lecture 3

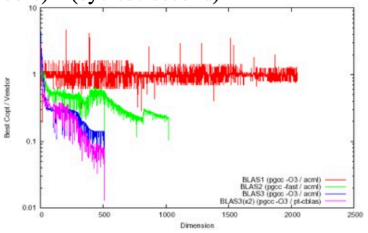
- BLAS complexity justified
- a word on error (absolute, relative, approximate, forward, backward, total)
- finite bit representation of numbers - introduction (big and little endian)
- IEEE floating point representation of numbers

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

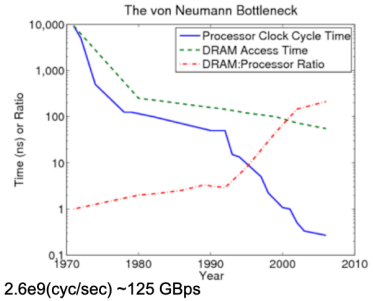
Memory Wall *Always* There ...

Computation: Theoretical peak: (# cpu cores) * (flops / cycle / core) * (cycles / second)

Memory: Theoretical peak: (bus width) * (bus speed)



BLAS 1: O(n) operations on O(n) operands
BLAS 2: O(n**2) operations on O(n**2) operands
BLAS 3: O(n**3) operations on O(n**2) operands

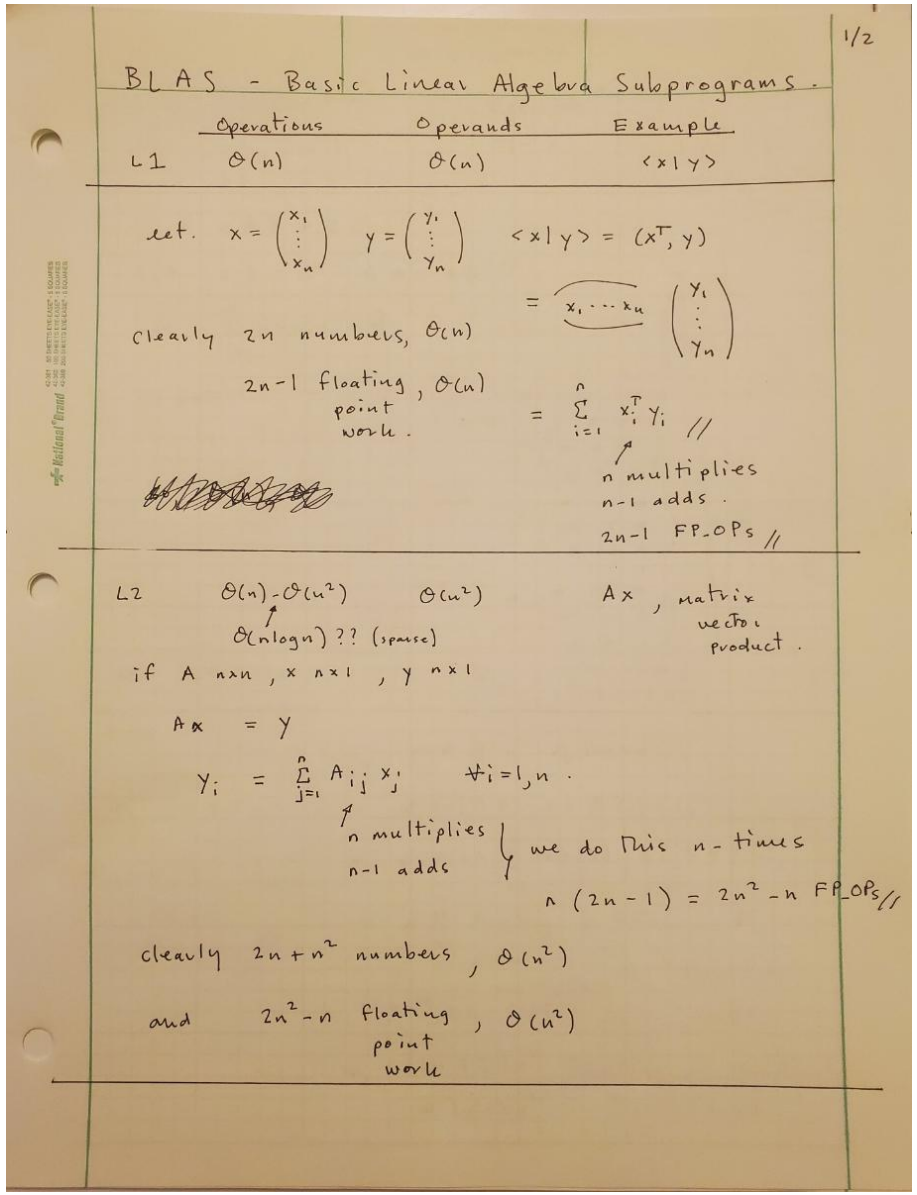


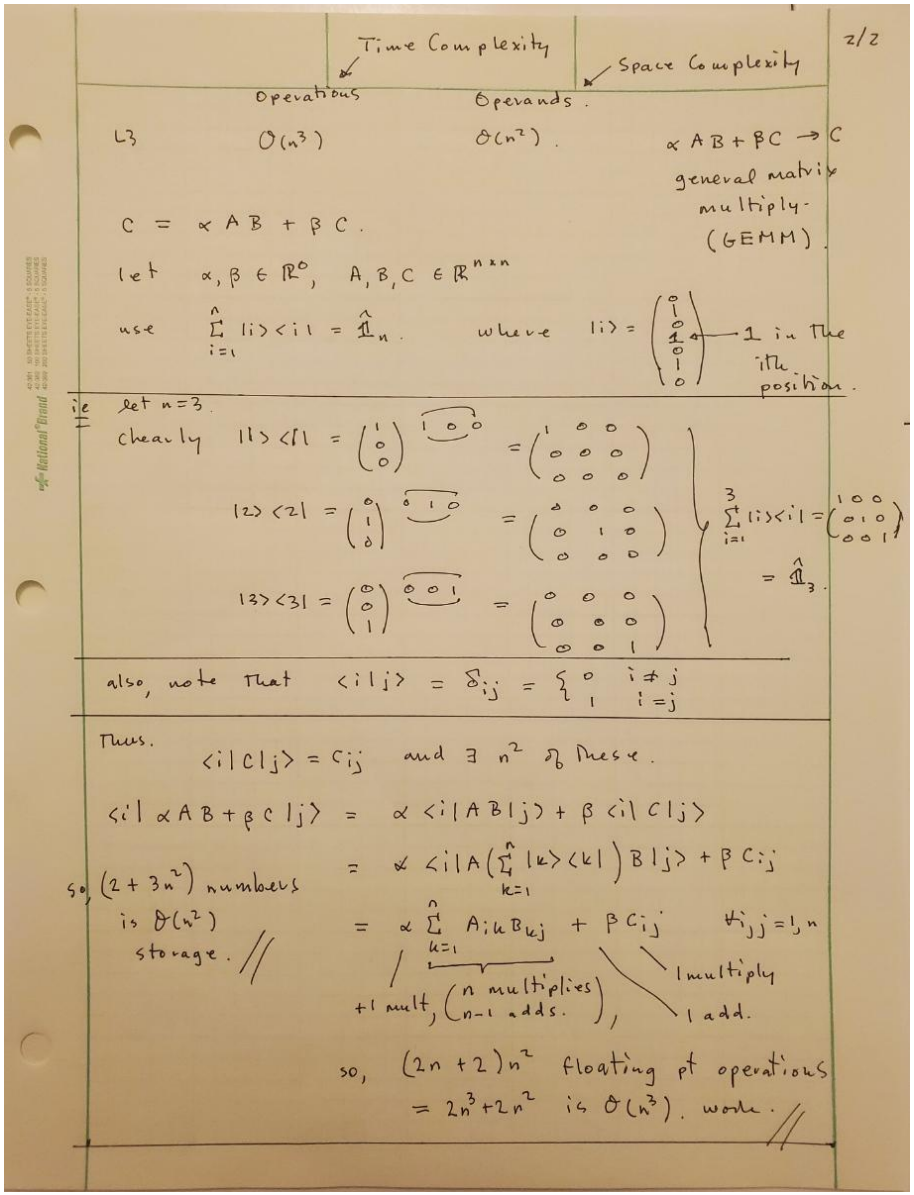
$y = \alpha x + y$:
3 loads, 1 store
(more expensive than FP_OPs by a long shot)
2 floating point operations (maybe 1) on 3 operands

eg, double precision on the FY10 target platform:
(3 operands / 2 flop) * (8 bytes / operand) * 6 core * 4 (flop / cyc /core) * 2.6e9(cyc/sec) ~125 GBps

... We don't have this and to get it is $$$ ... how to achieve **Sustainability??**

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025





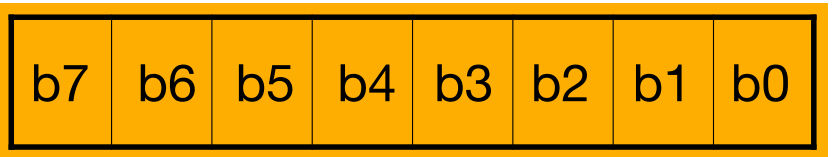
Matrix multiply
- a cornerstone of many applications including linear algebra, PDE solvers, and ML (both training and inference)

Kenneth J. Roche
University of Washington

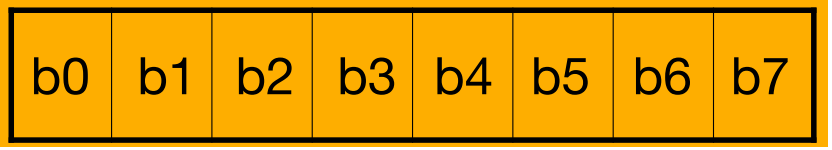
Powers of 2 (2^n)	Prefix (SI)	Numerical Range [0, $2^n - 1$]
2^{10}	Kilo (K)	0 to 1,023
2^{20}	Mega (M)	0 to 1,048,575
2^{30}	Giga (G)	0 to 1,073,741,823
2^{40}	Tera (T)	0 to 1,099,511,627,775
2^{50}	Peta (P)	0 to 1,125,899,906,842,623
2^{60}	Exa (E)	0 to 1,152,921,504,606,846,975

1 bit : {0,1} : 2^1 combinations
8 bits = 1 byte = 1 B
1 B : 2^8 = 256 combinations

Big endian, most significant bit (2^7) stored leftmost.



Little endian, least significant bit (2^0) stored leftmost.



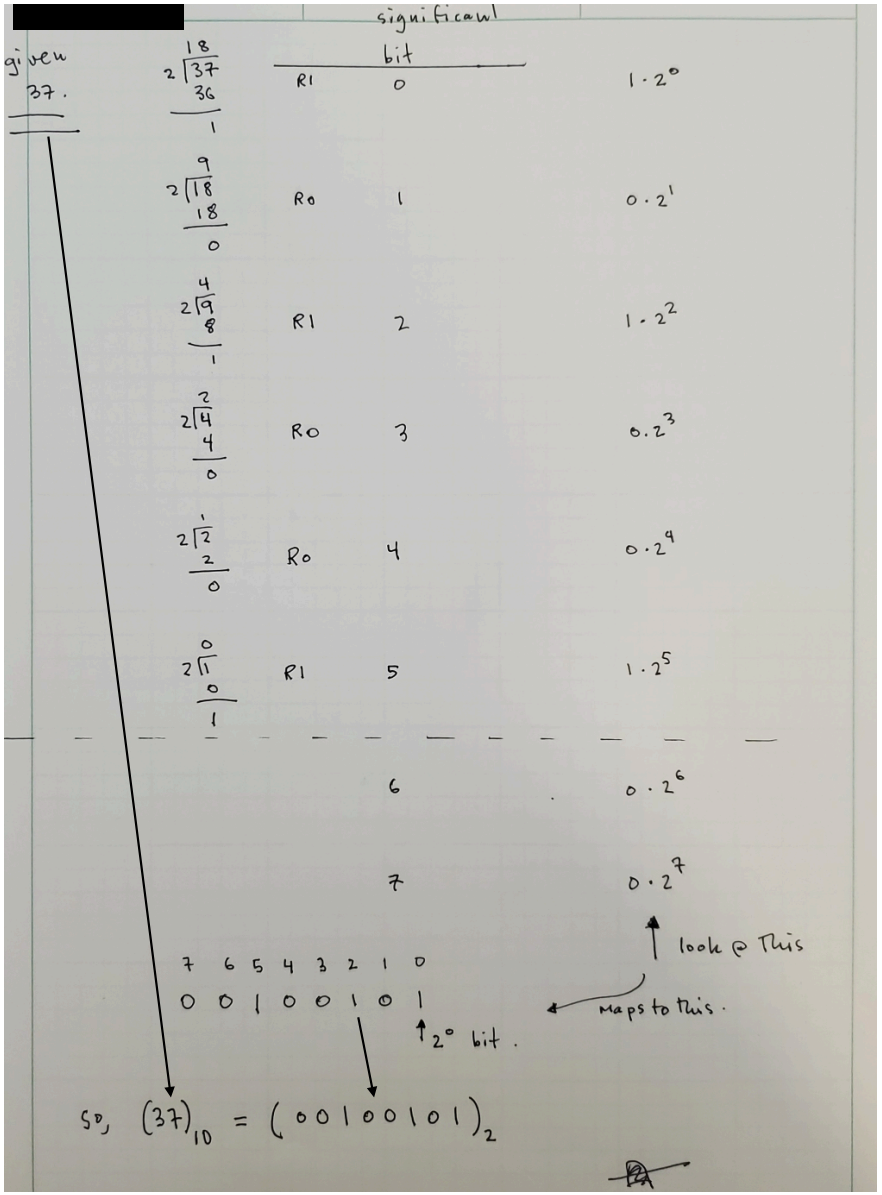
Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

Big endian

0	0	1	0	0	1	0	1
b7	b6	b5	b4	b3	b2	b1	b0

Little endian

1	0	1	0	0	1	0	0
b0	b1	b2	b3	b4	b5	b6	b7



Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

A word on error

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

$$\text{Absolute error} = \text{approximate value} - \text{true value}$$

$$\text{Relative error} = \frac{\text{absolute error}}{\text{true value}}$$

10^{-p} relative error in quantity has ~ *p* correct significant digits in its decimal representation

$$\text{Approximate value} = (\text{true value}) \times (1 + \text{relative error})$$

Do you see any problems here?

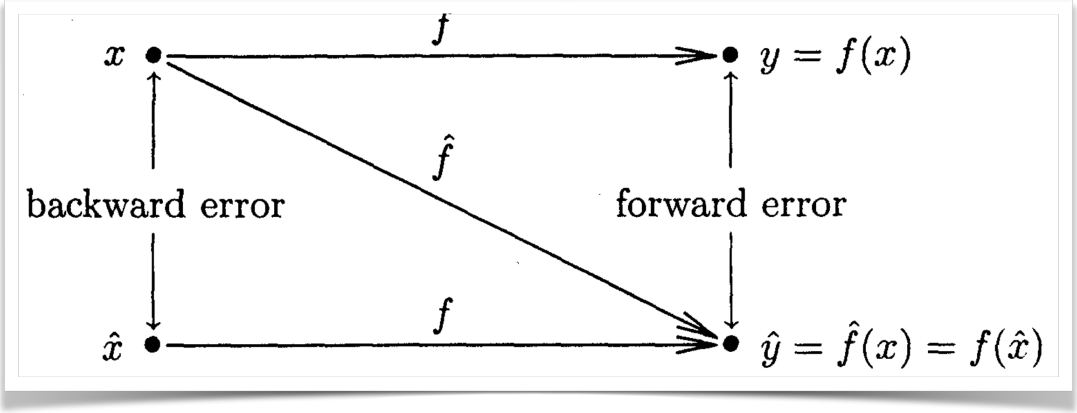
Ans. We don't usually know the *true value* or correct answer if we have committed to computing approximations. However, this is useful for composing tests where known values can be used to test fidelity of implemented algorithm.

$$
\begin{aligned}
\text{Total error} &= \hat{f}(\hat{x}) - f(x) \\
&= (\hat{f}(\hat{x}) - f(\hat{x})) + (f(\hat{x}) - f(x)) \\
&= \text{computational error} + \text{propagated data error}
\end{aligned}
$$

desired true result is $f(x)$

$\hat{f}$ *, is an approximation to the actual function*

$\hat{x}$ *, is an approximation to the correct input*



- Forward error: Measures the error in the computed function value.
- Backward error: Measures how much the input must be changed to explain the computed result.
- Total error: Includes computational error (caused by finite precision) and propagated error (caused by input errors affecting results).
- Numerical stability: An algorithm is stable if its backward error is small—meaning it computes the exact answer to a slightly different problem. (see condition number)

(in)stability

The condition number for a problem indicates the sensitivity of its computed values to perturbations in the input values.

$$\text{Condition number} = \frac{|(f(\hat{x}) - f(x))/f(x)|}{|(\hat{x} - x)/x|} = \frac{|(\hat{y} - y)/y|}{|(\hat{x} - x)/x|} = \frac{|\Delta y/y|}{|\Delta x/x|}$$

- Forward error: Measures the error in the computed function value.
- Backward error: Measures how much the input must be changed to explain the computed result.

$$f(2) = \sqrt{2}$$

The **exact value** (to high precision) is:

$$\sqrt{2} \approx 1.414213562373095$$

However, suppose our computer gives the **approximate value**:

$$\tilde{f}(2) = 1.4$$

Forward and Backward Error. As an approximation to $y = \sqrt{2}$, the value $\hat{y} = 1.4$ has an absolute forward error of

$$|\Delta y| = |\hat{y} - y| = |1.4 - 1.41421\ldots| \approx 0.0142,$$

or a relative forward error of about 1 percent. To determine the backward error we observe that $\sqrt{1.96} = 1.4$, so the absolute backward error is

$$|\Delta x| = |\hat{x} - x| = |1.96 - 2| = 0.04,$$

or a relative backward error of 2 percent.

The **forward error** is:

$$\text{Forward Error} = |\tilde{f}(2) - f(2)|$$

$$= |1.4 - 1.414213562373095|$$

$$\approx 0.01421356$$

This tells us how far off the computed result is from the true square root.

$$\sqrt{\tilde{x}} = 1.4$$

Squaring both sides:

$$\tilde{x} = 1.4^2 = 1.96$$

The **backward error** is:

$$\text{Backward Error} = |x - \tilde{x}|$$

$$= |2 - 1.96| = 0.04$$

The sensitivity of $f(x) = \sqrt{x}$ to input errors is given by its derivative:

$$f'(x) = \frac{1}{2\sqrt{x}}$$

At $x = 2$:

$$f'(2) = \frac{1}{2 \times 1.414} \approx 0.3536$$

If the input has an error $\Delta x = 0.04$, the propagated error in the output is:

$$\text{Propagated Error} \approx 0.3536 \times 0.04 = 0.01414$$

Adding the computational error 0.01421356, the total error is:

$$\text{Total Error} \approx 0.01421356 + 0.01414 = 0.02835$$

Kenneth J. Roche
University of Washington

(in)stability

The condition number for a problem indicates the sensitivity of its computed values to perturbations in the input values.

$$\text{Condition number} = \frac{|(f(\hat{x}) - f(x))/f(x)|}{|(\hat{x} - x)/x|} = \frac{|(\hat{y} - y)/y|}{|(\hat{x} - x)/x|} = \frac{|\Delta y/y|}{|\Delta x/x|}$$

Consider cos(x), cos(pi/2)=0. Let h perturb x around pi/2, cos(x+h).
By definition:

abs_error = cos(x+h) - cos(x) ~ - h sin(x) = -h for x ~ pi/2
rel_error = -h tan(x) = inf (divide by zero)

pi/2 ~ 1.570796327… look at a value close to this and perturb it:

x = 1.57078 , cos(x) = 1.6327e-5
x+h = 1.57079 , cos(x+h) = 6.327e-6

d(input) = 6.366e-6
d(output) = 6.125e-1
d(output)/d(input) ~ 96212.7 !HIGHLY SENSITIVE

IEEE 754 fp data structures

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

floating-point number $x \in \mathbb{F}$

mantissa

exponent

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \beta^E$$

a string of p base-β digits $d_0 d_1 \cdots d_{p-1}$

sign, exponent, mantissa stored in distinct fields

$$0 \le d_i \le \beta - 1, \quad i = 0, \ldots, p-1$$

β	Base or radix
p	Precision
$[L, U]$	Exponent range

$L \le E \le U$

System	β	p	L	U
IEEE SP	2	24	-126	127
IEEE DP	2	53	$-1,022$	$1,023$

Data structure for floating point numbers

IEEE

$V = (-)^S \cdot M \cdot 2^E$

S, sign, $S = 0$, positive
$S = 1$, negative
} sign bit

M, significand or mantissa
$[1, 2-\epsilon)$, or $[0, 1-\epsilon)$
} n bits

$frac\ f = f_{n-1} f_{n-2} \cdots f_0$

encoded value depends if $E = 0$ or not.

E, exponent $= e_{k-1} e_{k-2} \cdots e_1 e_0$
} k bit exponent field

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

SP floating point (float in C or C++)

s, 1-bit
exp, k = 8 bit
frac, n = 23 bit.

} 32 bit number.

DP floating point (double in C or C++)

s, 1-bit
exp, k = 11 bit
frac, n = 52 bit

} 64 bit number.

IEEE

$V = (-)^s \cdot M \cdot 2^E$

s, sign, s = 0, positive; s = 1, negative } sign bit

M, significand or mantissa } n bits
$[1, 2-\epsilon)$, or $[0, 1-\epsilon)$

frac $f = f_{n-1} f_{n-2} \cdots f_0$

encoded value depends if E = 0 or not.

E, exponent $= e_{k-1} e_{k-2} \cdots e_1 e_0$ } k bit exponent field

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

Normalized (most common)

Not all 0s, not all 1s

Normalized,

when exp has not $e_i = 0 \ \forall i$

and not $e_i = 1 \ \forall i$.

$E = e - \text{bias}$, e unsigned # $e_{k-1} \cdots e_1 e_0$

$\text{bias} = 2^{k-1} - 1$

$= 2^{8-1} - 1 = 2^7 - 1 = 127$ SP

$= 2^{11-1} - 1 = 2^{10} - 1 = 1023$ DP

So, $-126 \le E \le 127$ SP

$-1022 \le E \le 1023$ DP

IEEE

$V = (-)^S \cdot M \cdot 2^E$

S, sign, $S = 0$, positive; $S = 1$, negative } sign bit

M, significand or mantissa } n bits

$[1, 2-\epsilon)$, or $[0, 1-\epsilon)$

frac $f = f_{n-1} f_{n-2} \cdots f_0$

encoded value depends if $E = 0$ or not.

E, exponent $= e_{k-1} e_{k-2} \cdots e_1 e_0$ } k bit exponent field

---

frac, $0 \le f < 1$ $0.f_{n-1} f_{n-2} \cdots f_1 f_0$

binary pt to left most significant bit.

$M = 1 + f$

Thus, $V = (-)^S (1 + f) 2^{(e - (2^k - 1))}$

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

IEEE

$V = (-)^S \cdot M \cdot 2^E$

S, sign, $s = 0$, positive; $s = 1$, negative } sign bit

M, significand or mantissa } n bits
$[1, 2-\epsilon)$, or $[0, 1-\epsilon)$

frac $f = f_{n-1} f_{n-2} \cdots f_0$

encoded value depends if $E = 0$ or not.

E, exponent $= e_{k-1} e_{k-2} \cdots e_1 e_0$ } k bit exponent field

**Special case: if e=1s and f!=0 NANs

De Normalized

$e = e_{k-1} \cdots e_1 e_1$ and $e_i = 0 \quad \forall i$

"gradual underflow"

$E = 1 - \text{bias}$

$M = f$

thus $V = (-)^S \cdot f \cdot 2^{1-(2^{k-1}-1)}$

$= (-)^S \cdot f \cdot 2^{2-2^{k-1}}$

ie represent 0

$-0.0 \neq +0.0$

different sign bit!

if $e_i = 1 \; \forall i$ and $M=0$ $\rightarrow \pm \infty$ cases.

Denormalized

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

IEEE

$V = (-)^s \cdot M \cdot 2^E$

s, sign, $s = 0$, positive; $s = 1$, negative } sign bit

M, significand or mantissa } n bits

$[1, 2-\epsilon)$, or $[0, 1-\epsilon)$

frac $f = f_{n-1} f_{n-2} \cdots f_0$

encoded value depends if $E = 0$ or not.

E, exponent $= e_{k-1} e_{k-2} \cdots e_1 e_0$ } k bit exponent field

Smallest Normalized

1 in least significant bit, 0s in all other bits of exponent field.

$M = 1$

$E = -2^{k-1} + 2$

$V = 2^{-2^{k-1}+2}$

Largest Normalized

sign bit 0

least significant exp. bit 0, the rest 1s

$f = 1 - 2^{-n}$ $\qquad E = 2^{k-1} - 1$

$M = 2 - 2^{-n}$

$V = (2 - 2^{-n}) \cdot 2^{2^{k-1}-1}$

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

IEEE

$V = (-)^S \cdot M \cdot 2^E$

S, sign, S = 0, positive; S = 1, negative } sign bit

M, significand or mantissa } n bits

$[1, 2-\epsilon)$, or $[0, 1-\epsilon)$

frac $f = f_{n-1} f_{n-2} \cdots f_0$

encoded value depends if E = 0 or not.

E, exponent $= e_{k-1} e_{k-2} \cdots e_1 e_0$ } k bit exponent field

Smallest DeNormalized

1 in least significant bit, 0's otherwise

$M = f = 2^{-n}$

$E = -2^{k-1} + 2$

$V = 2^{-n} - 2^{k-1} + 2$

Largest DeNormalized

Exponent all 0s

frac, all 1s

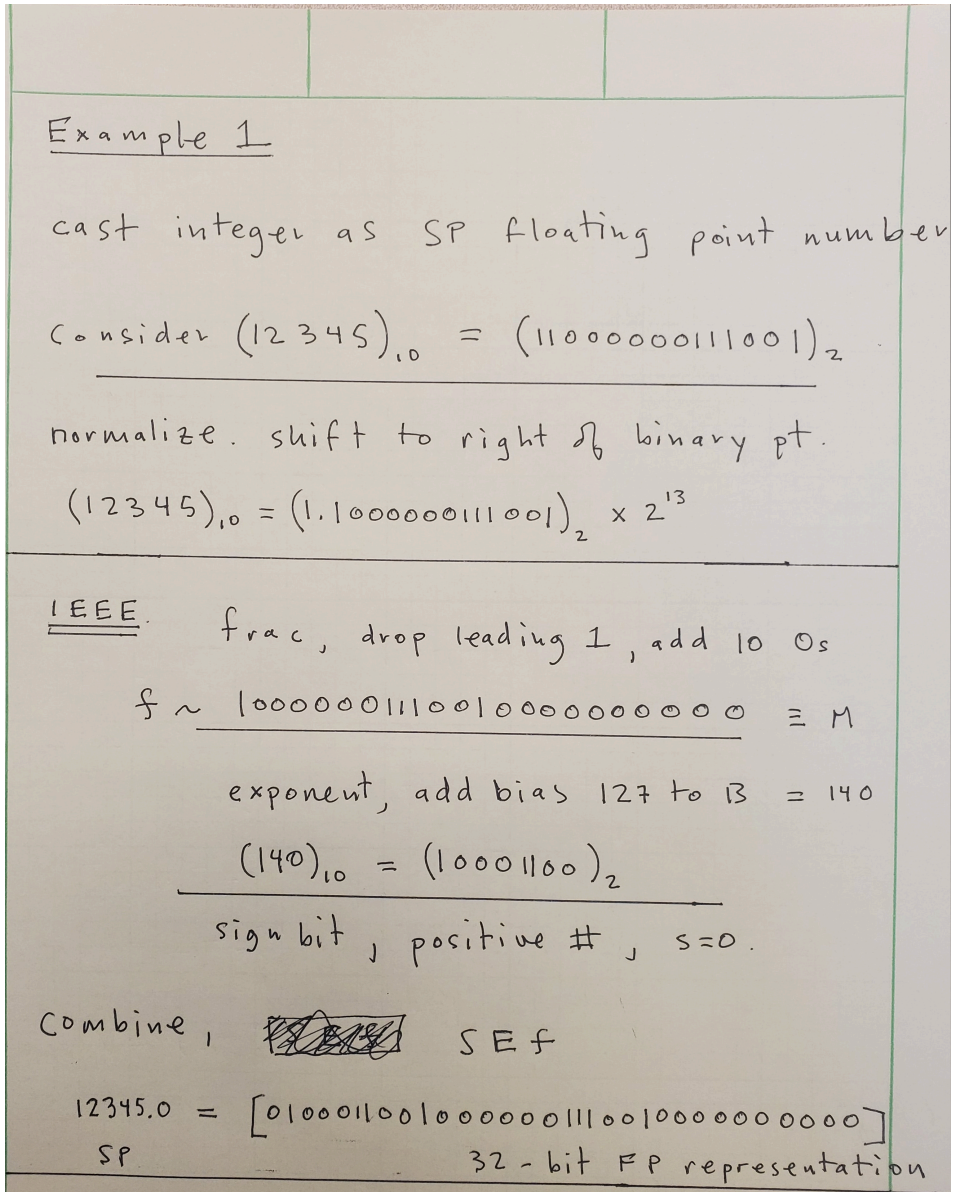
$M = f = 1 - 2^{-n}$

$E = -2^{k-1} + 2$

$V = (1 - 2^{-n}) \cdot 2^{-2^{k-1} + 2}$

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

casting example



Example 1

cast integer as SP floating point number

Consider $(12345)_{10} = (11000000111001)_2$

normalize. shift to right of binary pt.

$(12345)_{10} = (1.10000001110001)_2 \times 2^{13}$

IEEE. frac, drop leading 1, add 10 0s

$f \sim 10000001110010000000000 = M$

exponent, add bias 127 to 13 $= 140$

$(140)_{10} = (10001100)_2$

sign bit, positive #, $s=0$.

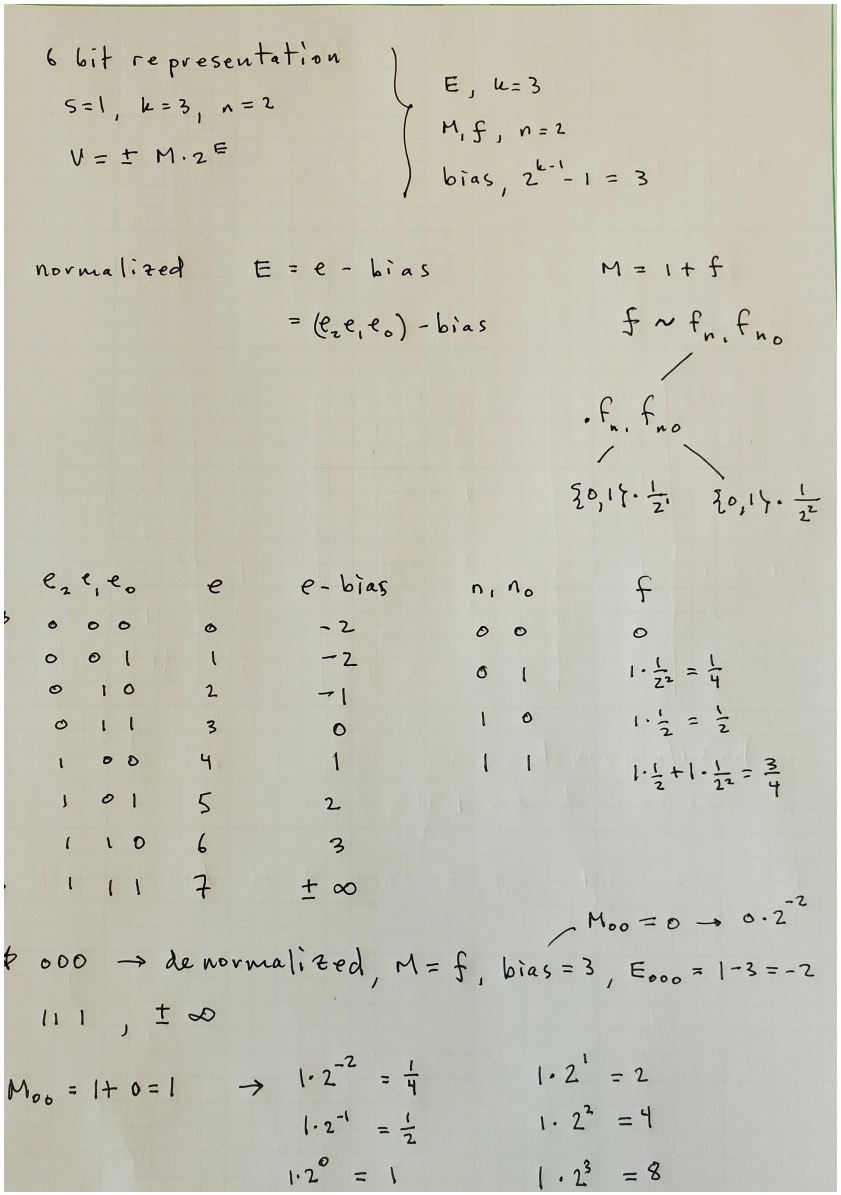
Combine, [redacted] SEf

12345.0 = [01000110010000001110010000000000]

SP 32-bit FP representation

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

toy data structure for HW
(just getting you started here)



Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

End Lecture 3 (end week 1)

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025