Chapter 4. Binary Floating-Point Numbers

Machine representation of real numbers:

- Fixed-point (discussed in Chapter 1)
- Floating-point

Compared to fixed point numbers, floating-point numbers are efficient in representing both very large and very small numbers.

§1. IEEE 754-2008 Standard

The IEEE 754-2008 standards are probably the most prevailing use of floating point numbers, which include four formats:

1. Single-precision format (32-bit)
2. Double-precision format (64-bit)
3. Single extended format ($\geqslant$ 44-bit)
4. Double extended format ($\geqslant$ 80-bit)

In this course we will introduce only the first two formats, IEEE single-precision format and IEEE double-precision format.

§1.1. IEEE single-precision (32 bits)

An IEEE single-precision floating-point number F has three parts, s, e and f, as shown below.

$$\underbrace{\underbrace{\boxed{s:\ \text{sign}}}_{1\text{ bit}}\underbrace{\boxed{e:\ \text{exponent}}}_{8\text{ bits}}\underbrace{\boxed{f:\ \text{significand or mantissa}}}_{23\text{ bits}}}_{32\text{ bits}}$$

IEEE floating-point number F can be evaluated by

$$F = (-1)^s \times 1.f \times 2^{e-127}. \tag{1}$$

Note in (1) that there is a hidden 1 that is not shown in the representation of F. The maximal and minimal values of F can be decided by

$$\begin{aligned} F_{\text{max}} &= (2 - 2^{-23}) \times 2^{254-127} = (1 - 2^{-24}) \times 2^{128}, \\ F_{\text{min}} &= 1 \times 2^{1-127} = 2^{-126}. \end{aligned}$$

How to represent zero? Use the reserved formats as shown in the table below.

Value or meaning of an IEEE single-precision floating-point representation		
$e = 0$	$f = 0$	$F = \pm 0$
	$f \neq 0$	F are subnormal numbers ($= \pm 0.f \times 2^{-126}$)
$e = 255$	$f = 0$	$F = \pm\infty$
	$f \neq 0$	F is NAN (Not a Number)
$1 \leqslant e \leqslant 254$	$-$	F is an ordinary number and $F = (-1)^s \times 1.f \times 2^{e-127}$

Example 1 Convert 46.5_{10} into IEEE single-precision standard.

Solution: First we convert the given number 46.5_{10} into binary of the form of (1):

$$46.5_{10} = 101110.1_2 = 1.0111 \times 2^5 = (-1)^0 \times 1.0111 \times 2^{132-127}.$$

Then it is easy to obtain

$$\begin{cases} s &= 0 \\ e &= 1000\ 0100 \\ f &= 0111\ 0000\ 0000\ 0000\ 0000\ 000 \end{cases}$$

§1.2. IEEE double-precision (64 bits)

An IEEE double-precision floating-point number F has also three parts, s, e and f, as shown below.

$$\underbrace{\underbrace{\boxed{s: \text{ sign}}}_{\text{1 bit}}\underbrace{\boxed{e: \text{ exponent}}}_{\text{11 bits}}\underbrace{\boxed{f: \text{ significand or mantissa}}}_{\text{52 bits}}}_{\text{64 bits}}$$

IEEE floating-point number F can be evaluated by

$$F = (-1)^s \times 1.f \times 2^{e-1023}. \qquad (2)$$

Note a hidden 1 also exists for double-precision case that is not shown in the representation of F. The maximal and minimal values of F can be decided by

$$\begin{aligned} F_{\text{max}} &= (2-2^{-52}) \times 2^{2046-1023} = (1-2^{-53}) \times 2^{1024}, \\ F_{\text{min}} &= 1 \times 2^{1-1023} = 2^{-1022}. \end{aligned}$$

The value or meaning of an IEEE double-precision number F is given in the following table:

Value or meaning of an IEEE double-precision floating-point representation		
$e = 0$	$f = 0$	$F = \pm 0$
	$f \neq 0$	F are subnormal numbers $(= \pm 0.f \times 2^{-1022})$
$e = 1023$	$f = 0$	$F = \pm\infty$
	$f \neq 0$	F is NAN (Not a Number)
$1 \leqslant e \leqslant 1022$	$-$	F is an ordinary number and $F = (-1)^s \times 1.f \times 2^{e-1023}$

Example 2 Convert 46.5_{10} into IEEE double-precision standard.

Solution: First we convert the given number 46.5_{10} into binary of the form of (2):

$$46.5_{10} = 101110.1_2 = 1.0111 \times 2^5 = (-1)^0 \times 1.0111 \times 2^{1028-1023}.$$

Then it is easy to obtain

$$\begin{cases} s &= 0 \\ e &= 1000\ 0000\ 100 \\ f &= 0111\ 0000\ \underbrace{00\cdots 00}_{\text{44 zeros}} \end{cases}$$

§2. A Summary of Floating-Point Representations

A floating-point representation F has three parts:

- The sign, S;
- The significand (or mantissa), f (or M);
- The exponent, E.

The bits of the floating-point number F are stored in a register or a memory unit shown as follows

$$F = \boxed{S: \text{ sign}}\boxed{E: \text{ exponent}}\boxed{f \text{ or } M: \text{ significand}}$$

and the value of such a floating-point number F is given by

$$F = (-1)^S \times M \times \beta^{E-\text{bias}}.$$

where β is the base of E which is implied in a system. Usually, β is often chosen as a power of 2.

The representation range is $[-F_{\text{max}}, -F_{\text{min}}]$ and $[F_{\text{min}}, F_{\text{max}}]$, where F_{max} and F_{min} are given by

$$\begin{aligned} F_{\text{max}} &= f_{\text{max}} \times \beta^{e_{\text{max}}-\text{bias}}, \\ F_{\text{min}} &= f_{\text{min}} \times \beta^{e_{\text{min}}-\text{bias}}. \end{aligned}$$

The computer system is overflow if a result is larger than F_{max} or smaller than $-F_{\text{max}}$. The system is called underflow if a result is nonzero and belongs to the interval $(-F_{\text{min}}, F_{\text{min}})$.

§3. Floating-Point Operations

The following discussions are using the generic floating point representation,

$$F = (-1)^S \times M \times \beta^{E-Bias}.$$

§3.1. Multiplication

Given two numbers

$$F_1 = (-1)^{S_1} \cdot M_1 \cdot \beta^{E_1 - \text{bias}} \text{ and } F_2 = (-1)^{S_2} \cdot M_2 \cdot \beta^{E_2 - \text{bias}},$$

and assume that they both are in normalized forms. Then the product $F_3 = F_1 \times F_2$ and

$$F_3 = (-1)^{S_3} \cdot M_3 \cdot \beta^{E_3 - \text{bias}}$$

can be obtained as follows.

1. Calculate $E_3 = E_1 + E_2 -$ bias.
   IF $(E_3 > E_{\max})$ THEN overflow ELSE IF $(E_3 < E_{\min})$ THEN underflow.
2. Calculate $M_3 = M_1 \times M_2$.
   IF $(M_3 < (1/\beta))$ THEN $M_3 = M_3 \times \beta$ and $E_3 = E_3 - 1$.
3. IF $(E_3 < E_{\min})$ THEN underflow.

The second step above is called post-normalization.

§3.2. Addition/Subtraction

Let two float-point numbers be given by

$$\begin{aligned} F_1 &= (-1)^{S_1} \times M_1 \times \beta^{E_1 - bias} \\ F_2 &= (-1)^{S_2} \times M_2 \times \beta^{E_2 - bias} \end{aligned}$$

then addition/subtraction operation can be performed as follows.

1. Assume that $E_1 \geqslant E_2$ and compute

$$F_3 = F_1 \pm F_2 = [(-1)^{S_1} \times M_1 \pm (-1)^{S_2} \times M_2 \times \beta^{-(E_1 - E_2)}] \times \beta^{E_1 - bias}$$

2. Let $(-1)^{S_1} \times M_1 \pm (-1)^{S_2} \times M_2 \times \beta^{-(E_1 - E_2)}$ be denoted as M_3. If $M_3 < (1/\beta)$ or $M_3 \geqslant 1$ then post-normalization is needed.

3. If there is post-normalization then we need to check whether or not the final exponent E_3 is overflow or underflow.

Example 3 Convert $F_1 = 4$ and $F_2 = 3$ into IEEE floating point single precision format. Then perform floating point operations $F_1 \times F_2$ and $F_1 + F_2$.

Solution:

$$F_1 = 4 = (-1)^{S_1} \times 1.f_1 \times 2^{E_1-127} = (-1)^0 \times 1.0 \times 2^{129-127},$$
$$F_2 = 3 = (-1)^{S_2} \times 1.f_2 \times 2^{E_2-127} = (-1)^0 \times 1.1 \times 2^{128-127}.$$

Then for floating point multiplication we have

$$F_3 = F_1 \times F_2 = (-1)^0 \times (1.0 \times 1.1) \times 2^{(129-127+128)-127} = (-1)^0 \times 1.1 \times 2^{130-127}.$$

For floating point addition, since $F_1 > F_2$ we first rewrite F_2 such that its 2's power part is the same as that of F_1.

$$F_2 = (-1)^0 \times 1.1 \times 2^{128-127} = (-1)^0 \times 0.11 \times 2^{129-127}.$$

Then it follows

$$F_4 = F_1 + F_2 = (-1)^0 \times (1.0 + 0.11) \times 2^{129-127} = (-1)^0 \times 1.11 \times 2^{129-127}.$$

§4. Rounding Schemes

Rounding is a technique to obtain low-precision representation from given high-precision representation:

High-precision → *Rounding* → Low-precision

We assume that the input to a rounding scheme has m integer bits and d fractional bits,

$$x_{m-1} \dots x_0.x_{-1} \dots x_{-d},$$

and the output contains only integer bits, $y_{m'-1} \dots y_0$. This can be shown as

$$x_{m-1} \dots x_0.x_{-1} \dots x_{-d} \rightarrow \textit{Rounding} \rightarrow y_{m'-1} \dots y_0.$$

Let X and $Y(X)$ denote the input and the output of a rounding scheme, respectively. Rounding error is defined as $R(X) = Y(X) - X$. We measure the accuracy of the rounding results by computing the maximum errors and the bias of the scheme, where bias is defined as the average error for a block of 2^d numbers including all the possible inputs to the rounding scheme.

Criteria for choosing a "good" rounding scheme (items with check mark will be discussed):

1. Accuracy of the output
   - small maximum errors ✓
   - small bias ✓
   - small variation
2. Low time delay or speed ✓
3. Low implementation cost ✓

§4.1. Truncation or chopping: chop(x)

1. Definition: chop$(x) = \lfloor x \rfloor$.
2. Truth table:

Chopping scheme with $d = 2$		
Input: x	Output: chop(x)	Error: chop$(x) - x$
$\times.00$	$\times.$	0
$\times.01$	$\times.$	$-1/4$
$\times.10$	$\times.$	$-1/2$
$\times.11$	$\times.$	$-3/4$

3. Error and bias: It can be seen from the above table that the maximal error is $e^-_{\text{max}} = -3/4$ and

$$\text{Bias} = \frac{1}{4}\Big(0 - \frac{1}{4} - \frac{1}{2} - \frac{3}{4}\Big) = -\frac{3}{8}.$$

4. Implementation: Its implementation is cost free.
5. Time delay: There is no delay incurred with this rounding scheme.

§4.2. Round to nearest integer: round(x)

1. definition: round$(x) = \lfloor x + 0.5 \rfloor$.

2. Truth table:

Round-to-nearest scheme with $d = 2$		
Input: x	Output: round(x)	Error: round$(x) - x$
$\times.00$	$\times.$	0
$\times.01$	$\times.$	$-1/4$
$\times.10$	$\times. + 1$	$+1/2$
$\times.11$	$\times. + 1$	$+1/4$

3. Error and bias: It can be seen from the above table that the maximal error is $e^+_{\text{max}} = +1/2$ and

$$\text{Bias} = \frac{1}{4}\Big(0 - \frac{1}{4} + \frac{1}{2} + \frac{1}{4}\Big) = +\frac{1}{8}.$$

4. Implementation: An implementation requires an adder and a few logic gates.

5. Time delay: It is equal to that of an adder of the size of the output.

§4.3. Round to nearest even integer: rtne(x)

1. definition: Round to the nearest even integer if it is a tie case.

2. Truth table:

Round-to-nearest-even scheme with $d = 2$		
Input: x	Output: rtne(x)	Error: rtne$(x) - x$
$\times0.00$	$\times0.$	0
$\times0.01$	$\times0.$	$-1/4$
$\times0.10$	$\times0.$	$-1/2$
$\times0.11$	$\times1.$	$+1/4$
$\times1.00$	$\times1.$	0
$\times1.01$	$\times1.$	$-1/4$
$\times1.10$	$\times1. + 1$	$+1/2$
$\times1.11$	$\times1. + 1$	$+1/4$

3. Errors and bias: It can be seen from the above table that the maximal errors are $e^{+}_{\text{max}} = +1/2$ and $e^{-}_{\text{max}} = -1/2$.

$$\text{Bias} = \frac{1}{8}\Big(0 - \frac{1}{4} - \frac{1}{2} + \frac{1}{4} + 0 - \frac{1}{4} + \frac{1}{2} + \frac{1}{4}\Big) = 0.$$

4. Implementation and time delay: slightly higher than round(x).

§4.4. ROM Rounding: ROM(x)

1. Definition: ROM rounding is given by

$$x_{m-1} \dots x_{\ell-1} x_{\ell-2} \dots x_0 . x_{-1} x_{-2} \dots x_{-d} \to \text{ROM}(x) \to x_{m-1} \dots x_{\ell-1} y_{\ell-2} \dots y_0 .$$

Note that only ℓ bits are taken as actual input to ROM, $x_{\ell-2} \dots x_0 . x_{-1}$, and then ROM generates $\ell - 1$ output bits, $y_{\ell-2} \dots y_0$.

2. Truth table:

ROM scheme with $\ell = 3$ input bits		
Input: x	Output: ROM(x)	Error: ROM$(x) - x$
$\times 00.0$	$\times 00.$	0
$\times 00.1$	$\times 01.$	$+1/2$
$\times 01.0$	$\times 01.$	0
$\times 01.1$	$\times 10.$	$+1/2$
$\times 10.0$	$\times 10.$	0
$\times 10.1$	$\times 11.$	$+1/2$
$\times 11.0$	$\times 11.$	0
$\times 11.1$	$\times 11.$	$-1/2$

3. Error and bias: It can be seen from the above table that the maximal errors are $e^{+}_{\text{max}} = +1/2$ and $e^{-}_{\text{max}} = -1/2$.

$$\text{Bias} = \frac{1}{8}\Big(0 + \frac{1}{2} + 0 + \frac{1}{2} + 0 + \frac{1}{2} + 0 - \frac{1}{2}\Big) = \frac{1}{8}.$$

4. Implementation cost: Size of the ROM used.

5. Time delay: Decided by the ROM reading speed.