INHA UNIVERSITY TASHKENT
SPRING SEMESTER 2024

SOC 2040
SYSTEM PROGRAMMING

CREDITS/HOURS PER WEEK : 3/3

COURSE TYPE : TECHNICAL CORE SEQUENCE

INSTRUCTOR
DR. A. R. NASEER
HEAD & PROFESSOR OF COMPUTER SCIENCE & ENGG

SOC 2040
SYSTEM PROGRAMMING

WEEK 5 LECTURE 1

DATA REPRESENTATION

Floating Point Arithmetic

Real Number Representation

- Fixed-point Representation

- Floating-point Representation

Real Number Representation

- **Fixed-point Representation**

- In this representation, a fixed binary point is assumed

- This format allows the representation of numbers with a fractional part

- Limitations
  - Very large number cannot be represented, nor can very small fractions
  - The fractional part of the quotient in a division of two large numbers could be lost

Real Number Representation

- **Floating-point Representation**

- Similar to scientific notation
- The binary point is dynamically moved to a convenient location and the exponent is used to keep track of that binary point
- This allows a range of very large and very small numbers to be represented with only a few bits.

Example : - 1000.11001 x 2^6

Sign of Mantissa

Mantissa (Significand)

True Exponent

IEEE Standard for Floating point Representation

- **IEEE Standard 754**
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs

- Driven by numerical concerns
  - Nice standards for rounding, overflow, underflow
  - Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining standard

IEEE Standard for Floating point Representation

- **IEEE Standard 754**

- Floating point representation is defined in IEEE Standard 754

- IEEE Standard defines

  - a 32-bit Single precision format

  - a 64-bit Double precision format

  - a 80-bit Extended precision format

IEEE Standard for Floating point Representation

❑ Precision options

- Single precision: 32 bits

s	exp	frac
1	8-bits	23-bits

- Double precision: 64 bits

s	exp	frac
1	11-bits	52-bits

- Extended precision: 80 bits (Intel only)

s	exp	frac
1	15-bits	64-bits

IEEE 754 Single Precision format

- Single precision representation uses a 32-bit format
- The format consists of three fields
  - **Sign bit field – 1 bit Most significant bit used to indicate the sign of the mantissa**
  - **Biased Exponent field – next 8 bits used to store the biased exponent (true exponent + bias of value 127)**
  - **Mantissa field – last 23 bits used to store the fractional mantissa**

    - Mantissa is expressed in Normalized form

    - The first bit of the mantissa is always 1 and need not be stored in the mantissa field

Example : -1000.11001 x 2^6 = -1 . 00011001 x 2^9

after Normalization

Fractional mantissa

IEEE 754 Single Precision format

- **Single precision representation uses a 32-bit format**

Sign bit

Biased Exponent

Fractional Mantissa

S	E	F
1 bit	**8 bits**	**23 bits**
31	30 … 23	22 … 0

$N = (-1)^S \ 1.F \times 2^{E-B}$

where B : Bias = +127

Sign Bit S = 1, negative mantissa

= 0, positive mantissa

True exponent e, Biased Exponent E = e + Bias = e + 127

True exponent e is an 8-bit signed integer with values ranging from
-128, -127, -126, …………………..…-1, 0, 1, ……………………….,+126, +127

Biased Exponent E is an unsigned integer obtained by adding +127 to e
+255, 0, +1, ………………….+126,+127,+128,………………., +253, +254

Reserved for Special Combinations

IEEE 754 Single Precision format

- **Single precision format parameters**

Word width	32 bits
Exponent width	8 bits
Mantissa width	23 bits
Exponent Bias	127
Maximum Exponent	127
Minimum Exponent	-126
Number of exponents	254
Number of fractions	2^{23}
Number range	10^{-38} to 10^{+38}

IEEE 754 Single Precision format

- **Express the following in Single precision format :**

- N = **1100.110110 x 2^10^**

After Normalization,

$N = 1.100110110 \times 2^{13}$

S	Exp E	Fractional Mantissa F
1	*8-bits*	*23-bits*

S =0, F = 100110110

True exponent, e = 13

E = e + B = 13 + 127 = 140 = 10001100

In single precision form :

N = 0 10001100 10011011000000000000000

= 464D8000 H (packed form)

IEEE 754 Single Precision format

- **Express the following in Single precision format :**

➢ **$N = -\ 0.000100110110 \times 2^{-22}$**

After Normalization,

$N = -\ 1.00110110 \times 2^{-26}$

S	*Exp E*	*Fractional Mantissa F*
1	*8-bits*	*23-bits*

S =1, F = 00110110

True exponent, e = -26

E = e + B = -26 + 127 = 101 = 01100101

In single precision form :

N = 1 01100101 00110110000000000000000

= B29B0000 H (packed form)

IEEE 754 Double Precision format

- **Double precision representation uses a 64-bit format**
- The format consists of three fields
  - **Sign bit S field – 1 bit Most significant bit used to indicate the sign of the mantissa**
  - **Biased Exponent E field – next 11 bits used to store the biased exponent (true exponent + bias of value 1023)**
  - **Mantissa F field – last 52 bits used to store the fractional mantissa**

    - Mantissa is expressed in Normalized form

    - The first bit of the mantissa is always 1 and need not be stored in the mantissa field

IEEE 754 Double Precision format

- **Double precision representation uses a 64-bit format**

Sign bit

Biased Exponent

Fractional Mantissa

S	E	F
1 bit	11 bits	52 bits

63 62 52 51 0

Sign Bit = 1, negative mantissa

= 0, positive mantissa

$$N = (-1)^{S}\ 1.F \times 2^{E-B}$$

Where B : Bias = +1023

IEEE 754 Double Precision format

Double precision format parameters

Word width	64 bits
Exponent width	11 bits
Mantissa width	52 bits
Exponent Bias	1023
Maximum Exponent	1023
Minimum Exponent	-1022
Number of exponents	2046
Number of fractions	2^{52}
Number range	10^{-308} to 10^{+308}

IEEE Standard for Floating point Representation

- **ZERO**

- An exponent of zero together with a fraction of zero represents positive or negative zero, depending on the sign bit

0 00000000 00000000000000000000000 = +0.0

1 00000000 00000000000000000000000 = -0.0

- **INFINITY**

- An exponent of all ones together with a fraction of zero represents positive or negative infinity, depending on the sign bit.

0 11111111 00000000000000000000000 = + ∞

1 11111111 00000000000000000000000 = - ∞

- **Denormalized Number**

- An exponent of zero together with a nonzero fraction represents denormalized number. In this case, the bit to the left of the binary point is zero (0.F) and the bias is 126 (single precision) or 1022 (double precision) i.e., true exponent is E-126 or E-1022. The number is positive or negative depending on the sign bit

  0 00000000 10000000000000000000000 = $+0.1 \times 2^{-126}$

  1 00000000 10000000000000000000000 = -0.1×2^{-126}

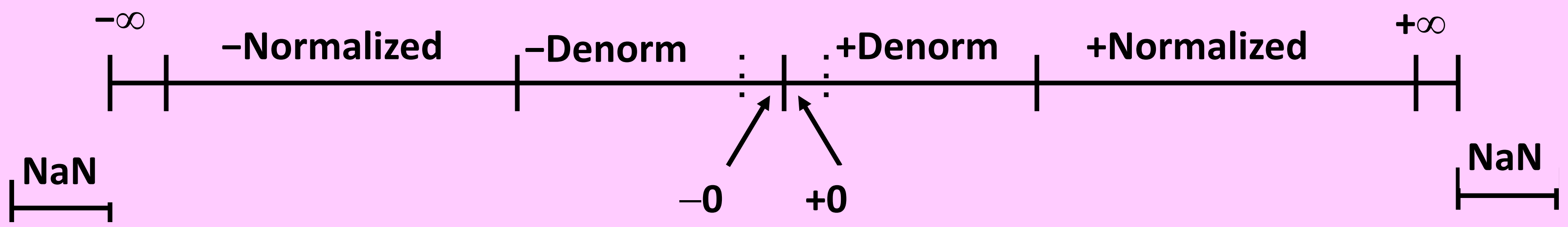
- **Not A Number (NaN)**

- An exponent of all ones together with a nonzero fraction NaN, which means not a number and is used to signal various exception conditions. .

  0 11111111 10000000000000000000000 = NaN

  1 11111111 10000000000000000000000 = NaN

Visualization: Floating Point Encodings



Floating Point Representation

Question :

Convert the following decimal numbers into the IEEE format for single precision floating point numbers and express each in packed form :

a) N1 = 121.6875

b) N2 = 23.895

c) N3 = - 39.525

Floating Point Representation

Solution :

a) N1 = **121.6875**

= 1111001.1011

Normalizing,

$N1 = 1.1110011011 \times 2^6$

S1 = 0, F1 = 11100110110000000000000, e1 = 6

E1 = e1 + 127 = 6 + 127 = 133 = **10000101**

In packed form,

N1 = 0**10000101**11100110110000000000000

= 0x42F36000

Floating Point Representation

Solution :

b) N2 = **23.895**

= 10111.11100101000111101011

Normalizing,

N2 = $1.01111111001010001111010 1 \times 2^4$

S2 = 0, F2 = 01111111001010001111010 1, e2 = 4

E2 = e2 + 127 = 4 + 127 = 131 = **10000011**

In packed form,

N2 = 0**10000011**01111111001010001111010 1

= 0x41BF28F5

Floating Point Representation

Solution :

c) N3 = **- 39.525**

= - 100111.10000110011001 1001

Normalizing,

N3= $-1.00111100001100110011001 \times 2^5$

S3 = 1, F3 = 00111100001100110011001 , e3 = 5

E3 = e3 + 127 = 5 + 127 = 132 = **10000100**

In packed form,

N3 = 1**10000100**00111100001100110011001

= 0xC21E1999

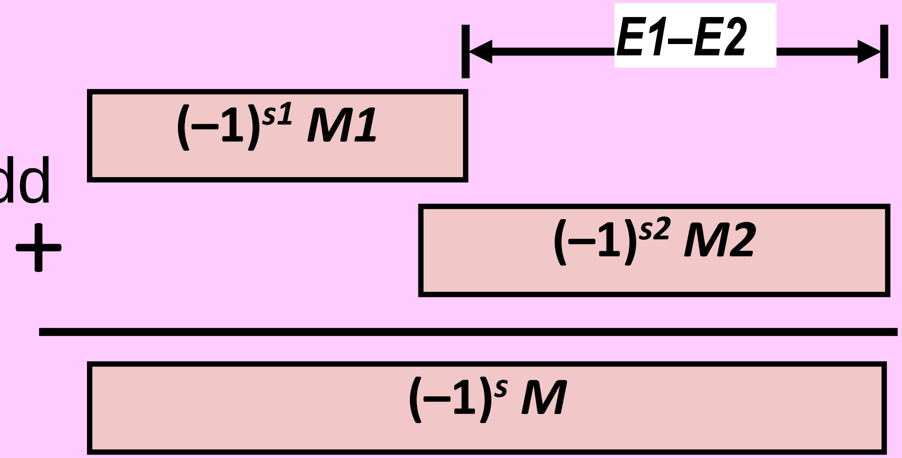
Floating Point Addition

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
  - Assume $E1 > E2$, Make E2=E1, by aligning M2

    Then Perform M=M1+M2

Subtraction : M=M1-M2

Get binary points lined up



- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign s, significand M:
    - Result of signed align & add
  - Exponent E: $E1$
- Fixing
  - If $M \geq 2$, shift M right, increment E
  - if $M < 1$, shift M left k positions, decrement E by k
  - Overflow if E out of range
  - Round M to fit **`frac`** precision

FP Multiplication

- $(-1)^{s1}\, \boldsymbol{M1}\ \ 2^{\boldsymbol{E1}} \ \ \times \ \ (-1)^{\boldsymbol{s2}}\, \boldsymbol{M2}\ \ 2^{\boldsymbol{E2}}$
- Exact Result: $(-1)^{s}\, \boldsymbol{M}\ \ 2^{\boldsymbol{E}}$
  - Sign *s*: *s1 ^ s2*
  - Significand *M*: *M1* x *M2*
  - Exponent *E*: *E1* + *E2 - Bias*

FP DIVISION

$(-1)^{s1}\, \boldsymbol{M1}\ \ 2^{\boldsymbol{E1}} \ \ / \ \ (-1)^{\boldsymbol{s2}}\, \boldsymbol{M2}\ \ 2^{\boldsymbol{E2}}$

- Sign *s*: *s1 ^ s2*
- Significand *M*: *M1 / M2*
- Exponent *E*: *E1 - E2 + Bias*

- Fixing
  - If *M* ≥ 2, shift *M* right, increment *E*
  - If *E* out of range, overflow
  - Round *M* to fit **`frac`** precision
- Implementation
  - Biggest chore is multiplying significands

Floating Point Operations: Basic Idea

- $\mathtt{x} +_{\mathtt{f}} \mathtt{y} = \mathtt{Round(x + y)}$
- $\mathtt{x} \times_{\mathtt{f}} \mathtt{y} = \mathtt{Round(x} \times \mathtt{y)}$
- Basic idea
  - First compute exact result
  - Make it fit into desired precision
    - Possibly overflow if exponent too large
    - Possibly round to fit into **`frac`**

Rounding

- Rounding Modes (illustrate with $ rounding)

	\$1.40	\$1.60	\$1.50	\$2.50	–\$1.50
- Round to Nearest Even	\$1	\$2	\$2	\$2	–\$2
– Round Towards zero	\$1	\$1	\$1	\$2	–\$1
– Round down (−∞)	\$1	\$1	\$1	\$2	–\$2
– Round up (+∞)	\$2	\$2	\$2	\$3	–\$1

Closer Look at Round-To-Even

- **Default Rounding Mode**
  - Hard to get any other kind without dropping into assembly
  - All others are statistically biased
    - Sum of set of positive numbers will consistently be over- or under-estimated
- **Applying to Other Decimal Places / Bit Positions**
  - When exactly halfway between two possible values
    - Round so that least significant digit is even
  - E.g., round to nearest hundredth

7.8949999	7.89	(Less than half way)
7.8950001	7.90	(Greater than half way)
7.8950000	7.90	(Half way—round up)
7.8850000	7.88	(Half way—round down)

Rounding Binary Numbers

- **Binary Fractional Numbers**
  - "Even" when least significant bit is `0`
  - "Half way" when bits to right of rounding position = `100`…$_2$

- **Examples**
  - Round to nearest 1/4 (2 bits right of binary point)

Value	Binary	Rounded	Action	Rounded Value
2 3/32	10.00011_2	10.00_2	(<1/2—down)	2
2 3/16	10.00110_2	10.01_2	(>1/2—up)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—up)	3
2 5/8	10.10100_2	10.10_2	(1/2—down)	2 1/2

Floating Point ARITHMETIC

Question : Given N1 & N2 and Perform the following operation and express the result in packed hex form :

(i) **N1 + N2**

N1 = 736.8365 = 1.01110000011010110001001001 x 2^9

N2 = - 985.78357 = - 1.1110110011100100010011000 x 2^9

Exponents are equal

Since magnitude of N2 > N1, subtract N1 from N2 and put sign of N2 for the result

N1 + N2 = N1 + (– N2) = - (N2 – N1)

= 1.11101 1001 1100 1000 1001 1000 x 2^9

1.01110 0000 1101 0110 0010 0100 x 2^9

- 0.01111 1000 1111 0010 0111 0100 x 2^9

Renormalizing,

N1 + N2 = = - 1.1111000111100100111 0100 x 2^7

S1 = 1, F1 = 11110001111001001110100, e1 = 7

E1 = e1 + 127 = 7 + 127 = 134 = 10000110

In packed form,

N1 + N2 = 11000011011110001111001001110100

= 0xC378F274

Floating Point ARITHMETIC

Question : **Given N1 & N3, perform the following operation and express the result in packed hex form :**

(ii) N1 + N3

N1 = 736.8365 = 1.01110000001101011000100100 x 2^9

N3 = 4678.768 = 1.00100100011011000100100011011 x 2^{12}

Make the exponents equal – shift the bits in N1 to the right by 3 bits

N1 = + 0.00101110000001101011000100100 x 2^{12}

Since both N1 & N3 are of the same sign, perform addition

N1 + N3 = 0.00101110000001101011000100100 x 2^{12}

1.00100100011011000100100011011 x 2^{12}

1.01010010011110011010101111111 x 2^{12}

No Normalization required,

N1 + N3 = 1.01010010011110011010101111111 x 2^{12}

S1 = 0, F1 01010010011110011010101111111, e1 = 12

E1 = e1 + 127 =12 + 127 = 139 = 10001011

In packed form,

N1 + N3 = 0 10001011 01010010011110011010101111111

= 0x45A93CD5F = or 0x45A93CD6 (after rounding)

Floating Point ARITHMETIC

Question : Given N2 & N4, perform the following operation and express the result in packed hex form :

(iii) N2 + N4

N2 = - 985.78357 = - 1.11101100111001000100110000 x 2^{9}

N4= -85789.9841875 = - 1.01001111000111011111101111110011x 2^{16}

Make the exponents equal – shift the bits in N2 to the right by 7 bits

N2 = -0.000000111101100111001000100110000 x 2^{16}

Since both N2 & N4 are of the same sign, perform addition

N2 + N4 = - 0.0000 0011 1101 1001 1100 1000 1001 1000 x 2^{16}

- 1.0100 1111 0001 1101 1111 1011 1111 0011 x 2^{16}

-1.0101 0010 1111 0111 1100 0100 1000 1011 x 2^{16}

No Normalization required,

N2 + N4 = -1.0101 0010 1111 0111 1100 0100 1000 1011 x 2^{16}

S1 = 1, F1 = 0101 0010 1111 0111 1100 0100 1000 1011, e1 = 16

E1 = e1 + 127 =16 + 127 = 143 = 10001111

In packed form,

N2 + N4 = 1 10001111 01010010111101111100010 01000101

= 0xC7A97BE245 or 0xC7A97BE2 (after rounding)

Floating Point Representation

Question : **Given N2 & N3, perform the following operation and express the result in packed hex form :**

(iv) N2 - N3

N2 = - 985.78357 = - 1.11101100111001000100110000 x 2^9

N3 = 4678.768 = 1.00100100011011000100100110011 x 2^{12}

Make the exponents equal – shift the bits in N2 to the right by 3 bits

N2 = - 0.00111101100111001000100110000 x 2^{12}

Since N2 – N3 = (-N2) + (-N3), perform addition

N2 + N3 = - 0.0011 1101 1001 1100 1000 1001 1000 x 2^{12}

- 1.0010 0100 0110 1100 0100 1001 1011 x 2^{12}

- 1.0110 0010 0000 1000 1101 0011 0011 x 2^{12}

No Renormalization

N2 - N3 = - 1.0110 0010 0000 1000 1101 0011 0011 x 2^{12}

S1 = 1, F1 = 0110 0010 0000 1000 1101 0011 0011, e1 = 12

E1 = e1 + 127 =12 + 127 = 139 = 10001011

In packed form,

N2 – N3 = 1 10001011 0110001000001000110100 1 1001

= 0xC5B104699 = 0xC5B1046A(after rounding)

Floating Point ARITHMETIC

Question :

- Given N1 & N2 and Perform the following operations and express the results in packed hex form :

(i) N1 + N2

N1 = 736.8365 = $1.01110000011010110001001 00 \times 2^9$

N2 = - 985.78357 = $- 1.11101100111001000100110 00 \times 2^9$

Exponents are equal

Since magnitude of N2 > N1, subtract N1 from N2 and put sign of N2 for the result

N1 + N2 = N1 + (– N2) = - (N2 – N1)

= $1.11101\ 1001\ 1100\ 1000\ 1001\ 1000 \times 2^9$

$\underline{1.01110\ 0000\ 1101\ 0110\ 0010\ 0100 \times 2^9}$

$\underline{-\ 0.01111\ 1000\ 1111\ 0010\ 0111\ 0100 \times 2^9}$

Renormalizing,

N1 + N2 = = $\underline{-\ 1.11110001111001001110100 \times 2^7}$

S1 = 1, F1 = 11110001111001001110100, e1 = 7

E1 = e1 + 127 = 7 + 127 = 134 = 10000110

In packed form,

N1 + N2 = 11000011011110001111001001110100

= 0xC378F274

Floating Point ARITHMETIC

Question :

(ii) N1 + N3

N1 = 736.8365 = 1.01110000001101011000100100 x 2^9

N3 = 4678.768 = 1.00100100011011000100100110 11 x 2^{12}

Make the exponents equal – shift the bits in N1 to the right by 3 bits

N1 = + 0.00101110000001101011000100100 x 2^{12}

Since both N1 & N3 are of the same sign, perform addition

N1 + N3 = 0.00101110000001101011000100100 x 2^{12}

1.00100100011011000100100110 11 x 2^{12}

1.01010010011110011010101111 11 x 2^{12}

No Normalization required,

N1 + N3 = 1.0101001001111001101010111111 x 2^{12}

S1 = 0, F1 0101001001111001101010111111, e1 = 12

E1 = e1 + 127 =12 + 127 = 139 = 10001011

In packed form,

N1 + N3 = 0 10001011 0101001001111001101010111111

= 0x45A93CD5F = or 0x45A93CD6 (after rounding)

Floating Point ARITHMETIC

Question :

(iii) N2 + N4

N2 = - 985.78357 = $- 1.11101100111001000100110000 \times 2^{9}$

N4= -85789.9841875 = $- 1.01001111000111011111101111110011 \times 2^{16}$

Make the exponents equal – shift the bits in N2 to the right by 7 bits

N2 = $-0.0000001111011001110010001001100000 \times 2^{16}$

Since both N2 & N4 are of the same sign, perform addition

N2 + N4 = $- 0.0000\ 0011\ 1101\ 1001\ 1100\ 1000\ 1001\ 1000 \times 2^{16}$

$- \underline{1.0100\ 1111\ 0001\ 1101\ 1111\ 1011\ 1111\ 0011 \times 2^{16}}$

$\underline{-1.0101\ 0010\ 1111\ 0111\ 1100\ 0100\ 1000\ 1011 \times 2^{16}}$

No Normalization required,

N2 + N4 = $-1.0101\ 0010\ 1111\ 0111\ 1100\ 0100\ 1000\ 1011 \times 2^{16}$

S1 = 1, F1 = 0101 0010 1111 0111 1100 0100 1000 1011, e1 = 16

E1 = e1 + 127 =16 + 127 = 143 = 10001111

In packed form,

N2 + N4 = 1 10001111 01010010111101111100010 01000101

= 0xC7A97BE245 or 0xC7A97BE2 (after rounding)

Floating Point Representation

Question :

(iv) N2 - N3

N2 = - 985.78357 = - 1.11101100111001000100110000 x 2^9

N3 = 4678.768 = 1.00100100011011000100100110111 x 2^{12}

Make the exponents equal – shift the bits in N2 to the right by 3 bits

N2 = - 0.00111101100111001000100110000 x 2^{12}

Since N2 – N3 = (-N2) + (-N3), perform addition

N2 + N3 = 0.0011 1101 1001 1100 1000 1001 1000 x 2^{12}

1.0010 0100 0110 1100 0100 1001 1011 x 2^{12}

- 1.0110 0010 0000 1000 1101 0011 0011 x 2^{12}

No Renormalization

N2 - N3 = - 1.0110 0010 0000 1000 1101 0011 0011 x 2^{12}

S1 = 1, F1 = 0110 0010 0000 1000 1101 0011 0011, e1 = 12

E1 = e1 + 127 =12 + 127 = 139 = 10001011

In packed form,

N2 – N3 = 1 10001011 01100010000010001101001 1001

= 0xC5B104699 = 0xC5B1046A(after rounding)

Mathematical Properties of FP Add

- Compare to those of Abelian Group
  - Closed under addition? *Yes*
    - But may generate infinity or NaN
  - Commutative? *Yes*
  - Associative? *No*
    - Overflow and inexactness of rounding
    - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
  - 0 is additive identity? *Yes*
  - Every element has additive inverse? *Almost*
    - Yes, except for infinities & NaNs
- Monotonicity
  - $a \geq b \Rightarrow a+c \geq b+c$? *Almost*
    - Except for infinities & NaNs

Mathematical Properties of FP Mult

- Compare to Commutative Ring
  - Closed under multiplication? ***Yes***
    - But may generate infinity or NaN
  - Multiplication Commutative? ***Yes***
  - Multiplication is Associative? ***No***
    - Possibility of overflow, inexactness of rounding
    - Ex: `(1e20*1e20)*1e-20= inf`, `1e20*(1e20*1e-20)= 1e20`
  - 1 is multiplicative identity? ***Yes***
  - Multiplication distributes over addition? ***No***
    - Possibility of overflow, inexactness of rounding
    - `1e20*(1e20-1e20)= 0.0`, `1e20*1e20 – 1e20*1e20 = NaN`
- Monotonicity
  - $a \geq b \;\; \& \; c \geq 0 \;\; \Rightarrow a * c \geq b * c$? ***Almost***
    - Except for infinities & NaNs

Floating Point in C

- **C Guarantees Two Levels**
  - `float` single precision
  - `double` double precision

- **Conversions/Casting**
  - Casting between `int`, `float`, and `double` changes bit representation
  - `double`/`float` → `int`
    - Truncates fractional part
    - Like rounding toward zero
    - Not defined when out of range or NaN: Generally sets to TMin
  - `int` → `double`
    - Exact conversion, as long as `int` has ≤ 53 bit word size
  - `int` → `float`
    - Will round according to rounding mode

Floating Point Puzzles

- For each of the following C expressions, either:
  - Argue that it is true for all argument values
  - Explain why not true

```
int x = …;
float f = …;
double d = …;
```

Assume neither **d** nor **f** is NaN

- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (double)(float) d`
- `f == -(-f);`
- `2/3 == 2/3.0`
- `d < 0.0    ⇒    ((d*2) < 0.0)`
- `d > f      ⇒    -f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

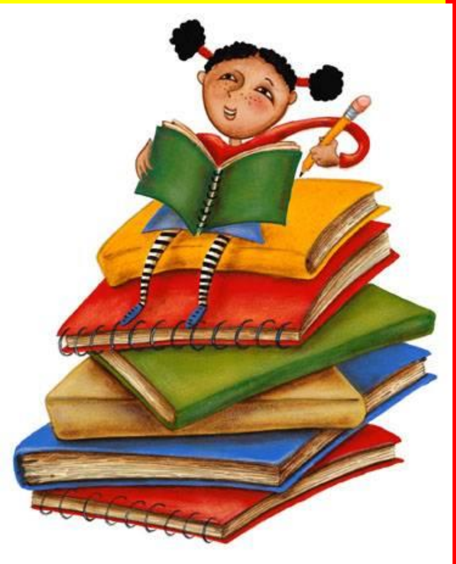
Summary

- IEEE Floating Point has clear mathematical properties
- Represents numbers of form M x 2^E
- One can reason about operations independent of implementation
  - As if computed with perfect precision and then rounded
- Not the same as real arithmetic
  - Violates associativity/distributivity
  - Makes life difficult for compilers & serious numerical applications programmers

TRY OUT ALL PRACTICE PROBLEMS

CHAPTER 2 - Practice Problems

- 2.45(page 102 Solution page 148)
- 2.46(page 102 Solution page 148)
- 2.47(page 107 Solution page 148)
- 2.48(page 110 Solution page 149)
- 2.49(page 110 Solution page 149)
- 2.50(page 112 Solution page 150)
- 2.51(page 112 Solution page 150)
- 2.52(page 112 Solution page 150)
- 2.53(page 115 Solution page 150)
- 2.54(page 117 Solution page 151)



SOC 2040 SYSTEM PROGRAMMING

Reading Assignments

Chapter 2 of Text Book

- **Computer Systems : A Programmer's Perspective**

  - Randal E. Bryant and David R. O'Hallaron
  3rd Edition, Global Edition Prentice Hall 2016
  ISBN 10:1-292-10176-8

SOC 2040
SYSTEM PROGRAMMING

END OF

WEEK 5 LECTURE 1

DATA REPRESENTATION

WISH YOU ALL THE BEST

INHA UNIVERSITY TASHKENT
FALL SEMESTER 2024

SOC 2040
SYSTEM PROGRAMMING

CREDITS/HOURS PER WEEK : 3/3

COURSE TYPE : TECHNICAL CORE SEQUENCE

INSTRUCTOR
DR. A. R. NASEER
HEAD & PROFESSOR OF COMPUTER SCIENCE & ENGG