Floating-Point Numbers

Caiwen Ding
Department of Computer Science and Engineering
University of Connecticut

CSE3666: Introduction to Computer Architecture

Outline

- Real numbers in binary
- IEEE 754 floating-point number standards
  - Single precision and double precision
- RISC-V support for floating-point numbers

Reading: Section 3.5, excluding hardware support for floating-point numbers.

Real numbers

- Computers need to deal with
  - Numbers with fractions (not just whole numbers)
  - Very big numbers
  - Very small numbers

Example of real numbers in decimal:

$3.14159\ldots$

-0.002×10^{-20}

not normalized

9.4607×10^{15} (meters in a light year)

Normalized scientific notation:
Only one non-zero digit to the left of the decimal point.

Binary number with fraction

- To represent fractions in binary, we use bits after the binary point

What is the value of the following binary number?

101.1101

Binary point

Binary to decimal

Example: `0b101.1101`

1	0	1	1	1	0	1
2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

Multiply each bit with weight:

Integer part

Fractional part

0b101.1101

$= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$+ 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$

$= 4 + 0 + 1 + 0.5 + 0.25 + 0 + 0.0625$

$= 5.8125$

Decimal to binary

Example:

Convert the decimal number 0.8 to a binary number

0						
2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}

Converting decimal to binary

Decimal	Binary
0.8	0.
0.8 * 2 = 1.6	0.1
0.6 * 2 = 1.2	0.11
0.2 * 2 = 0.4	0.110
0.4 * 2 = 0.8	0.1100
0.8 * 2 = 1.6	0.11001…
Continue….	0.11001100110011001100 …

Fraction .8 appears again. The pattern 1100 will repeat forever.

```
python
>>> float.hex(0.8)
'0x1.999999999999ap-1'
```

Question

Convert the decimal number 0.9 to a binary number

0						
2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}

Converting decimal to binary

Decimal	Binary
0.9	0.
0.9 * 2 = 1.8	0.1
0.8 * 2 = 1.6	0.11
0.6 * 2 = 1.2	0.111
0.2 * 2 = 0.4	0.1110

What is the next digit?

A. 0   B. 1

Normalized notation of binary numbers

- There are many representations as we move the binary point

$$101.1101 = 10.11101 \times 2^1 = 1.011101 \times 2^2 = 0.1011101 \times 2^3$$

Normalized binary representation

The normalized binary representation has a single 1 before the point

$$\underbrace{\pm 1.x}_{\text{Significand}} \times 2^E$$

Exponent
is written in decimal for convenience

```
python
>>> float.hex(float.fromhex('5.d'))
'0x1.7400000000000p+2'
```

Encode floating-point numbers

- Given a number of bits, how do we represent

$$\pm 1.x \times 2^{E}$$

- What need to be encoded?
- How many bits for each?

Floating Point Standard (single and double precisions)

- Defined by IEEE Std 754-1985
  - Developed in response to divergence of representations
  - Solve the portability issues for scientific code
  - Now almost universally adopted

- Single precision (32-bit) and double precision (64-bit)
  - Double have more bits to represent exponent and fraction
  - They are types float and double in C

- Later versions of the standard include more types
  - E.g., 128-bit quad-precision

IEEE Floating-Point Format: single-precision

	8 bits	23 bits
S	Exponent	Fraction

Sign
0 : non-negative,
1: negative

Exponent in
excess-k representation
or biased representation

Bits after the binary point
There is a hidden 1 before the point!

$$\text{value} = (-1)^S \times (1.\text{Fraction}) \times 2^E$$

Exponent is in excess-127 representation. The Bias = 127.

$$\text{EncodedExponent} = \text{ActualExponent} + 127$$

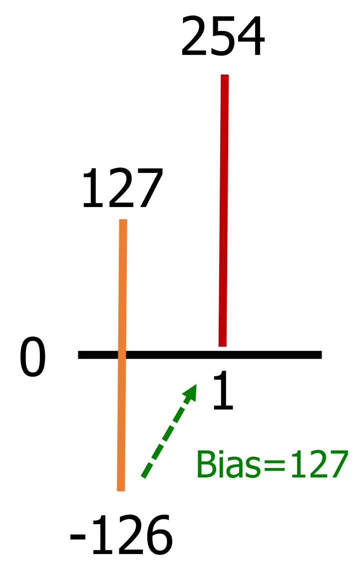
Exponent field in single-precision

- The exponent field has 8-bit, keeping a value in [0, 255]
  - [1, 254]: A normal SP number
  - We will discuss 0 and 255 soon
- The range of actual exponent: [-126, 127]
  - Excess-127 representation!

$$\pm 1.x \times 2^{E} \quad \text{and } E \in [-126, 127]$$

Encoded = E + 127

Bits in the exponent field
1 .. 254



Questions: Excess-127

- Given the eight bits in the exponent field of single-precision FP numbers, find the actual exponents in decimal.

  0111 1111

  0000 0100

  1000 0001

  1001 0000

Example: Read Single-Precision FP numbers

- What number (in decimal) is represented by the following single-precision floating-point number?

  1 10000001 010 0000 0000 0000 0000 0000

S = 1

Fraction = $01000\ldots00_2$

Encoded exponent = 10000001_2 = 129 (as 8-bit unsigned number)

Actual exponent = 129 – 127 = 2

The value is

$$(-1)^1 \times (1 + 0.01_2) \times 2^{(129-127)}$$
$$= (-1) \times 1.25 \times 2^2$$
$$= -5$$

Question

What is the actual exponent of the following single-precision floating-point number?

What is its value in decimal?

0x C1C0 0000

Example: Convert to Single-Precision FP numbers

Represent –0.75 with a single precision floating-point number

$$-0.75 = -\,0.11_2 = (-1)^1 \times 1.1_2 \times 2^{-1}$$

S = 1

Fraction = $1000\ldots00_2$

EncodedExponent = –1 + Bias = –1 + 127 = 126 = 01111110_2

1 01111110 100 0000 0000 0000 0000 0000

0xBF40 0000

Question: Convert to Single-Precision FP numbers

Represent 4.75 with a single precision floating-point number

Solutions

Represent 4.75 with a single precision floating-point number

$$4.75 = 100.11_2 = (-1)^0 \times 1.0011_2 \times 2^2$$

S = 0

Fraction = $0011000\ldots00_2$

EncodedExponent = 2 + Bias = 2 + 127 = 129 = 10000001_2

0 10000001 001 1000 0000 0000 0000 0000

0x4098 0000

Single-Precision Range (Normal Numbers)

- In normal SP FP numbers, encoded exponents are in [1, 254]
  - 00000000_2 and 11111111_2 are reserved

- What is the smallest positive value of normal SP FP numbers?

- What is the largest positive value of normal SP FP numbers?

Single-Precision Range (Normal Numbers)

- In normal SP FP numbers, exponents are from 1 to 254
  - 00000000_2 and 11111111_2 are reserved

- Smallest positive value
  - Exponent: $00000001_2 \Rightarrow$ actual exponent = 1 – 127 = –126
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0

$$1.0 \times 2^{-126} \approx 1.2 \times 10^{-38}$$

Can we have positive numbers that are even closer to 0?
How do we represent 0.0?

- Largest positive value
  - Exponent: $11111110_2 \Rightarrow$ actual exponent = 254 – 127 = 127
  - Fraction: 111…11 $\Rightarrow$ significand $\approx$ 2.0

$$2.0 \times 2^{+127} \approx 3.4 \times 10^{+38}$$

Denormalized/subnormal Numbers - 1

- Denormalized number: the exponent field is 0
  - The actual exponent is – 126 (= 1 – Bias) for single precision numbers
  - The hidden bit is 0

$$v = (-1)^S \times (0.\text{Fraction}) \times 2^{-126}$$

- 0 is a denormalized number !

All bits in exponent and fraction are 0.

But the sign can be 0 or 1. So we have two 0's!

0	0000 0000	000 0000 0000 0000 0000

1	0000 0000	000 0000 0000 0000 0000

$$x = (-1)^S \times (0.0) \times 2^{-126} = \pm 0.0$$

Denormalized Numbers - 2

- Denormalized numbers can represent numbers smaller than normal numbers
  - Allow for gradually approaching to 0, with diminishing precision

In the table, only the first number is a normal number

Exponent	Fraction	Actual exponent in decimal	Value
0000 0001	00000…00	-126	1.0×2^{-126} (normal number)
0000 0000	10000…00	-126	$0.1 \times 2^{-126} = 2^{-127}$
0000 0000	01000…00	-126	$0.01 \times 2^{-126} = 2^{-128}$
…			
0000 0000	00000…01	-126	$0.0…01 \times 2^{-126} = 2^{-149}$
0000 0000	00000…00	-126	$0.0…00 \times 2^{-126} = 0$

Infinities and NaN

- Exponent = 1111 1111, Fraction = 000...0
  - ±Infinity
  - Can be used in subsequent calculations, avoiding need for overflow check
- Exponent = 1111 1111, Fraction ≠ 000...0
  - Not-a-Number (NaN)
  - Indicates illegal or undefined result
    - e.g., 0.0 / 0.0
  - Can be used in subsequent calculations

Try these in Python:

```
float('inf') + 1.0
float('inf') + float('-inf')
```

IEEE Floating-Point Format: double precision

single: 8 bits
double: 11 bits

single: 23 bits
double: 52 bits

S	Exponent	Fraction

Sign
0 : non-negative,
1: negative

Exponent in
excess-k representation

Bits after the binary point
There is a hidden 1 !

$$\text{value} = (-1)^S \times (1.\text{Fraction}) \times 2^{(\text{EncodedExponent}-\text{Bias})}$$

Exponent in single-precision: excess-127: Bias = 127.

Exponent in double-precision: excess-1023: Bias = 1023

Single precision vs double precision

	Single	Double
Total number of bits	32	64
Number of bits in exponent	8	11
Number of bits in fraction	23	52
Bias	127	1023
Smallest positive value (normal values)	1.0×2^{-126} $\approx 1.18 \times 10^{-38}$	1.0×2^{-1022} $\approx 2.2 \times 10^{-308}$
Largest positive value	$2.0 \times 2^{+127}$ $\approx 3.4 \times 10^{+38}$	$2.0 \times 2^{+1023}$ $\approx 1.8 \times 10^{+308}$
Precision	23 bits ≈ 6 dec. digits	52 bits ≈ 16 dec. digits

Converting decimal to binary - 2

Decimal	Binary
0.9	0.
0.9 * 2 = 1.8	0.1
0.8 * 2 = 1.6	0.11
0.6 * 2 = 1.2	0.111
0.2 * 2 = 0.4	0.1110

We can find the first 4 digits after the binary point by the following steps:

$0.9 * 2^4 = 14.4$

Convert 14 to 4-bit binary number and we get 1110.

Solutions

```
0x C1C0 0000
1100 0001 1100 0000 0000 0000 0000 0000
```

S = 1

Fraction = $10000\ldots00_2$

Encoded Exponent = 10000011_2 = 131 (as unsigned)

Actual exponent = 131 – 127 = 4

The value is

$$(-1)^1 \times (1 + 0.1_2) \times 2^{(131-127)}$$

$$= -1 \times 1.5 \times 2^4$$

$$= -24$$