Floating Point Representation

COMP370

Introduction to Computer Architecture

Binary Fractions

- Each position is twice the value of the position to the right.

2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
8	4	2	1	.	1/2	1/4	1/8
1	1	1	0	.	0	1	0

Adding the powers of 2 gives
8+4+2+0.25 = 14.25

What is 111.11 in decimal?

1. 7.75
2. 31
3. 7.375
4. 15.25

What is 8.5 in binary?

1. 11111111.11111
2. 1000.01
3. 0.100011
4. 1000.10

Range of Values

- Unsigned integers: 0 to 2^n-1
  - For byte, from 0 to 255
  - For int, from 0 to 4.2×10^9
- 2's complement: $-(2^{n-1})$ to $+(2^{n-1}-1)$
  - For byte, from -128 to 127
  - For short, from -32768 to 32767
  - For int, from -2147483648 to 2147483647
  - For long, from $-9{,}2 \times 10^{18}$ to $9{,}2 \times 10^{18}$

Scientific Notation

$$241{,}506{,}800 = \underbrace{0.2415068}_{\boxed{\text{Mantissa}}} \times 10^{\overset{\boxed{\text{Exponent}}}{9}}$$

Shifting Exponents

- 241,506,800 can be

  2.415068×10^8

  24.15068×10^7

  241.5068×10^6

  2415.068×10^5

  24150.68×10^4

  241506.8×10^3

  etc.

Binary Scientific Notation

- A binary number, such as 10110011, can be expressed as:

$$1.0110011 \times 2^7$$

- Note the exponent is a power of two not ten.

Shifting Binary Exponents

- A binary number can be expressed in "scientific" notation is several ways like decimal numbers.

0.110010×2^5	$0.78125 \times 32 = 25$
1.10010×2^4	$1.5625 \times 16 = 25$
11.0010×2^3	$3.125 \times 8 = 25$
110.010×2^2	$6.25 \times 4 = 25$
1100.10×2^1	$12.5 \times 2 = 25$
11001.0×2^0	$25 \times 1 = 25$

110.010 is equivalent to

1. 11001.0×2^{-2}
2. 0.110010×2^3
3. 6.25
4. All of the above
5. None of the above

Standard Format

- Most computers (including Intel Pentiums) follow the *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Standard 754-1985
- Before the standard different computers used different formats for floating point numbers.
- The standard defines the format, accuracy and action taken when errors are detected.

Floating-point Sizes

- ANS/IEEE Standard 754-1985
  - Single-precision (32 bits)
  - Double-precision (64 bits)
  - Extended-precision (80 bits)

Single-Precision Floating-point Numbers

- **`float`** variables in C++ or Java

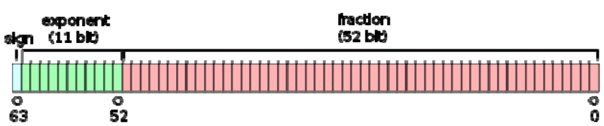
Signed Magnitude

- For positive numbers, the sign bit is zero
- For negative numbers, the sign bit is one and everything else is the same

Single Precision Float Range

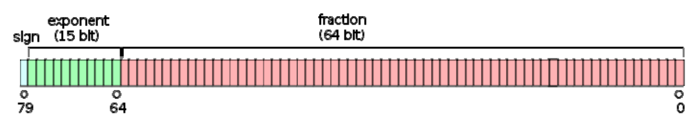
- A little more than 7 decimal digits accuracy
- From -3.4×10^{38} to 3.4×10^{38}
- Positive numbers can be as small as 1.18×10^{-38} before going to zero.

Double Precision Floating Point



- **`double`** variables in C++ or Java
- approximately 16 decimal digits of accuracy
- From -1.798×10^{308} to 1.798×10^{308}

Extended Precision Floating Point



- almost 20 decimal digits of accuracy
- 3.37×10^{-4932} to 1.18×10^{4932}
- Not directly supported in C++ or Java
- Often used internally for calculations which are then rounded to desired precision

Exponent Bias

- The exponent represents the power of 2.
- The single precision exponent is biased by adding 127 to the actual exponent
- This avoids an extra sign bit for the exponent
- The exponent range is -126 to 128

Exponent value	Decimal exponent	Binary exponent
2^0	127	01111111
2^5	132	10000100
2^{-5}	122	01111010

Normalization

- Floating point numbers are adjusted so the mantissa or fractional part has a single “1” bit before the radix point.

Decimal	Binary	Normalized
5.75	101.11×2^0	1.0111×2^2
0.125	0.001×2^0	1.0×2^{-3}
32.0	100000.0×2^0	1.0×2^5

Saving a Bit

- The fractional part or mantissa is always adjusted so the leftmost bit is a one.
- Since this bit is **always** a one, it is not actually stored in the floating point number.
- The mantissa is stored without the leading one bit although the one bit is assumed in calculating the value of the number.

Creating a Floating Point Number

1. Write the number in binary with a fractional part as necessary.
2. Adjust the exponent so the radix point is to the right of the first one bit.
3. The mantissa is the binary number without the leading one bit.
4. The exponent field is created by adding 127 to the binary exponent.
5. The sign is the same as the number’s sign.

Decimal to Floating Point Example

- Convert 4.5 to single precision floating point
- Decimal 4.5 is 100.1 in binary
- Adjust radix to get 1.001 x 2^2
- The exponent field is 127+2 =129 = 10000001
- The floating point number in binary is

S	Exponent	Mantissa
0	10000001	00100000000000000000000

Convert 15.375 to Floating Point

Convert 15.375 to Floating Point

- Decimal 15.375 is 1111.011 in binary
- Adjust the exponent to 1.111011 x 2^3
- Exponent field is 3 + 127 = 130_{10} = 10000010_2

S	Exponent	Mantissa
0	10000010	11101100000000000000000

Floating Point to Decimal Example

S	Exponent	Mantissa
1	10000011	01001000000000000000000

- What is the decimal value of this number?
- Exponent 10000011 = 131 – 127 = 4
- Mantissa is 1.01001000000000000000000
- -1.01001 x 2^4 = -10100.1
- -10100.1 is -20.5 in decimal

What is the decimal value of

S	Exponent	Mantissa
0	10000001	10100000000000000000000

1. 4.5
2. 3.25
3. 6.5
4. 13.0

Special Floating Point Values

- **Zero** is represented as all zero bits.
- Not a Number (**NaN**) is a special value that indicates a floating point error, such as taking the square root of a negative number.
- Infinity (**INF**) both positive and negative.

Special Value Representation

Value	Sign	Exponent	Mantissa
Zero	0	0	0
+INF $+\infty$	0	11111111	0
-INF $-\infty$	1	11111111	0
NaN	0 or 1	0	Not zero

Overflow and Underflow

- When you calculate a number that is too big to fit into the floating point format, the result is infinity.
- Calculating a number that is too small (a positive number smaller than 1.18×10^{-38} for single precision) produces zero.
- Dividing by zero produces infinity with the proper sign.

Calculating with Infinity

- (+INF) + (+7) = (+INF)
- (+INF) × (−2) = (−INF)
- (+INF) × 0 = NaN—meaningless result