} \left(2^i \mid \lfloor \log_2(v) \rfloor\right),\end{aligned}$$

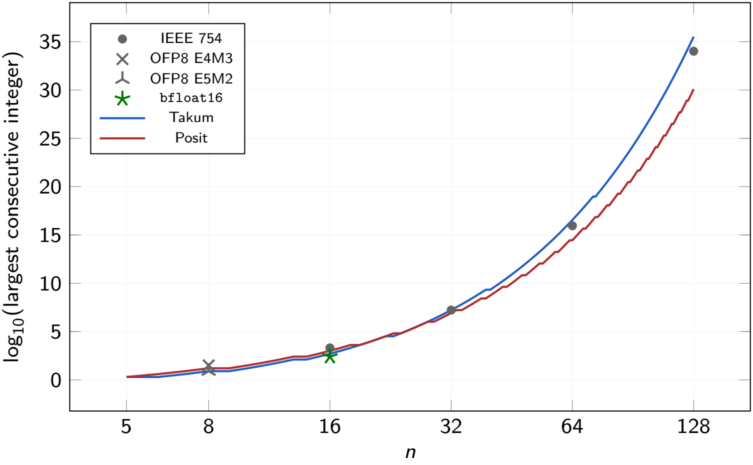
  which is the shortest possible representation.
- Proof: Just like for posits, only harder. $\square$
- Let $n \in \mathbb{N}_5$. It holds for all $m \in \mathbb{Z}$ with $|m| \leq 2^{254}$ and

$$|m| \leq 2^{\left\lceil \frac{W_0\left(2^{n-3}\ln(2)\right)}{\ln(2)} - 1 \right\rceil},$$

  where W_0 is the principal branch of the Lambert W function, that there exists an $M \in \{0,1\}^n$ with $m = \tau(M)$.
- Proof: Just like for posits, only harder. $\square$

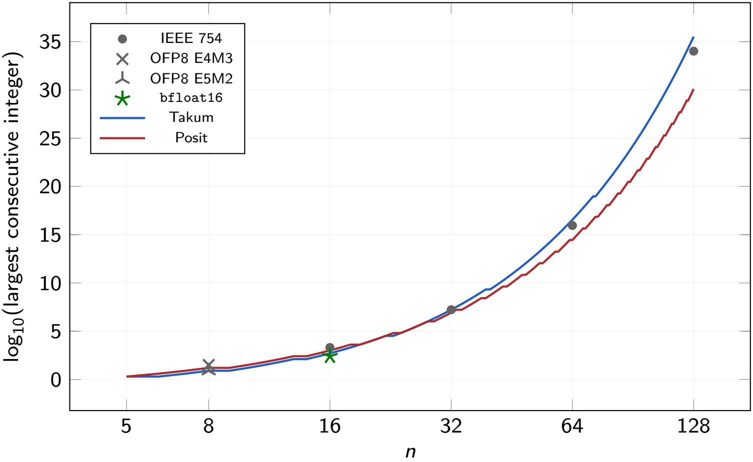
Integer Representations

Comparison



Integer Representations

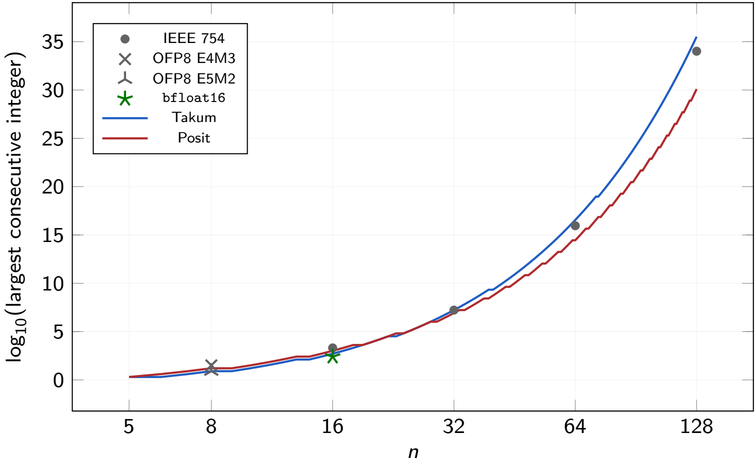
Comparison



Posits slightly better than takums for $n < 20$

Integer Representations

Comparison



Posits slightly better than takums for $n < 20$

Posits scale worse, takums scale better than IEEE 754 (both outperform `bfloat16`)

Integer Representations

Conclusion and Outlook

Integer Representations

Conclusion and Outlook

- Posits standard is now formally verified in this regard

Integer Representations

Conclusion and Outlook

- Posits standard is now formally verified in this regard
- Formal approach necessary for takums; empirical derivation would have been impossible

Integer Representations

Conclusion and Outlook

- Posits standard is now formally verified in this regard
- Formal approach necessary for takums; empirical derivation would have been impossible
- Tapered precision formats require new proof techniques; challenge for tapered-precision numerical analysis

Integer Representations

Conclusion and Outlook

- Posits standard is now formally verified in this regard
- Formal approach necessary for takums; empirical derivation would have been impossible
- Tapered precision formats require new proof techniques; challenge for tapered-precision numerical analysis
- Outlook: B-Posits

Second Paper

Design and Implementation of a Takum Arithmetic Hardware Codec

Takum FPGA Codec

Overview and Decoder Design

Takum FPGA Codec

Overview and Decoder Design

- Current state of posit APUs: Decode to IR, compute with IR, encode from IR

Takum FPGA Codec

Overview and Decoder Design

- Current state of posit APUs: Decode to IR, compute with IR, encode from IR
- Posits and takums share the same IR

Takum FPGA Codec

Overview and Decoder Design

- Current state of posit APUs: Decode to IR, compute with IR, encode from IR
- Posits and takums share the same IR
- Contribution on the side: More efficient IR for logarithmic takums/posits (*barred logarithmic value*)

Takum FPGA Codec

Overview and Decoder Design

- Current state of posit APUs: Decode to IR, compute with IR, encode from IR
- Posits and takums share the same IR
- Contribution on the side: More efficient IR for logarithmic takums/posits (*barred logarithmic value*)
- Codec is main differentiator in performance of posits vs. takums in hardware

Takum FPGA Codec

Overview and Decoder Design

- Current state of posit APUs: Decode to IR, compute with IR, encode from IR
- Posits and takums share the same IR
- Contribution on the side: More efficient IR for logarithmic takums/posits (*barred logarithmic value*)
- Codec is main differentiator in performance of posits vs. takums in hardware
- Requirement: Fully formalised, VHDL-generic, with test benches

Takum FPGA Codec

Overview and Decoder Design

- Current state of posit APUs: Decode to IR, compute with IR, encode from IR
- Posits and takums share the same IR
- Contribution on the side: More efficient IR for logarithmic takums/posits (*barred logarithmic value*)
- Codec is main differentiator in performance of posits vs. takums in hardware
- Requirement: Fully formalised, VHDL-generic, with test benches

Decoder

- If $n < 12$, expand to 12 bits, extract the first 12 bits

Takum FPGA Codec

Overview and Decoder Design

- Current state of posit APUs: Decode to IR, compute with IR, encode from IR
- Posits and takums share the same IR
- Contribution on the side: More efficient IR for logarithmic takums/posits (*barred logarithmic value*)
- Codec is main differentiator in performance of posits vs. takums in hardware
- Requirement: Fully formalised, VHDL-generic, with test benches

Decoder

- If $n < 12$, expand to 12 bits, extract the first 12 bits
- Sign, regime, characteristic bits are at fixed offsets

Takum FPGA Codec

Overview and Decoder Design

- Current state of posit APUs: Decode to IR, compute with IR, encode from IR
- Posits and takums share the same IR
- Contribution on the side: More efficient IR for logarithmic takums/posits (*barred logarithmic value*)
- Codec is main differentiator in performance of posits vs. takums in hardware
- Requirement: Fully formalised, VHDL-generic, with test benches

Decoder

- If $n < 12$, expand to 12 bits, extract the first 12 bits
- Sign, regime, characteristic bits are at fixed offsets
- Highly efficient computation of characteristic value

Takum FPGA Codec

Overview and Decoder Design

- Current state of posit APUs: Decode to IR, compute with IR, encode from IR
- Posits and takums share the same IR
- Contribution on the side: More efficient IR for logarithmic takums/posits (*barred logarithmic value*)
- Codec is main differentiator in performance of posits vs. takums in hardware
- Requirement: Fully formalised, VHDL-generic, with test benches

Decoder

- If $n < 12$, expand to 12 bits, extract the first 12 bits
- Sign, regime, characteristic bits are at fixed offsets
- Highly efficient computation of characteristic value
- Fraction obtained with simple shift by regime value (between 0 and 7; 3 bit shifter input independent of n)

Takum FPGA Codec

Overview and Decoder Design

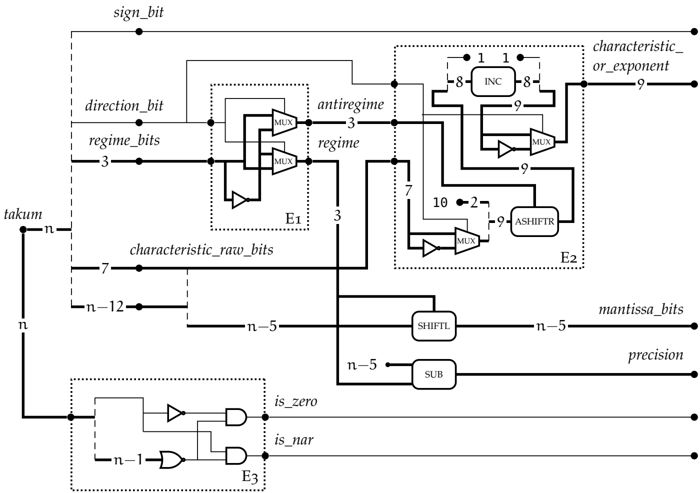
- Current state of posit APUs: Decode to IR, compute with IR, encode from IR
- Posits and takums share the same IR
- Contribution on the side: More efficient IR for logarithmic takums/posits (*barred logarithmic value*)
- Codec is main differentiator in performance of posits vs. takums in hardware
- Requirement: Fully formalised, VHDL-generic, with test benches

Decoder

- If $n < 12$, expand to 12 bits, extract the first 12 bits
- Sign, regime, characteristic bits are at fixed offsets
- Highly efficient computation of characteristic value
- Fraction obtained with simple shift by regime value (between 0 and 7; 3 bit shifter input independent of n)
- Posits: Exponent can span entire width, requires leading-zero-counter, shifter scales in n

Takum FPGA Codec

Decoder Schematic



regime/antiregime determinator (E1), characteristic/exponent determinator (E2), special case detector (E3)

Takum FPGA Codec

Encoder Design

Takum FPGA Codec

Encoder Design

- SOTA FloPoCo posit encoders depend on external rounding information $\rightarrow$ less flexible, most expensive part!

Takum FPGA Codec

Encoder Design

- SOTA FloPoCo posit encoders depend on external rounding information $\rightarrow$ less flexible, most expensive part!
- Goal: Do everything in the encoder

Takum FPGA Codec

Encoder Design

- SOTA FloPoCo posit encoders depend on external rounding information → less flexible, most expensive part!
- Goal: Do everything in the encoder
- Perform rounding prediction in parallel to encoding, select at the end

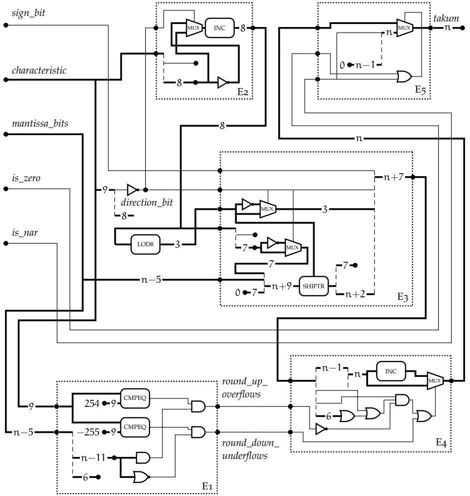
Takum FPGA Codec

Encoder Design

- SOTA FloPoCo posit encoders depend on external rounding information $\rightarrow$ less flexible, most expensive part!
- Goal: Do everything in the encoder
- Perform rounding prediction in parallel to encoding, select at the end
- Generate $(n+7)$-bit *extended takum* as a rounding candidate to avoid double-rounding

Takum FPGA Codec

Encoder Schematic



underflow/overflow predictor (E1), characteristic precursor determinator (E2), extended takum generator (E3), rounder (E4), output driver (E5)

Takum FPGA Codec

Benchmarks

[4] Raul Murillo et al. ‘Comparing different decodings for posit arithmetic’. CoNGA, 2022

Takum FPGA Codec

Benchmarks

Comparison against the state of the art (FloPoCo, 2018-2023)[4], target 200MHz

[4] Raul Murillo et al. 'Comparing different decodings for posit arithmetic'. CoNGA, 2022

Takum FPGA Codec

Benchmarks

Comparison against the state of the art (FloPoCo, 2018-2023)[4], target 200MHz

Decoder

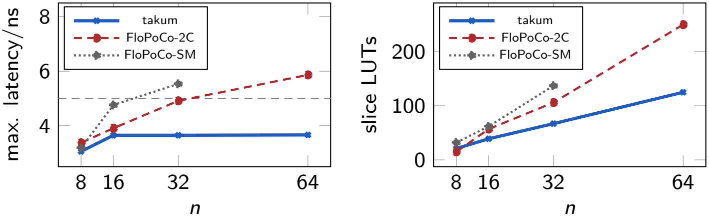
[4] Raul Murillo et al. 'Comparing different decodings for posit arithmetic'. CoNGA, 2022

Takum FPGA Codec

Benchmarks

Comparison against the state of the art (FloPoCo, 2018-2023)[4], target 200MHz

Decoder



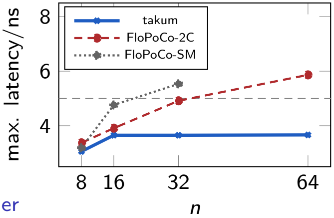
[4] Raul Murillo et al. 'Comparing different decodings for posit arithmetic'. CoNGA, 2022

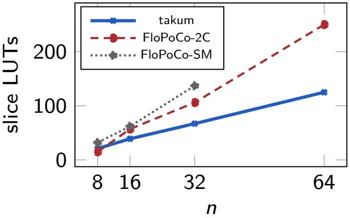
Takum FPGA Codec

Benchmarks

Comparison against the state of the art (FloPoCo, 2018-2023)[4], target 200MHz

Decoder





Encoder

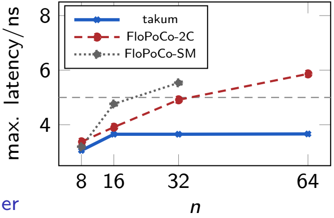
[4] Raul Murillo et al. 'Comparing different decodings for posit arithmetic'. CoNGA, 2022

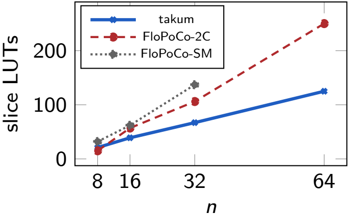
Takum FPGA Codec

Benchmarks

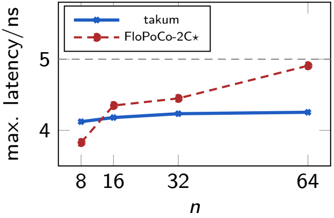
Comparison against the state of the art (FloPoCo, 2018-2023)[4], target 200MHz

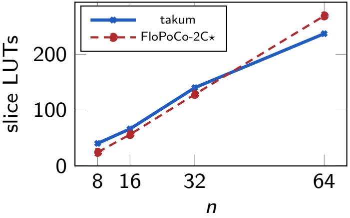
Decoder





Encoder





⋆: requires external rounding information

[4] Raul Murillo et al. 'Comparing different decodings for posit arithmetic'. CoNGA, 2022

Takum FPGA Codec

Conclusion and Outlook

Takum FPGA Codec

Conclusion and Outlook

- Takums lend themselves to efficient hardware implementations

Takum FPGA Codec

Conclusion and Outlook

- Takums lend themselves to efficient hardware implementations
- Posits have an insurmountable overhead due to the unbounded exponent width
  $\rightarrow$ B-Posits

Takum FPGA Codec

Conclusion and Outlook

- Takums lend themselves to efficient hardware implementations
- Posits have an insurmountable overhead due to the unbounded exponent width $\rightarrow$ B-Posits
- Posit APUs are moving towards more integrated codecs (see Thotli et al.'s 'Energy-Efficient Posit Arithmetic with Fast Approximate Division')

Takum FPGA Codec

Conclusion and Outlook

- Takums lend themselves to efficient hardware implementations
- Posits have an insurmountable overhead due to the unbounded exponent width $\rightarrow$ B-Posits
- Posit APUs are moving towards more integrated codecs (see Thotli et al.'s 'Energy-Efficient Posit Arithmetic with Fast Approximate Division')

SC25 Research Track (with James Quinlan, Stefan Wesner)

Numerical Performance of the Implicitly Restarted Arnoldi Method in OFP8, Bfloat16, Posit, and Takum Arithmetics

Tuesday, 3:52-4:15 p.m., Hall 274

Third Paper

Tekum: Balanced Ternary Tapered Precision Real Arithmetic

Tekums

Motivation and Prior Works

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science.

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)
  - Each trit has information content $\log_2(3) \approx 1.58$ bits.

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)
  - Each trit has information content $\log_2(3) \approx 1.58$ bits.
  - Only signed integers,

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)
  - Each trit has information content $\log_2(3) \approx 1.58$ bits.
  - Only signed integers, hardware-efficient negation (no increment),

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)
  - Each trit has information content $\log_2(3) \approx 1.58$ bits.
  - Only signed integers, hardware-efficient negation (no increment), subtraction as simple as addition,

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)
  - Each trit has information content $\log_2(3) \approx 1.58$ bits.
  - Only signed integers, hardware-efficient negation (no increment), subtraction as simple as addition, fewer carries,

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0,1,2\}$ have states $\{-1,0,1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)
  - Each trit has information content $\log_2(3) \approx 1.58$ bits.
  - Only signed integers, hardware-efficient negation (no increment), subtraction as simple as addition, fewer carries, truncation is rounding

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)
  - Each trit has information content $\log_2(3) \approx 1.58$ bits.
  - Only signed integers, hardware-efficient negation (no increment), subtraction as simple as addition, fewer carries, truncation is rounding
  - Knuth: 'perhaps the prettiest number system of all'

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)
  - Each trit has information content $\log_2(3) \approx 1.58$ bits.
  - Only signed integers, hardware-efficient negation (no increment), subtraction as simple as addition, fewer carries, truncation is rounding
  - Knuth: 'perhaps the prettiest number system of all'
- Ternary hardware is coming (CNTFETs, optical computers, ternary LLMs)

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)
  - Each trit has information content $\log_2(3) \approx 1.58$ bits.
  - Only signed integers, hardware-efficient negation (no increment), subtraction as simple as addition, fewer carries, truncation is rounding
  - Knuth: 'perhaps the prettiest number system of all'
- Ternary hardware is coming (CNTFETs, optical computers, ternary LLMs)
- Can you represent real numbers in balanced ternary? Don't want practice to overtake theory, as with AI number formats

Tekums

Motivation and Prior Works

- Binary logic is the foundation of computer science. But is it the best choice?
- Radix economy: base complexity vs. representation length, optimal for base $\mathrm{e} \approx 2.718$
  - Base-2 leans on base simplicity, but higher representation length (memory wall)
  - Base-3 closest to the optimal, slightly higher base-complexity, lower representation complexity
- Odd bases can be balanced: For Base-3, instead of $\{0, 1, 2\}$ have states $\{-1, 0, 1\}$ and define $\mathrm{T} := -1$ (balanced ternary)
  - Works like any other base (e.g. $1\mathrm{T}0$ represents $9 + (-3) + 0 = 6$)
  - Each trit has information content $\log_2(3) \approx 1.58$ bits.
  - Only signed integers, hardware-efficient negation (no increment), subtraction as simple as addition, fewer carries, truncation is rounding
  - Knuth: 'perhaps the prettiest number system of all'
- Ternary hardware is coming (CNTFETs, optical computers, ternary LLMs)
- Can you represent real numbers in balanced ternary? Don't want practice to overtake theory, as with AI number formats
- No prior works, except `ternary27` by O'Hare (unpublished Hackaday project, inspired by IEEE 754, tritfield approach...), archived on Zenodo for posterity

Tekums

First Filter - Asymmetry

Tekums

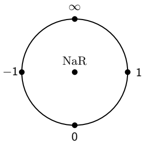
First Filter - Asymmetry

- During the design phase, three major blockers could be identified. We call them ‘filters’ (referring to Robert HANSON’s ‘Great Filter’)

Tekums

First Filter - Asymmetry

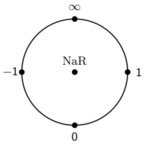
- During the design phase, three major blockers could be identified. We call them 'filters' (referring to Robert HANSON's 'Great Filter')
- Real wheel algebra has four quadrants and five special cases (0, 1, -1, ∞, NaR)



Tekums

First Filter - Asymmetry

- During the design phase, three major blockers could be identified. We call them 'filters' (referring to Robert Hanson's 'Great Filter')
- Real wheel algebra has four quadrants and five special cases (0, 1, -1, ∞, NaR)

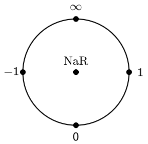


- Powers of three are odd, and $4r + 5$ (for $r \in \mathbb{N}_0$) is also odd (reason why posits/takums have only one of ∞ or NaR)

Tekums

First Filter - Asymmetry

- During the design phase, three major blockers could be identified. We call them 'filters' (referring to Robert Hanson's 'Great Filter')
- Real wheel algebra has four quadrants and five special cases (0, 1, -1, ∞, NaR)

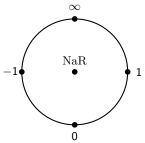


- Powers of three are odd, and $4r + 5$ (for $r \in \mathbb{N}_0$) is also odd (reason why posits/takums have only one of ∞ or NaR)
- Want monotonicity: Line up ternary representations in ascending order

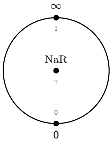
Tekums

First Filter - Asymmetry

- During the design phase, three major blockers could be identified. We call them 'filters' (referring to Robert HANSON's 'Great Filter')
- Real wheel algebra has four quadrants and five special cases (0, 1, -1, ∞, NaR)



- Powers of three are odd, and $4r + 5$ (for $r \in \mathbb{N}_0$) is also odd (reason why posits/takums have only one of ∞ or NaR)
- Want monotonicity: Line up ternary representations in ascending order

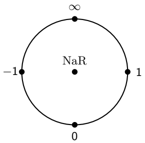


(a) $n = 1$

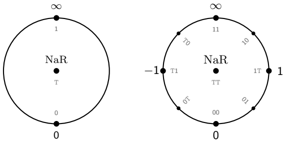
Tekums

First Filter - Asymmetry

- During the design phase, three major blockers could be identified. We call them ‘filters’ (referring to Robert HANSON’s ‘Great Filter’)
- Real wheel algebra has four quadrants and five special cases (0, 1, -1, ∞, NaR)



- Powers of three are odd, and $4r + 5$ (for $r \in \mathbb{N}_0$) is also odd (reason why posits/takums have only one of ∞ or NaR)
- Want monotonicity: Line up ternary representations in ascending order

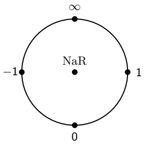


(a) $n = 1$ (b) $n = 2$

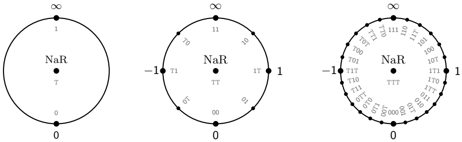
Tekums

First Filter - Asymmetry

- During the design phase, three major blockers could be identified. We call them ‘filters’ (referring to Robert HANSON’s ‘Great Filter’)
- Real wheel algebra has four quadrants and five special cases (0, 1, -1, ∞, NaR)



- Powers of three are odd, and $4r + 5$ (for $r \in \mathbb{N}_0$) is also odd (reason why posits/takums have only one of ∞ or NaR)
- Want monotonicity: Line up ternary representations in ascending order

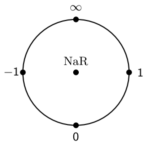


(a) $n = 1$ (b) $n = 2$ (c) $n = 3$

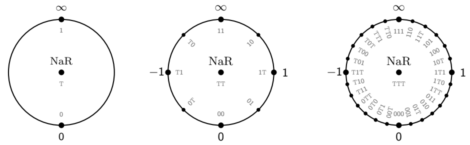
Tekums

First Filter - Asymmetry

- During the design phase, three major blockers could be identified. We call them 'filters' (referring to Robert HANSON's 'Great Filter')
- Real wheel algebra has four quadrants and five special cases (0, 1, -1, ∞, NaR)



- Powers of three are odd, and $4r + 5$ (for $r \in \mathbb{N}_0$) is also odd (reason why posits/takums have only one of ∞ or NaR)
- Want monotonicity: Line up ternary representations in ascending order



(a) $n = 1$ (b) $n = 2$ (c) $n = 3$

- Solution: It holds $4 \mid (3^{2n} - 5)$ and $4 \nmid (3^n - 4)$! Dropping one special case will never work, but even ns always work.

Tekums

Second Filter - Misfit Tool

Tekums

Second Filter - Misfit Tool

- Problem: posit prefix strings cannot be used in ternary

Tekums

Second Filter - Misfit Tool

- Problem: posit prefix strings cannot be used in ternary
- Consider a positive trit string $\boldsymbol{t} \in \{0 \cdots 01, \ldots, 1 \cdots 10\}$

Tekums

Second Filter - Misfit Tool

- Problem: posit prefix strings cannot be used in ternary
- Consider a positive trit string $\boldsymbol{t} \in \{0 \cdots 01, \dots, 1 \cdots 10\}$
- First insight: 1 has ternary representation $1\mathrm{T} \cdots 1\mathrm{T}$ (in the middle of 0 and ∞)

Tekums

Second Filter - Misfit Tool

- Problem: posit prefix strings cannot be used in ternary
- Consider a positive trit string $\boldsymbol{t} \in \{0\cdots01,\ldots,1\cdots10\}$
- First insight: 1 has ternary representation $1\mathrm{T}\cdots1\mathrm{T}$ (in the middle of 0 and ∞)
- 1 has exponent zero, so subtract $1\mathrm{T}\cdots1\mathrm{T}$ from $\boldsymbol{t}$ to obtain range $\{\mathrm{T}1\cdots\mathrm{T}10\mathrm{T},\ldots,1\mathrm{T}\cdots1\mathrm{T}01\}$

Tekums

Second Filter - Misfit Tool

- Problem: posit prefix strings cannot be used in ternary
- Consider a positive trit string $\boldsymbol{t} \in \{0\cdots01, \dots, 1\cdots10\}$
- First insight: 1 has ternary representation $1\mathrm{T}\cdots1\mathrm{T}$ (in the middle of 0 and ∞)
- 1 has exponent zero, so subtract $1\mathrm{T}\cdots1\mathrm{T}$ from $\boldsymbol{t}$ to obtain range $\{\mathrm{T}1\cdots\mathrm{T}10\mathrm{T}, \dots, 1\mathrm{T}\cdots1\mathrm{T}01\}$
- Works the same for negative numbers, define *anchor function* $\mathrm{anc}(\boldsymbol{t}) := |t| - 1\mathrm{T}\cdots1\mathrm{T}$.

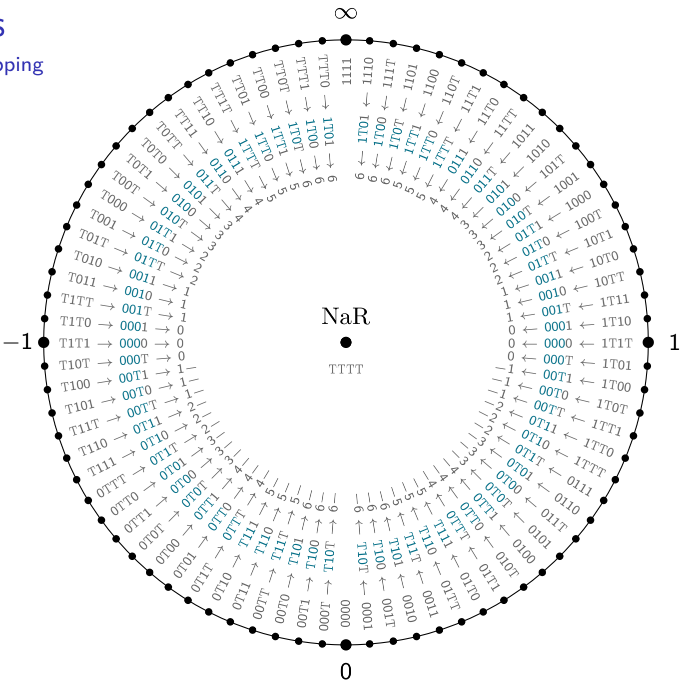
Tekums

Second Filter - Misfit Tool

- Problem: posit prefix strings cannot be used in ternary
- Consider a positive trit string $\boldsymbol{t} \in \{0 \cdots 01, \ldots, 1 \cdots 10\}$
- First insight: 1 has ternary representation $1\mathrm{T} \cdots 1\mathrm{T}$ (in the middle of 0 and ∞)
- 1 has exponent zero, so subtract $1\mathrm{T} \cdots 1\mathrm{T}$ from $\boldsymbol{t}$ to obtain range $\{\mathrm{T}1 \cdots \mathrm{T}10\mathrm{T}, \ldots, 1\mathrm{T} \cdots 1\mathrm{T}01\}$
- Works the same for negative numbers, define *anchor function* $\mathrm{anc}(\boldsymbol{t}) := |t| - 1\mathrm{T} \cdots 1\mathrm{T}$.
- First three trits of anchor are in the range $\mathrm{T}1\mathrm{T} = -7$ to $1\mathrm{T}1 = 7$. Use the takum approach, designate first three trits to be regime (two would be too few, four would be too many)

Tekums

Regime Mapping



Tekums

Third Filter - Excess

Tekums

Third Filter - Excess

- Regime values r are in the range -7 to 7

Tekums

Third Filter - Excess

- Regime values r are in the range -7 to 7
- Map regime modulus to number of exponent trits $\boldsymbol{e}$ following regime trits, apply bias for proper ranges

r	0	1	2	3	4	5	6	7
$\lvert r \rvert$	0	1	2	3	4	5	6	7
$\mathrm{int}(\boldsymbol{e})$	$0\,..\,0$	$-1\,..\,1$	$-4\,..\,4$	$-13\,..\,13$	$-40\,..\,40$	$-121\,..\,121$	$-364\,..\,364$	$-1093\,..\,-547$
b	0	2	8	26	80	242	728	2186
e	$0\,..\,0$	$1\,..\,3$	$4\,..\,12$	$13\,..\,39$	$40\,..\,120$	$121\,..\,363$	$364\,..\,1092$	$1093\,..\,1639$

Tekums

Third Filter - Excess

- Regime values r are in the range -7 to 7
- Map regime modulus to number of exponent trits $\boldsymbol{e}$ following regime trits, apply bias for proper ranges

r	0	1	2	3	4	5	6	7
$\lvert r\rvert$	0	1	2	3	4	5	6	7
$\mathrm{int}(\boldsymbol{e})$	$0\,..\,0$	$-1\,..\,1$	$-4\,..\,4$	$-13\,..\,13$	$-40\,..\,40$	$-121\,..\,121$	$-364\,..\,364$	$-1093\,..\,-547$
b	0	2	8	26	80	242	728	2186
e	$0\,..\,0$	$1\,..\,3$	$4\,..\,12$	$13\,..\,39$	$40\,..\,120$	$121\,..\,363$	$364\,..\,1092$	$1093\,..\,1639$

- $3^{1639} \approx 10^{782}$, which is excessive; we want around 10^{55}

Tekums

Third Filter - Excess

- Regime values r are in the range -7 to 7
- Map regime modulus to number of exponent trits $\boldsymbol{e}$ following regime trits, apply bias for proper ranges

r	0	1	2	3	4	5	6	7
$\lvert r \rvert$	0	1	2	3	4	5	6	7
$\operatorname{int}(\boldsymbol{e})$	$0..0$	$-1..1$	$-4..4$	$-13..13$	$-40..40$	$-121..121$	$-364..364$	$-1093..-547$
b	0	2	8	26	80	242	728	2186
e	$0..0$	$1..3$	$4..12$	$13..39$	$40..120$	$121..363$	$364..1092$	$1093..1639$

- $3^{1639} \approx 10^{782}$, which is excessive; we want around 10^{55}
- Try other mappings (see paper); best choice:

r	0	1	2	3	4	5	6	7
$\max(0, \lvert r \rvert - 2)$	0	0	0	1	2	3	4	5
$\operatorname{int}(\boldsymbol{e})$	$0..0$	$0..0$	$0..0$	$-1..1$	$-4..4$	$-13..13$	$-40..40$	$-121..-61$
b	0	1	2	4	10	28	82	244
e	$0..0$	$1..1$	$2..2$	$3..5$	$6..14$	$15..41$	$42..122$	$123..183$

Tekums

Third Filter - Excess

- Regime values r are in the range -7 to 7
- Map regime modulus to number of exponent trits $\boldsymbol{e}$ following regime trits, apply bias for proper ranges

r	0	1	2	3	4	5	6	7
$\|r\|$	0	1	2	3	4	5	6	7
$\mathrm{int}(\boldsymbol{e})$	$0..0$	$-1..1$	$-4..4$	$-13..13$	$-40..40$	$-121..121$	$-364..364$	$-1093..-547$
b	0	2	8	26	80	242	728	2186
e	$0..0$	$1..3$	$4..12$	$13..39$	$40..120$	$121..363$	$364..1092$	$1093..1639$

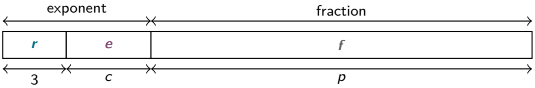
- $3^{1639} \approx 10^{782}$, which is excessive; we want around 10^{55}
- Try other mappings (see paper); best choice:

r	0	1	2	3	4	5	6	7
$\max(0, \|r\| - 2)$	0	0	0	1	2	3	4	5
$\mathrm{int}(\boldsymbol{e})$	$0..0$	$0..0$	$0..0$	$-1..1$	$-4..4$	$-13..13$	$-40..40$	$-121..-61$
b	0	1	2	4	10	28	82	244
e	$0..0$	$1..1$	$2..2$	$3..5$	$6..14$	$15..41$	$42..122$	$123..183$

- $3^{183} \approx 10^{87}$, which is a good dynamic range

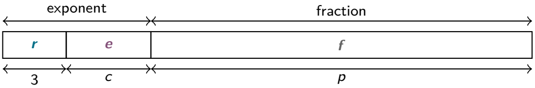
Tekums

Definition



Tekums

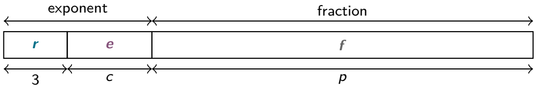
Definition



- Let $n \in 2\mathbb{N}$, $n \geq 8$ and $\boldsymbol{t} \in \{\mathrm{T}, 0, 1\}^n$

Tekums

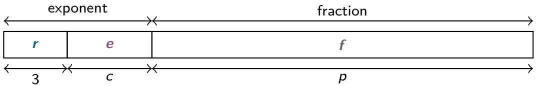
Definition



- Let $n \in 2\mathbb{N}$, $n \geq 8$ and $\boldsymbol{t} \in \{\mathrm{T}, 0, 1\}^n$
- Define $\boldsymbol{r} + \boldsymbol{e} + \boldsymbol{f} := \mathrm{anc}_n(\boldsymbol{t})$ (*regime trits* $\boldsymbol{r}$, *exponent trits* $\boldsymbol{e}$, *fraction trits* $\boldsymbol{f}$)

Tekums

Definition

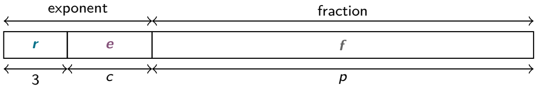


- Let $n \in 2\mathbb{N}$, $n \geq 8$ and $\boldsymbol{t} \in \{\mathrm{T}, 0, 1\}^n$
- Define $\boldsymbol{r} \mathbin{+\!\!+} \boldsymbol{e} \mathbin{+\!\!+} \boldsymbol{f} := \mathrm{anc}_n(\boldsymbol{t})$ (*regime trits* $\boldsymbol{r}$, *exponent trits* $\boldsymbol{e}$, *fraction trits* $\boldsymbol{f}$)
- Define

$$s := \mathrm{sign}(\mathrm{int}_n(\boldsymbol{t})) \qquad : \text{sign}$$

Tekums

Definition



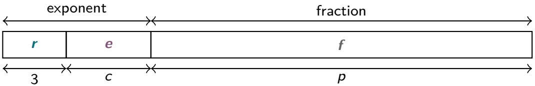
- Let $n \in 2\mathbb{N}$, $n \geq 8$ and $\boldsymbol{t} \in \{\mathrm{T}, 0, 1\}^n$
- Define $\boldsymbol{r} \mathbin{+\!\!+} \boldsymbol{e} \mathbin{+\!\!+} \boldsymbol{f} := \mathrm{anc}_n(\boldsymbol{t})$ (*regime trits* $\boldsymbol{r}$, *exponent trits* $\boldsymbol{e}$, *fraction trits* $\boldsymbol{f}$)
- Define

$$s := \mathrm{sign}(\mathrm{int}_n(\boldsymbol{t})) \quad : \text{sign}$$

$$r := \mathrm{int}_3(\boldsymbol{r}) \in \{-7, \ldots, 7\} \quad : \text{regime value}$$

Tekums

Definition

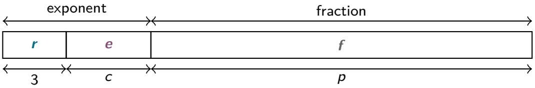


- Let $n \in 2\mathbb{N}$, $n \geq 8$ and $\boldsymbol{t} \in \{\mathrm{T}, 0, 1\}^n$
- Define $\boldsymbol{r} + \boldsymbol{e} + \boldsymbol{f} := \mathrm{anc}_n(\boldsymbol{t})$ (*regime trits* $\boldsymbol{r}$, *exponent trits* $\boldsymbol{e}$, *fraction trits* $\boldsymbol{f}$)
- Define

$$s := \mathrm{sign}(\mathrm{int}_n(\boldsymbol{t})) \quad : \text{sign}$$
$$r := \mathrm{int}_3(\boldsymbol{r}) \in \{-7, \dots, 7\} \quad : \text{regime value}$$
$$c := \max(0, |r| - 2) \in \{0, \dots, 5\} \quad : \text{exponent trit count}$$

Tekums

Definition

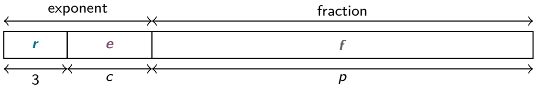


- Let $n \in 2\mathbb{N}$, $n \geq 8$ and $\boldsymbol{t} \in \{\mathrm{T}, 0, 1\}^n$
- Define $\boldsymbol{r} \mathbin{+\!\!\!+} \boldsymbol{e} \mathbin{+\!\!\!+} \boldsymbol{f} := \operatorname{anc}_n(\boldsymbol{t})$ (*regime trits* $\boldsymbol{r}$, *exponent trits* $\boldsymbol{e}$, *fraction trits* $\boldsymbol{f}$)
- Define

$$
\begin{aligned}
s &:= \operatorname{sign}(\operatorname{int}_n(\boldsymbol{t})) && : \text{sign} \\
r &:= \operatorname{int}_3(\boldsymbol{r}) \in \{-7, \dots, 7\} && : \text{regime value} \\
c &:= \max(0, |r| - 2) \in \{0, \dots, 5\} && : \text{exponent trit count} \\
p &:= n - c - 3 \in \{n - 8, \dots, n - 3\} && : \text{fraction trit count}
\end{aligned}
$$

Tekums

Definition

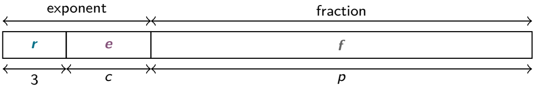


- Let $n \in 2\mathbb{N}$, $n \geq 8$ and $\boldsymbol{t} \in \{\mathrm{T}, 0, 1\}^n$
- Define $\boldsymbol{r} \mathbin{+\!\!+} \boldsymbol{e} \mathbin{+\!\!+} \boldsymbol{f} := \mathrm{anc}_n(\boldsymbol{t})$ (*regime trits* $\boldsymbol{r}$, *exponent trits* $\boldsymbol{e}$, *fraction trits* $\boldsymbol{f}$)
- Define

$$
\begin{aligned}
s &:= \mathrm{sign}(\mathrm{int}_n(\boldsymbol{t})) && : \text{sign} \\
r &:= \mathrm{int}_3(\boldsymbol{r}) \in \{-7, \dots, 7\} && : \text{regime value} \\
c &:= \max(0, |r| - 2) \in \{0, \dots, 5\} && : \text{exponent trit count} \\
p &:= n - c - 3 \in \{n - 8, \dots, n - 3\} && : \text{fraction trit count} \\
b &:= \mathrm{sign}(r) \cdot (0, 1, 2, 4, 10, 28, 82, 244)_{|r|} && : \text{exponent bias}
\end{aligned}
$$

Tekums

Definition

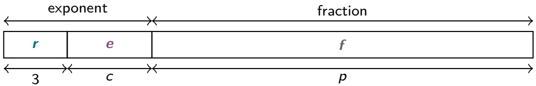


- Let $n \in 2\mathbb{N}$, $n \geq 8$ and $\boldsymbol{t} \in \{\mathrm{T}, 0, 1\}^n$
- Define $\boldsymbol{r} \mathbin{+\!\!+} \boldsymbol{e} \mathbin{+\!\!+} \boldsymbol{f} := \mathrm{anc}_n(\boldsymbol{t})$ (*regime trits* $\boldsymbol{r}$, *exponent trits* $\boldsymbol{e}$, *fraction trits* $\boldsymbol{f}$)
- Define

$$
\begin{aligned}
s &:= \mathrm{sign}(\mathrm{int}_n(\boldsymbol{t})) && : \text{sign} \\
r &:= \mathrm{int}_3(\boldsymbol{r}) \in \{-7, \dots, 7\} && : \text{regime value} \\
c &:= \max(0, |r| - 2) \in \{0, \dots, 5\} && : \text{exponent trit count} \\
p &:= n - c - 3 \in \{n - 8, \dots, n - 3\} && : \text{fraction trit count} \\
b &:= \mathrm{sign}(r) \cdot (0, 1, 2, 4, 10, 28, 82, 244)_{|r|} && : \text{exponent bias} \\
e &:= \mathrm{int}_c(\boldsymbol{e}) + b \in \{-183, \dots, 183\} && : \text{exponent value}
\end{aligned}
$$

Tekums

Definition

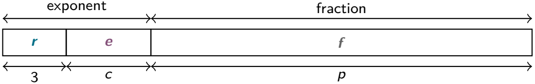


- Let $n \in 2\mathbb{N}$, $n \geq 8$ and $\boldsymbol{t} \in \{\mathrm{T}, 0, 1\}^n$
- Define $\boldsymbol{r} \mathbin{+\!\!+} \boldsymbol{e} \mathbin{+\!\!+} \boldsymbol{f} := \mathrm{anc}_n(\boldsymbol{t})$ (*regime trits* $\boldsymbol{r}$, *exponent trits* $\boldsymbol{e}$, *fraction trits* $\boldsymbol{f}$)
- Define

$s := \mathrm{sign}(\mathrm{int}_n(\boldsymbol{t}))$	: sign
$r := \mathrm{int}_3(\boldsymbol{r}) \in \{-7, \dots, 7\}$	: regime value
$c := \max(0, \lvert r \rvert - 2) \in \{0, \dots, 5\}$	: exponent trit count
$p := n - c - 3 \in \{n - 8, \dots, n - 3\}$	: fraction trit count
$b := \mathrm{sign}(r) \cdot (0, 1, 2, 4, 10, 28, 82, 244)_{\lvert r \rvert}$	: exponent bias
$e := \mathrm{int}_c(\boldsymbol{e}) + b \in \{-183, \dots, 183\}$	: exponent value
$f := 3^{-p}\,\mathrm{int}_p(\boldsymbol{f}) \in (-0.5, 0.5)$	: fraction value

Tekums

Definition



- Let $n \in 2\mathbb{N}$, $n \geq 8$ and $\boldsymbol{t} \in \{\mathrm{T}, 0, 1\}^n$
- Define $\boldsymbol{r} \mathbin{+\!\!+} \boldsymbol{e} \mathbin{+\!\!+} \boldsymbol{f} := \mathrm{anc}_n(\boldsymbol{t})$ (*regime trits* $\boldsymbol{r}$, *exponent trits* $\boldsymbol{e}$, *fraction trits* $\boldsymbol{f}$)
- Define

$$
\begin{aligned}
s &:= \mathrm{sign}(\mathrm{int}_n(\boldsymbol{t})) && : \text{sign} \\
r &:= \mathrm{int}_3(\boldsymbol{r}) \in \{-7, \dots, 7\} && : \text{regime value} \\
c &:= \max(0, |r| - 2) \in \{0, \dots, 5\} && : \text{exponent trit count} \\
p &:= n - c - 3 \in \{n - 8, \dots, n - 3\} && : \text{fraction trit count} \\
b &:= \mathrm{sign}(r) \cdot (0, 1, 2, 4, 10, 28, 82, 244)_{|r|} && : \text{exponent bias} \\
e &:= \mathrm{int}_c(\boldsymbol{e}) + b \in \{-183, \dots, 183\} && : \text{exponent value} \\
f &:= 3^{-p}\,\mathrm{int}_p(\boldsymbol{f}) \in (-0.5, 0.5) && : \text{fraction value}
\end{aligned}
$$

- Obtain the tekum value

$$
\theta_n(\boldsymbol{t}) := \begin{cases} \mathrm{NaR} & \boldsymbol{r} \mathbin{+\!\!+} \boldsymbol{e} \mathbin{+\!\!+} \boldsymbol{f} = \mathrm{T} \cdots \mathrm{T} \\ 0 & \boldsymbol{r} \mathbin{+\!\!+} \boldsymbol{e} \mathbin{+\!\!+} \boldsymbol{f} = 0 \cdots 0 \\ \infty & \boldsymbol{r} \mathbin{+\!\!+} \boldsymbol{e} \mathbin{+\!\!+} \boldsymbol{f} = 1 \cdots 1 \\ s \cdot (1 + f) \cdot 3^e & \text{otherwise} \end{cases}
$$

Tekums

Properties

Tekums

Properties

- Only 3 trit fields instead of 5 bit fields (posits, takums)

Tekums

Properties

- Only 3 trit fields instead of 5 bit fields (posits, takums)
- regime and exponent trits together only at most 8 trits long

Tekums

Properties

- Only 3 trit fields instead of 5 bit fields (posits, takums)
- regime and exponent trits together only at most 8 trits long
- No redundant representations

Tekums

Properties

- Only 3 trit fields instead of 5 bit fields (posits, takums)
- regime and exponent trits together only at most 8 trits long
- No redundant representations
- Monotonic in its ternary representation

Tekums

Properties

- Only 3 trit fields instead of 5 bit fields (posits, takums)
- regime and exponent trits together only at most 8 trits long
- No redundant representations
- Monotonic in its ternary representation
- Negating the ternary string negates the tekum value

Tekums

Properties

- Only 3 trit fields instead of 5 bit fields (posits, takums)
- regime and exponent trits together only at most 8 trits long
- No redundant representations
- Monotonic in its ternary representation
- Negating the ternary string negates the tekum value
- Truncation is rounding

Tekums

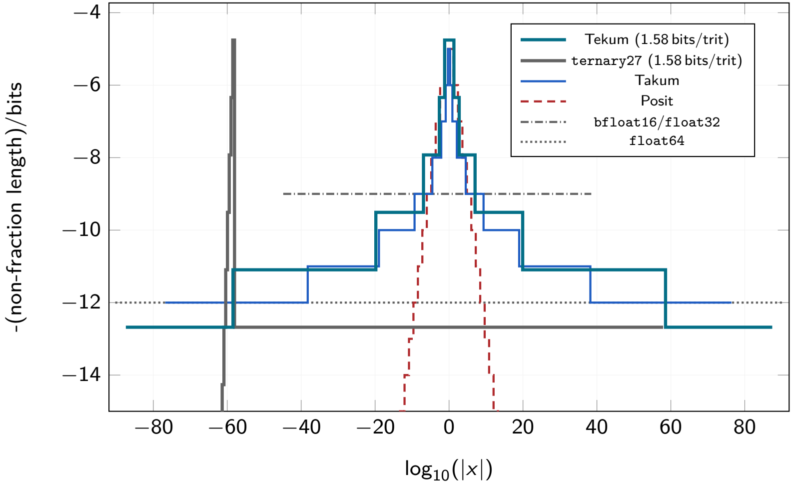
Properties

- Only 3 trit fields instead of 5 bit fields (posits, takums)
- regime and exponent trits together only at most 8 trits long
- No redundant representations
- Monotonic in its ternary representation
- Negating the ternary string negates the tekum value
- Truncation is rounding

How do we compare tekums against binary number formats?

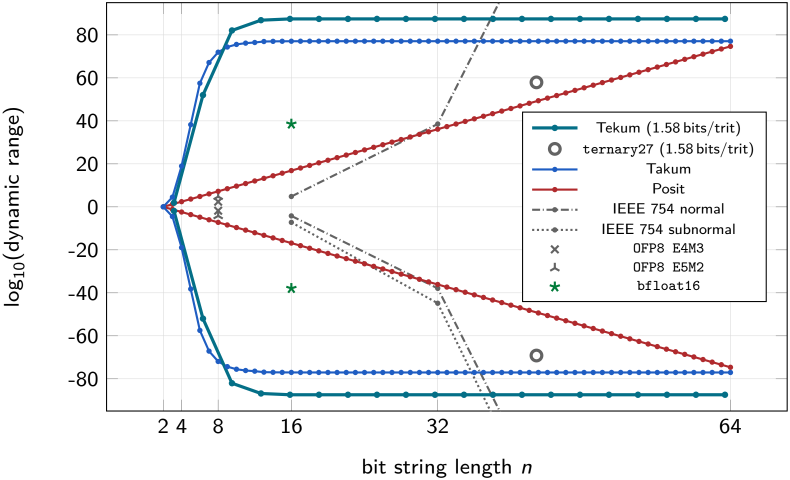
Tekums

Accuracy



Tekums

Dynamic Range



Tekums

Conclusion and Outlook

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)
- Challenge: Bit-budgets hard to translate (5 trits $\approx$ 8 bits, 10 trits $\approx$ 16 bits, 20 trits $\approx$ 32 bits, 40 trits $\approx$ 64 bits)

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)
- Challenge: Bit-budgets hard to translate (5 trits $\approx$ 8 bits, 10 trits $\approx$ 16 bits, 20 trits $\approx$ 32 bits, 40 trits $\approx$ 64 bits)
- How might computers look like in 100 years? Avoid letting practice overtake theory

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)
- Challenge: Bit-budgets hard to translate (5 trits $\approx$ 8 bits, 10 trits $\approx$ 16 bits, 20 trits $\approx$ 32 bits, 40 trits $\approx$ 64 bits)
- How might computers look like in 100 years? Avoid letting practice overtake theory
  - Steam engines before thermodynamics,

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)
- Challenge: Bit-budgets hard to translate (5 trits $\approx$ 8 bits, 10 trits $\approx$ 16 bits, 20 trits $\approx$ 32 bits, 40 trits $\approx$ 64 bits)
- How might computers look like in 100 years? Avoid letting practice overtake theory
  - Steam engines before thermodynamics, flight before aerodynamics,

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)
- Challenge: Bit-budgets hard to translate (5 trits $\approx$ 8 bits, 10 trits $\approx$ 16 bits, 20 trits $\approx$ 32 bits, 40 trits $\approx$ 64 bits)
- How might computers look like in 100 years? Avoid letting practice overtake theory
  - Steam engines before thermodynamics, flight before aerodynamics, genetics before DNA/molecular biology

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)
- Challenge: Bit-budgets hard to translate (5 trits $\approx$ 8 bits, 10 trits $\approx$ 16 bits, 20 trits $\approx$ 32 bits, 40 trits $\approx$ 64 bits)
- How might computers look like in 100 years? Avoid letting practice overtake theory
  - Steam engines before thermodynamics, flight before aerodynamics, genetics before DNA/molecular biology
  - Germ theory,

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)
- Challenge: Bit-budgets hard to translate (5 trits $\approx$ 8 bits, 10 trits $\approx$ 16 bits, 20 trits $\approx$ 32 bits, 40 trits $\approx$ 64 bits)
- How might computers look like in 100 years? Avoid letting practice overtake theory
  - Steam engines before thermodynamics, flight before aerodynamics, genetics before DNA/molecular biology
  - Germ theory, relativity and quantum mechanics,

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)
- Challenge: Bit-budgets hard to translate (5 trits $\approx$ 8 bits, 10 trits $\approx$ 16 bits, 20 trits $\approx$ 32 bits, 40 trits $\approx$ 64 bits)
- How might computers look like in 100 years? Avoid letting practice overtake theory
  - Steam engines before thermodynamics, flight before aerodynamics, genetics before DNA/molecular biology
  - Germ theory, relativity and quantum mechanics, TURING's Universal Machine

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)
- Challenge: Bit-budgets hard to translate (5 trits $\approx$ 8 bits, 10 trits $\approx$ 16 bits, 20 trits $\approx$ 32 bits, 40 trits $\approx$ 64 bits)
- How might computers look like in 100 years? Avoid letting practice overtake theory
  - Steam engines before thermodynamics, flight before aerodynamics, genetics before DNA/molecular biology
  - Germ theory, relativity and quantum mechanics, TURING's Universal Machine
- We can simulate arithmetics of any kind, let's make use of it!

Tekums

Conclusion and Outlook

- Tekums carry over properties from posits and takums to balanced ternary
- Many interesting properties, warranting more exploration (Julia implementation WIP)
- Challenge: Bit-budgets hard to translate (5 trits $\approx$ 8 bits, 10 trits $\approx$ 16 bits, 20 trits $\approx$ 32 bits, 40 trits $\approx$ 64 bits)
- How might computers look like in 100 years? Avoid letting practice overtake theory
  - Steam engines before thermodynamics, flight before aerodynamics, genetics before DNA/molecular biology
  - Germ theory, relativity and quantum mechanics, TURING's Universal Machine
- We can simulate arithmetics of any kind, let's make use of it!

What will the arithmetic of the future be?
Tailored for AI or general-purpose?

takum
arithmetic