Numbers with Fractions

Reference:

- David Money Harris and Sarah L. Harris. Digital Design and Computer Architecture, 2nd Edition. Elsevier – Morgan Kaufmann, 2013.

Numbers with Fractions

- Two common notations:
  - **Fixed-point:** binary point fixed
  - **Floating-point:** binary point floats to the right of the most significant 1

Fixed-Point Numbers

- 6.75 using 4 integer bits and 4 fraction bits:

01101100

0110.1100

$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6.75$

  - Binary point is implied
  - The number of integer and fraction bits must be agreed upon beforehand

Fixed-Point Number Example

- Represent 7.5_{10} using 4 integer bits and 4 fraction bits.

Fixed-Point Number Example

- Represent 7.5_{10} using 4 integer bits and 4 fraction bits.

01111000

Floating-Point Numbers

- Binary point floats to the right of the most significant 1
- Similar to decimal scientific notation

- For example, write 273_{10} in scientific notation:

$$\mathbf{273 = 2.73 \times 10^2}$$

- In general, a number is written in scientific notation as:

$$\mathbf{\pm M \times B^E}$$

  - **M** = mantissa
  - **B** = base
  - **E** = exponent
  - In the example, M = 2.73, B = 10, and E = 2

Floating-Point Numbers

1 bit	8 bits	23 bits
Sign	**Exponent**	**Mantissa**

- **Example:** represent the value 228_{10} using a 32-bit floating point representation

We show three versions –final version is called the **IEEE 754 floating-point standard**

Floating-Point Representation 1

1. Convert decimal to binary (**don't reverse steps 1 & 2!**):

$$228_{10} = 11100100_2$$

2. Write the number in "binary scientific notation":

$$11100100_2 = 1.11001_2 \times 2^7$$

3. Fill in each field of the 32-bit floating point number:
   - The sign bit is positive (0)
   - The 8 exponent bits represent the value 7
   - The remaining 23 bits are the mantissa

1 bit	8 bits	23 bits
0	00000111	11 1001 0000 0000 0000 0000
Sign	**Exponent**	**Mantissa**

Floating-Point Representation 2

- First bit of the mantissa is always 1:
  - $228_{10} = 11100100_2 = \mathbf{1.11001} \times 2^7$
- So, no need to store it: *implicit leading 1*
- Store just fraction bits in 23-bit field

1 bit	8 bits	23 bits
0	00000111	110 0100 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

ELSEVIER

Excess Representation

- Besides sign-and-magnitude and complement schemes, the **excess representation** is another scheme.
- It allows the range of values to be distributed evenly between the positive and negative values, by a simple translation (addition/subtraction).
- Example: Excess-4 representation on 3-bit numbers. See table on the right.

Excess-4 Representation	*Value*
000	-4
001	-3
010	-2
011	-1
100	0
101	1
110	2
111	3

Floating-Point Representation 3

- bias = 0.5radix^{s} – 1 = r^{s-1} – 1 (s adalah jumlah bit exponent)
- *Biased exponent*: bias = 127 (01111111_2)
  - **Biased exponent = bias + exponent**
  - Exponent of 7 is stored as:

$$127 + 7 = 134 = 0x10000110_2$$

- The **IEEE 754 32-bit floating-point representation** of 228_{10}

1 bit	8 bits	23 bits
0	10000110	110 0100 0000 0000 0000 0000
Sign	**Biased Exponent**	**Fraction**

in hexadecimal: **0x43640000**

Floating-Point Example

Write -58.25_{10} in floating point (IEEE 754)

Floating-Point Example

Write -58.25_{10} in floating point (IEEE 754)

1. Convert decimal to binary:

   $58.25_{10} = \mathbf{111010.01_2}$

2. Write in binary scientific notation:

   $\mathbf{1.1101001 \times 2^5}$

3. Fill in fields:

   Sign bit: 1 (negative)
   8 biased exponent bits: (127 + 5) = 132 = $\mathbf{10000100_2}$
   23 fraction bits: **110 1001 0000 0000 0000 0000**

1 bit	8 bits	23 bits
1	100 0010 0	110 1001 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

in hexadecimal: **0xC2690000**

Exercise

1. Nyatakan bilangan desimal (+78.75) dalam bentuk IEEE-754 single-precision floating-point. Tulis jawaban anda dalam hexadecimal.

2. Nyatakan bilangan IEEE-754 single-precision floating-point format **0x C2220000** sebagai bilangan desimal

Floating-Point: Special Cases

Number	Sign	Exponent	Fraction
0	X	00000000	00000000000000000000000
∞	0	11111111	00000000000000000000000
- ∞	1	11111111	00000000000000000000000
NaN	X	11111111	non-zero

Floating-Point Precision

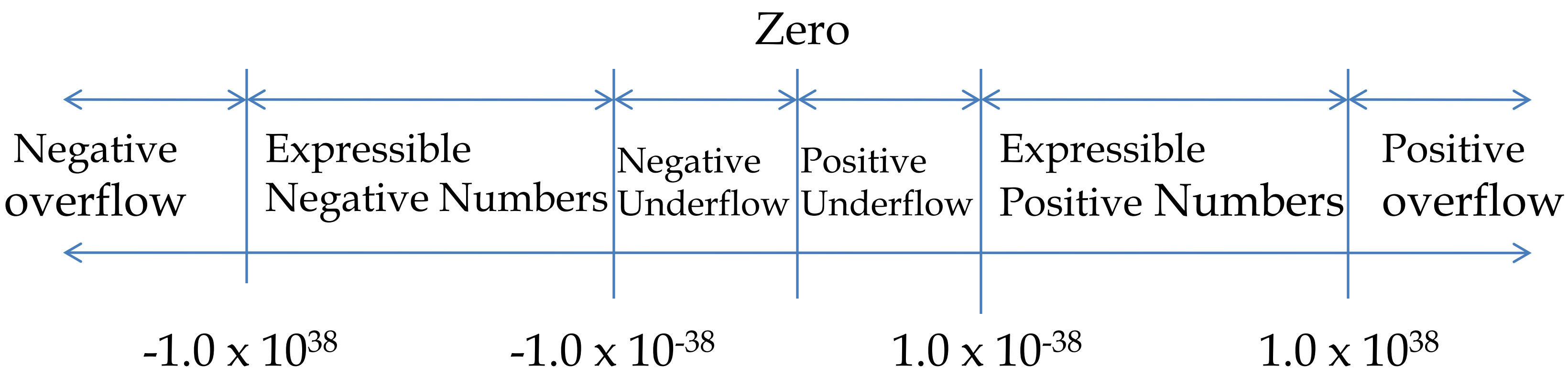
- **Single-Precision:**
  - 32-bit
  - 1 sign bit, 8 exponent bits, 23 fraction bits
  - bias = 127
- **Double-Precision:**
  - 64-bit
  - 1 sign bit, 11 exponent bits, 52 fraction bits
  - bias = 1023

IEEE Standard 754 Floating Point Numbers

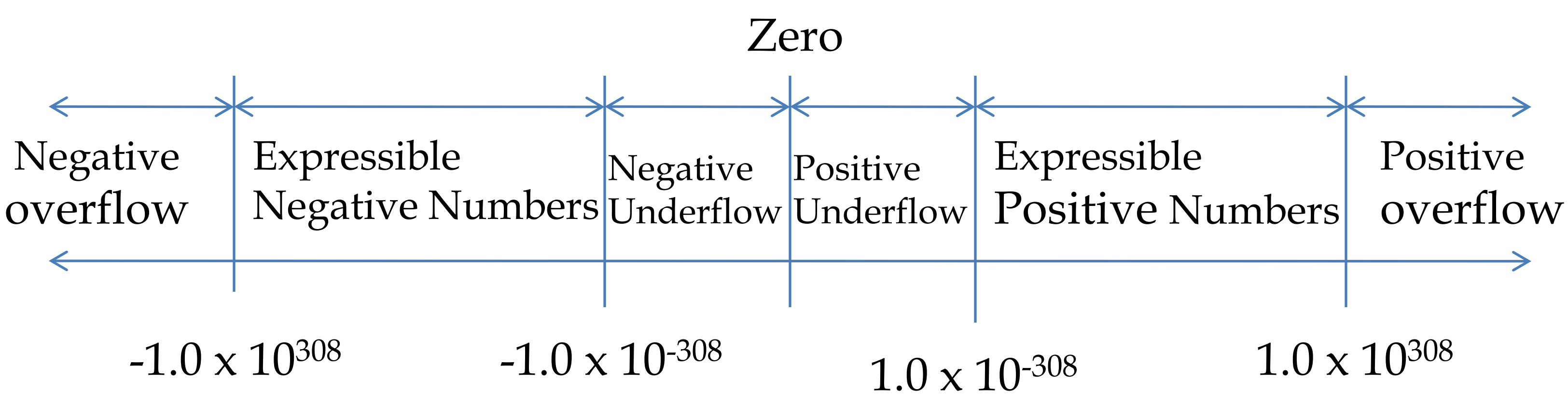
The range of single precision numbers:

- As small as: $\pm 1.0000\ 0000\ 0000\ 0000\ 0000\ 000_2 \times 2^{-126}$
- As large as: $\pm 1.\ 1111\ 1111\ 1111\ 1111\ 1111\ 111_2 \times 2^{+127}$

	Denormalized	Normalized	Approximate Decimal
Single Precision	$\pm 2^{-149}$ to $(1-2^{-23})\times 2^{-126}$	$\pm 2^{-126}$ to $(2-2^{-23})\times 2^{127}$	$\pm \sim 10^{-44.85}$ to $\sim 10^{38.53}$
Double Precision	$\pm 2^{-1074}$ to $(1-2^{-52})\times 2^{-1022}$	$\pm 2^{-1022}$ to $(2-2^{-52})\times 2^{1023}$	$\pm \sim 10^{-323.3}$ to $\sim 10^{308.3}$



Range of IEEE-754 Single-precision Numbers



Range of IEEE-754 Double-Precision Numbers

Floating-Point: Rounding

- **Overflow:** number too large to be represented
- **Underflow:** number too small to be represented
- **Rounding modes:**
  - Down
  - Up
  - Toward zero
  - To nearest
- **Example:** round 1.100101 (1.578125) to only 3 fraction bits
  - Down: 1.100
  - Up: 1.101
  - Toward zero: 1.100
  - To nearest: 1.101 (1.625 is closer to 1.578125 than 1.5 is)

Floating-Point Addition

1. Extract exponent and fraction bits
2. Prepend leading 1 to form mantissa
3. Compare exponents
4. Shift smaller mantissa if necessary
5. Add mantissas
6. Normalize mantissa and adjust exponent if necessary
7. Round result
8. Assemble exponent and fraction back into floating-point format

Floating-Point Addition Example

Add the following floating-point numbers:

0x3FC00000

0x40500000

Floating-Point Addition Example

1. **Extract exponent and fraction bits**

1 bit	8 bits	23 bits
0	01111111	100 0000 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

1 bit	8 bits	23 bits
0	10000000	101 0000 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

For first number (N1): S = 0, E = 127, F = .1

For second number (N2): S = 0, E = 128, F = .101

2. **Prepend leading 1 to form mantissa**

N1: 1.1

N2: 1.101

Floating-Point Addition Example

3. **Compare exponents**

   127 – 128 = -1, so shift N1 right by 1 bit

4. **Shift smaller mantissa if necessary**

   shift N1's mantissa: 1.1 >> 1 = 0.11 $(\times 2^1)$

5. **Add mantissas**

$$
\begin{array}{r}
0.11 \times 2^1 \\
+\ 1.101 \times 2^1 \\
\hline
10.011 \times 2^1
\end{array}
$$

Floating Point Addition Example

6. **Normalize mantissa and adjust exponent if necessary**

   $10.011 \times 2^1 = 1.0011 \times 2^2$

7. **Round result**

   No need (fits in 23 bits)

8. **Assemble exponent and fraction back into floating-point format**

   $S = 0$, $E = 2 + 127 = 129 = 10000001_2$, $F = 001100..$

1 bit	8 bits	23 bits
0	10000001	001 1000 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

in hexadecimal: **0x40980000**

ELSEVIER

Floating-Point Subtraction

Subtract the following floating-point numbers:

0x3FC00000

0x40500000

FP Multiplication Example

Multiply the following floating-point numbers:

0x3FC00000

0x40500000

Floating-Point Division Example

Divide the following floating-point numbers:

0x3FC00000

0x40500000