Floating-Point Binary

Lectures 15
Section 3.5

Robb T. Koether

Hampden-Sydney College

Mon, Sep 30, 2019

Outline

Outline

Binary Fractions

Binary Fractions

$$0.a_{-1}a_{-2}a_{-3}\ldots = a_{-1}\times 2^{-1} + a_{-2}\times 2^{-2} + a_{-3}\times 2^{-3} + \cdots$$

- Positions to the right of the “binary” point represent negative powers of 2.

Binary Fractions

Binary Fractions

$$
\begin{aligned}
0.110101_2 &= 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} + 0 \times 2^{-5} + 1 \times 2^{-6} \\
&= \frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} + \frac{0}{32} + \frac{1}{64} \\
&= \frac{53}{64} \\
&= 0.828125_{10}.
\end{aligned}
$$

- Write each fraction and add up their values.

Binary Fractions

Binary Fractions

$$\begin{aligned} 0.110101_2 &= 110101_2 \times 2^{-6} \\ &= 53_{10} \times 2^{-6} \\ &= \frac{53}{64}. \end{aligned}$$

- Or shift the binary point and compute the numerator as a binary integer.

Outline

Binary Fractions

- The algorithm to convert from a decimal fraction to a binary fraction is similar to the algorithm to convert from a decimal integer to a binary integer, except reversed.
- Multiply the binary fractional part repeatedly by 2.
- After each multiplication, the integer part is the binary digit.
- Continue with the fractional part.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.828125 \times 2 = 1.65625$$

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.828125 \times 2 = 1.65625$$
$$0.65625 \times 2 = 1.3125$$

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.828125 \times 2 = 1.65625$$
$$0.65625 \times 2 = 1.3125$$
$$0.3125 \times 2 = 0.625$$

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.828125 \times 2 = 1.65625$$
$$0.65625 \times 2 = 1.3125$$
$$0.3125 \times 2 = 0.625$$
$$0.625 \times 2 = 1.25$$

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.828125 \times 2 = 1.65625$$
$$0.65625 \times 2 = 1.3125$$
$$0.3125 \times 2 = 0.625$$
$$0.625 \times 2 = 1.25$$
$$0.25 \times 2 = 0.5$$

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.828125 \times 2 = 1.65625$$
$$0.65625 \times 2 = 1.3125$$
$$0.3125 \times 2 = 0.625$$
$$0.625 \times 2 = 1.25$$
$$0.25 \times 2 = 0.5$$
$$0.5 \times 2 = 1.0$$

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.828125 \times 2 = 1.65625$$
$$0.65625 \times 2 = 1.3125$$
$$0.3125 \times 2 = 0.625$$
$$0.625 \times 2 = 1.25$$
$$0.25 \times 2 = 0.5$$
$$0.5 \times 2 = 1.0$$
$$0.828125_{10} = 0.110101_2.$$

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.1 \times 2 = 0.2$$

- Terminating expansions in decimal are not necessarily terminating in binary.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.1 \times 2 = 0.2$$
$$0.2 \times 2 = 0.4$$

- Terminating expansions in decimal are not necessarily terminating in binary.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.1 \times 2 = 0.2$$
$$0.2 \times 2 = 0.4$$
$$0.4 \times 2 = 0.8$$

- Terminating expansions in decimal are not necessarily terminating in binary.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.1 \times 2 = 0.2$$
$$0.2 \times 2 = 0.4$$
$$0.4 \times 2 = 0.8$$
$$0.8 \times 2 = 1.6$$

- Terminating expansions in decimal are not necessarily terminating in binary.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.1 \times 2 = 0.2$$
$$0.2 \times 2 = 0.4$$
$$0.4 \times 2 = 0.8$$
$$0.8 \times 2 = 1.6$$
$$0.6 \times 2 = 1.2$$

- Terminating expansions in decimal are not necessarily terminating in binary.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$
\begin{aligned}
0.1 \times 2 &= 0.2 \\
0.2 \times 2 &= 0.4 \\
0.4 \times 2 &= 0.8 \\
0.8 \times 2 &= 1.6 \\
0.6 \times 2 &= 1.2 \\
0.2 \times 2 &= 0.4 \\
&\vdots
\end{aligned}
$$

- Terminating expansions in decimal are not necessarily terminating in binary.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$
\begin{aligned}
0.1 \times 2 &= 0.2 \\
0.2 \times 2 &= 0.4 \\
0.4 \times 2 &= 0.8 \\
0.8 \times 2 &= 1.6 \\
0.6 \times 2 &= 1.2 \\
0.2 \times 2 &= 0.4 \\
&\vdots \\
0.1_{10} &= 0.0001100110011_2 \ldots .
\end{aligned}
$$

- Terminating expansions in decimal are not necessarily terminating in binary.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.333\ldots \times 2 = 0.666\ldots$$

- Non-terminating expansions in decimal are always non-terminating in binary.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.333\ldots \times 2 = 0.666\ldots$$
$$0.666\ldots \times 2 = 1.333\ldots$$

- Non-terminating expansions in decimal are always non-terminating in binary.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$0.333\ldots \times 2 = 0.666\ldots$$
$$0.666\ldots \times 2 = 1.333\ldots$$
$$0.333\ldots \times 2 = 0.666\ldots$$
$$\vdots$$

- Non-terminating expansions in decimal are always non-terminating in binary.

Decimal-to-Binary Conversions

Decimal-to-Binary Conversion

$$
\begin{aligned}
0.333\ldots \times 2 &= 0.666\ldots \\
0.666\ldots \times 2 &= 1.333\ldots \\
0.333\ldots \times 2 &= 0.666\ldots \\
&\vdots \\
0.333_{10} &= 0.01010101_2\ldots.
\end{aligned}
$$

- Non-terminating expansions in decimal are always non-terminating in binary.

Outline

Storing Floating-Point Numbers

- Floating-point numbers are stored in the binary equivalent of scientific notation.
- For example, 1101_2 would be stored as

$$1101_2 = 1.101 \times 2^3.$$

- The IEEE 754 standard for single precision specifies
  - 1 bit for the sign.
  - 8 bits for the exponent.
  - 23 bits for the mantissa.

Storing the Mantissa

- Every binary number except 0 begins with the digit 1.
- Knowing this, there is no need to store the leading 1.
- Thus, the leading 1 is assumed and the rest of the mantissa is stored, giving us 24-bit precision.

Storing the Mantissa

- Every binary number except 0 begins with the digit 1.
- Knowing this, there is no need to store the leading 1.
- Thus, the leading 1 is assumed and the rest of the mantissa is stored, giving us 24-bit precision.
- So how is `1.0000...` distinguished from `0.0000...`?

Storing the Mantissa

- Every binary number except 0 begins with the digit 1.
- Knowing this, there is no need to store the leading 1.
- Thus, the leading 1 is assumed and the rest of the mantissa is stored, giving us 24-bit precision.
- So how is `1.0000...` distinguished from `0.0000...`?
- The number 0 is represented by a special pattern (later).

Storing the Exponent

- In order to represent positive numbers less than 1, we must be able to store negative exponents.
- Using two's complement to represent negative exponents creates unnecessary complications.
- Therefore, we store a biased exponent:

$$\text{stored exponent} = \text{true exponent} + 127.$$

Storing the Exponent

- Thus, stored exponents of 0 through 255 *would* represent true exponents of -127 through $+128$.

Storing the Exponent

- Thus, stored exponents of 0 through 255 *would* represent true exponents of -127 through $+128$.
- However, the stored exponents of 0 and 255 themselves are used to represent special quantities.

Storing the Exponent

- Thus, stored exponents of 0 through 255 *would* represent true exponents of -127 through $+128$.
- However, the stored exponents of 0 and 255 themselves are used to represent special quantities.
- The stored exponent of 0 signifies the number 0.0.

Storing the Exponent

- Thus, stored exponents of 0 through 255 *would* represent true exponents of -127 through $+128$.
- However, the stored exponents of 0 and 255 themselves are used to represent special quantities.
- The stored exponent of 0 signifies the number 0.0.
- The stored exponent of 255 is used to represent $\pm\infty$ and NaN.

Storing the Exponent

- Thus, stored exponents of 0 through 255 *would* represent true exponents of -127 through $+128$.
- However, the stored exponents of 0 and 255 themselves are used to represent special quantities.
- The stored exponent of 0 signifies the number 0.0.
- The stored exponent of 255 is used to represent $\pm\infty$ and NaN.
- The remaining values 1 through 254 are used for the exponents -126 through $+127$ of non-zero numbers.

Minimum and Maximum Values

- The smallest possible positive single-precision floating point number is

$$+1.00000000000000000000000_2 \times 2^{-126} \approx 1.1754945 \times 10^{-38}.$$

- The largest possible positive single-precision floating point number is

$$+1.11111111111111111111111_2 \times 2^{+127} \approx 3.4028235 \times 10^{38}.$$

Special Values

Value	Displayed	Sign	Exponent	Mantissa
0	`0`	0/1	0	0
$+\infty$	`inf`	0	255	0
$-\infty$	`-inf`	1	255	0
invalid	`nan`	0/1	255	$\neq 0$

- IEEE 754 reserves specific patterns for 0, infinity, and NaN.

Floating-point Limits

- Run the program `FloatLimitTest.cpp`.

Outline

Assignment

Assignment

- Read Section 3.5.