Chapter 03: Computer Arithmetic

Lesson 08:

Floating-Point Numbers (IEEE754 Standard) and Operations

Objective

- Understand use of Floating Point Numbers and operations

Floating Point Numbers

Floating-point numbers

- Used to represent quantities that cannot be represented by integers, either because they contain fractional values or because they lie outside the range representable within the system's bit width

IEEE 754 Format

- Virtually all modern computers use the floating-point representation specified in IEEE standard 754
- Numbers represented by a mantissa and an exponent

Mantissa and Exponent

- Similar to scientific notation, the value of a floating-point number is

$$\underline{mantissa} \times \underline{2^{exponent}}$$

IEEE 754 Floating-Point format Single Precision

Sign	Exponent	Fraction
1	8	23

$\longleftarrow$ 32-bit $\longrightarrow$

IEEE 754 Floating-Point format Double Precision

Sign	Exponent	Fraction
1	11	52

$\longleftarrow$ 64-bit $\longrightarrow$

Fractional Field

- The fraction field— a sign-magnitude number
- Represents the fractional portion of a binary number whose integer portion is assumed to be 1

Mantissa of an IEEE 754 floating number

- ±(1. *fraction*)
- Plus or minus— Depending on the value of the sign bit
- Using an assumed "leading 1" in this way
- Increases the number of significant digits that can be represented by a floating-point number of a given width

Exponent field

- Uses a biased integer representation
- Fixed bias added to a value to determine its representation

Exponent field For single-precision floating-point numbers

- The bias = 127_d
- Value of the exponent field— found by subtracting 127 from the unsigned binary number contained in the field
- Bias = 1023_d for double precision numbers
- Value of the exponent field— found by subtracting 1023

Normal case: Fields and IEEE 754 Representation

- Exponent Field not 0 but all 1s
- Fraction Field   Any
- Representation

$\pm\,(1.\text{fraction}) \times 2^{(\text{exponent} - \text{bias})}$ depending on sign bit

Exponent field for double precision numbers

- Bias = 1023_d
- Value of the exponent field— found by subtracting 1023

Special case: Fields and IEEE 754 Representation

- Exponent Field 0
- Fraction Field 0
- Representation

  0

Special case: Fields and IEEE 754 Representation

- Exponent Field 0
- Fraction Field not 0
- Representation

$\pm\,(0.\text{fraction}) \times 2^{(1-\text{bias})}$ depending on sign bit

Special case: Fields and IEEE 754 Representation

- Exponent Field All 1s
- Fraction Field 0
- Representation

± infinity depending on sign bit

Special case: Fields and IEEE 754 Representation

- Exponent Field All 1s
- Fraction Field   not 0
- Representation

  NaN

Examples of Representation of Floating Point Numbers

Mantissa of a floating number 6.25

- Fractional binary numbers use the same place-value representation as decimal numbers, with a base of 2, so the binary number 0b11.111=$2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = 3.875$
- Using this format, a decimal fraction converted to a binary fraction directly
- $6.25 = 2^1 + 2^0 + 2^{-1} + 2^{-2} + =$ 0b110.01

Floating-point representation of 6.25

- To find the fraction field in IEEE format, we shift the binary representation of the number down so that the value to the left of the binary point is 1, so 0b110.01 becomes 0b1.1001 $\times\ 2^2$
- Sign = 0, Exponent = 2, Mantissa = 1001

Normalized fraction representation

- The leading 1 is assumed, and only the values to the right of the binary point (1001 in this case) are represented
- Extending the value to the 23-bit fraction format of single-precision floating-point, we get
- 1001 0000 0000 0000 0000 000 as the fraction field

Example of Exponent of a floating number

- –45 be represented in the 8-bit biased notation used in the exponents of single-precision numbers
- The bias value for this format 127
- Add 127 to number to get the biased representation
- –45 + 127 = 82 = 0b01010010

Example of Exponent of a floating number

- 123 be represented in the 8-bit biased notation used in the exponents of single-precision numbers
- The bias value for this format 127
- Add 127 to number to get the biased representation
- 123+127 = 250 =0b11111010

Example of exponent field of 0b11100010

- 8-bits used thus single-precision floating-point number
- 0b11100010 = 226
- 226 – 127 = 99, so the exponent field has a value of 99
- 2^{99}

Single-precision floating-point number 0b0l00 0000 0110 0000 0000 0000 0000 0000

- Dividing this number into the fields S, E and M fields
- Sign bit = 0
- Exponent field of bl0000000 = 128
- Fraction field of 0b1100 0000 0000 0000 0000 000

Solution —exponent field

- Subtracting the bias of 127 from the exponent field gives an exponent = 1

  2^1

Solution —Mantissa field

- $.11_2 = .75$
- Include the implied 1 to the left of the binary point in the fraction field
- The mantissa is $1.11_2 = 1.75$

Answer

- The value of the floating-point number is $+ 1.75 \times 2^1 = 3.5$

Rounding

Floating-point number representation

- Allows a wide range of values to be represented in a relatively small number of bits, including both fractional values and values whose magnitude is much too large to represent in an integer with the same number of bits

Floating-point number representation

- Creates the problem that many of the values in the range of the floating-point representation cannot be represented exactly, just as many of the real numbers cannot be represented by a decimal number with a fixed number of significant digits

IEEE 754 standard rounding mode

- When a computation creates a value that cannot be represented exactly by the floating-point format, the hardware must round the result to a value that can be represented exactly
- Standard *default rounding mode* is round-to-nearest

IEEE 754 standard rounding mode

- In round-to-nearest, values are rounded to the closest representable number, and results that lie exactly halfway between two representable numbers are rounded such that the least-significant digit of their result is even

IEEE 754 standard rounding mode

- The standard specifies several other rounding modes that can be selected by programs, including round toward 0, round toward +infinity, and round toward –infinity

Denormalized numbers

Denormalized Numbers

- Are in contrast to numbers with other values of the exponent field, which have an assumed 1 in the integer portion of their mantissa and which are known as normalized numbers

Assumed 1 in the mantissa of floating-point numbers

- It allows an additional bit of precision in the representation
- But prevents the value 0 from being represented exactly, since a fraction field of 0 represents the mantissa 1.0

Floating-point number with an exponent field = 0

- The numbers (except for 0) that have an exponent field of 0 are known as *denormalized* numbers because of the assumed 0 in the integer portion of their mantissa

Representing 0

- Since representing 0 exactly is very important for numerical computations, the IEEE standard specifies that, when the exponent field of a floating-point number is 0, the leading bit of the mantissa is assumed to be 0

Floating-point number with an exponent field = 0

- When fraction field of = 0, it represents 0 exactly
- It allows numbers that lie closer to 0 than $1.0 \times 2^{(1-\text{bias})}$ to be represented

Numbers closer to 0

- Although they have fewer bits of precision than numbers that can be represented with an assumed 1 before the fraction field

Denormalized numbers

- All denormalized numbers are assumed to have an exponent field of (1-bias), instead of the (0-bias) that would be generated by just subtracting the bias from the value of their exponent.

Denormalized numbers

- All denormalized numbers are assumed to have an exponent field of (1-bias), instead of the (0-bias) that would be generated by just subtracting the bias from the value of their exponent

A gap

- A smaller gap between the smallest-magnitude normalized number and the largest-magnitude denormalized number that can be represented by a format

NaN

Floating-point standard NaNs

- NaN (Not a number)
- NaNs used to signal error conditions such as overflows, underflows, division by 0, and so on
- When one of these error conditions occur in an operation, the hardware generates a NaN as its result rather than signaling an exception

Floating-point standard NaNs

- Subsequent operations that receive a NaN as one of their inputs copy that NaN to their outputs rather than performing their normal computation

Floating-point standard NaNs

- NaNs are indicated by all 1s in the exponent field of a floating-point number, unless the fraction field of the number is 0, in which case the number represents infinity

Existence of NaNs

- Makes it easier to write programs that run on multiple different computers, because programmers can check the results of each computation for errors within the program
- Rather than relying on the system's exception-handling functions, which vary significantly between different computers

Summary

We learnt

- Floating point IEEE 754 Representation
- Representation of 0, + infinity, – infinity, number very close to zero (denormalized numbers) and NaNs

End of Lesson 08 on

Floating-Point Numbers (IEEE754 Standard) and Operations