

FLOATING-POINT REPRESENTATIONS

- Fixed-point representation of Avogadro's number:

$$N_0 = 602\,000\,000\,000\,000\,000\,000\,000$$

- A floating-point representation is just scientific notation:

$$N_0 = 6.02 \times 10^{23}$$

- ▷ The decimal point “floats”: It can represent any power of 10
- ▷ A floating-point representation is much more economical than a fixed-point representation for very large or very small numbers
- Examples of areas where a floating-point representation is necessary:
 - ▷ Engineering: Electromagnetics, aeronautical engineering
 - ▷ Physics: Semiconductors, elementary particles
- Fixed-point representation often used in digital signal processing (DSP)

FLOATING-POINT REPRESENTATIONS (2)

- Decimal floating-point representation:
 - ▷ Decimal place-value notation, extended to fractions
 - ▷ Expand a real number r in powers of 10:

$$\begin{aligned} r &= d_n d_{n-1} \cdots d_0 . f_1 f_2 \cdots f_m \cdots \\ &= d_n 10^n + d_{n-1} 10^{n-1} + \cdots + d_0 10^0 \\ &\quad + \frac{f_1}{10} + \frac{f_2}{10^2} + \cdots + \frac{f_m}{10^m} + \cdots \end{aligned}$$

- ▷ Scientific notation for the same real number:

$$r = d_n . d_{n-1} \cdots d_0 f_1 f_2 \cdots f_m \cdots \times 10^n$$

- ▷ Re-label the digits:

$$r = f'_0 . f'_1 f'_2 \cdots f'_m \cdots \times 10^n = \sum_{k=0}^{\infty} \frac{f'_k}{10^k} \times 10^n$$

FLOATING-POINT REPRESENTATIONS (3)

- Binary floating-point representation:
 - ▷ Binary place-value notation, extended to fractions
 - ▷ Expand a real number r in powers of 2:

$$r = f_0.f_1f_2\cdots f_m\cdots \times 2^n = \sum_{k=0}^{\infty} \frac{f_k}{2^k} \times 2^n$$

- ▷ Example:

$$\begin{aligned}\frac{1}{3} &= \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \cdots \\ &= 1.0101010\cdots \times 2^{-2}\end{aligned}$$

- ▷ **Normalization:** Require that

$$f_0 \neq 0$$

FLOATING-POINT REPRESENTATIONS (4)

- Important binary floating-point numbers:

▷ One:

$$1.0 = 1.00 \dots \times 2^0$$

▷ Two:

$$1.111 \dots \times 2^0 = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots$$

$$= \frac{1}{1 - \frac{1}{2}} \quad (\text{sum of a geometric series})$$

$$= 1.000 \dots \times 2^1$$

▷ **Rule:**

$$1.b_1b_2 \dots b_k 111 \dots = 1.b_1b_2 \dots (b_k + 1)000 \dots$$

(note that the sum $b_k + 1$ may generate a carry bit)

FLOATING-POINT REPRESENTATIONS (5)

- A more complicated example:
 - ▷ Obtain the binary floating-point representation of the number 2.6_{10}
 - ▷ Expand in powers of 2:

$$2.6 = 2\frac{3}{5} = 1 \times 2^1 + 0 \times 2^0 + \frac{3}{5}$$

where (see later slides for a more efficient method)

$$\begin{aligned}\frac{3}{5} &= \frac{1}{2} + \frac{1}{10} \\ &= \frac{1}{2} + \frac{1}{16} + \left(\frac{1}{10} - \frac{1}{16}\right) = \frac{1}{2} + \frac{1}{16} + \frac{3}{80} = \frac{1}{2} + \frac{1}{16} + \left(\frac{1}{16}\right)\left(\frac{3}{5}\right) \\ &= \frac{1}{2^1} + \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^8} + \frac{1}{2^9} + \dots \\ &= 0.10011001100\dots \times 2^0\end{aligned}$$

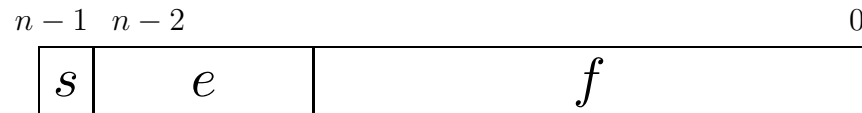
$$2.6_{10} = 1.010011001100\dots \times 2^1$$

INTEGER OPERATIONS ON FLOATING-POINT NUMBERS

- Requirements for optimization of important operations:
 - ▷ Use existing integer operations to test sign of, or compare, FPNs
 - ▷ Sign must be shown by most significant bit
 - ▷ Lexicographic order of exponents = numerical order
⇒ biased representation

FLOATING-POINT REPRESENTATIONS (6)

- Bit sequence:



2 conventions for assigning a value to this bit string:

$$r = (-1)^s 2^{e-B} 0.f \quad \text{or} \quad r = (-1)^s 2^{e-B'} 1.f$$

- ▷ s is the **sign bit**, e is the **exponent field**, B is the **bias**, f is the **fraction** or **mantissa**, and the extra 1 (if any) is the **implicit 1**
- ▷ The exponent is represented in biased format
- ▷ The bits of the fraction are interpreted as the coefficients of powers of $\frac{1}{2}$, in place-value notation

FLOATING-POINT REPRESENTATIONS (7)

- IEEE-754 single precision format:



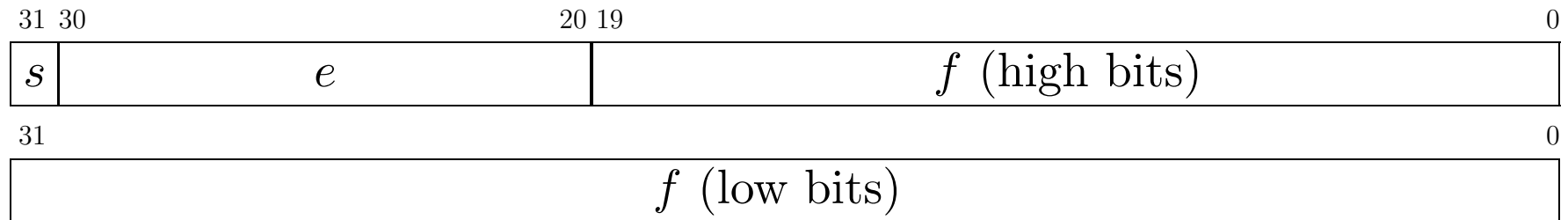
Numerical value assigned to this 32-bit word, interpreted as a floating-point number:

$$r = (-1)^s 2^{e-127} 1.f$$

- ▷ e is the exponent, interpreted as an unsigned integer ($0 < e < 255$)
- ▷ The value of the exponent is calculated in biased-127 format
- ▷ The notation $1.f$ means $1 + \frac{f}{2^{23}}$ if f is interpreted as an unsigned integer

FLOATING-POINT REPRESENTATIONS (8)

- IEEE-754 double precision format uses two consecutive 32-bit words:



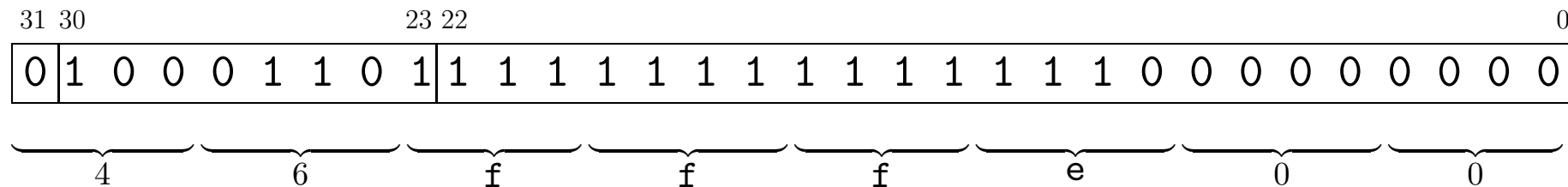
Numerical value assigned to this 32-bit word, interpreted as a floating-point number:

$$r = (-1)^s 2^{e-1023} 1.f$$

- ▷ e is the exponent, interpreted as an unsigned integer ($0 < e < 2047$)
- ▷ The value of the exponent is calculated in biased-1023 format
- ▷ The notation $1.f$ means $1 + \frac{f}{2^{52}}$ (where f = unsigned integer)

FLOATING-POINT REPRESENTATIONS (9)

- Find the numerical value of the floating-point number with the IEEE-754 single-precision representation 0x46fffe00:



▷ Value of exponent = $0x8d - B = 141 - B = 141 - 127 = 14$

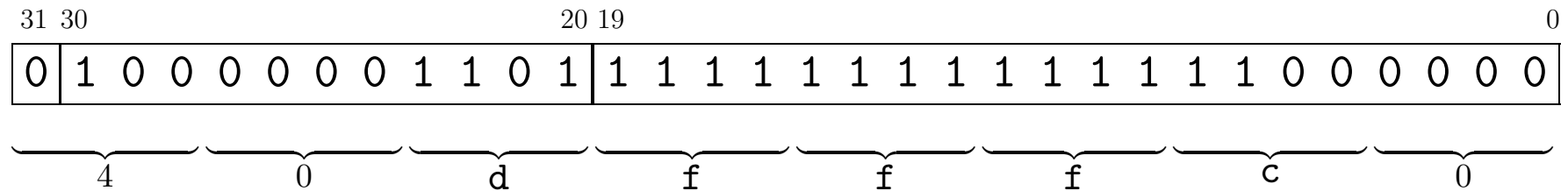
$$\text{Value of fraction} = 1 + \frac{1111111111111110000000000}{2^{23}}$$

$$= 1 + \frac{111111111111111}{1000000000000000} = 2^{-14} \times \underbrace{111111111111111}_{15 \text{ 1's}}$$

$$\text{Value of number} = 2^{-14} \times 0x7fff \times 2^{14} = 0x7fff = 32767_{10}$$

FLOATING-POINT REPRESENTATIONS (10)

- Find the numerical value of the floating-point number with the IEEE-754 double-precision representation 0x40dffffc0 00000000:



(second word is all 0's)

$$\triangleright \text{Value of exponent} = 0x40d - B = 1037 - B = 1037 - 1023 = 14$$

$$\text{Value of fraction} = 1 + \frac{1111111111111111000000}{2^{52-32}}$$

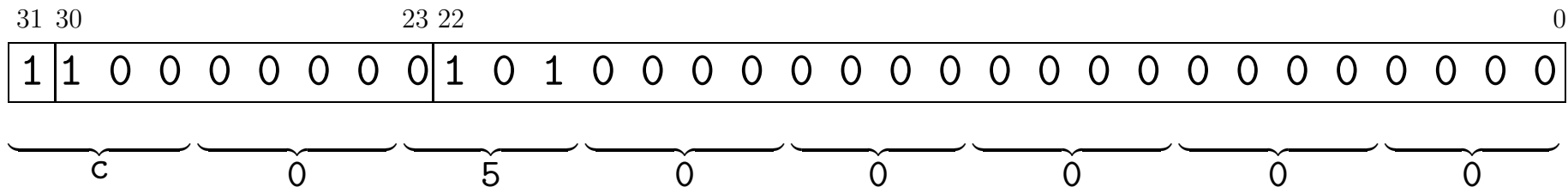
$$= 1 + \frac{1111111111111111}{10000000000000000} = 2^{-14} \times \underbrace{1111111111111111}_{15 \text{ 1's}}$$

$$\text{Value of number} = 2^{-14} \times 0x7fff \times 2^{14} = 0x7fff = 32767_{10}$$

FLOATING-POINT REPRESENTATIONS (11)

- Conversion of a number r from the decimal representation to the IEEE-754 single-precision binary representation:
 1. If $r < 0$, perform the following steps with $-r$ and change the sign at the end
 2. Find the base-2 exponent k such that $2^k \leq r < 2^{k+1}$
 3. Compute $e = k + B$, where $B = 127$ for single precision and $B = 1023$ for double precision, and express e in base 2
 4. Compute $1.f = \frac{r}{2^k}$; check that $1 \leq 1.f < 2$
 5. Expand $0.f = 1.f - 1$ as a binary fraction $\frac{b_{p-2}}{2} + \frac{b_{p-3}}{2^2} + \cdots + \frac{b_0}{2^{p-1}}$ where $p = 24$ for single precision and $p = 53$ for double precision. Then $f = b_{p-2}b_{p-3} \cdots b_0$.

- Then $f = 10100000000000000000000_2$.



FLOATING-POINT REPRESENTATIONS (12a)

- Most efficient method for conversion of the fraction from base 10 to base 2:
 1. Let $0.f$ (base 2) $= 0.d_{-1}d_{-2}\cdots d_{-k}\cdots$, where d_{-k} multiplies 2^{-k}
 2. Set F = fraction in base 10 (the digits after the decimal point)
 3. Set $k = 1$
 4. Compute $d_{-k} = \lfloor 2F \rfloor$ = integer part of $2F$
 5. Replace F with the fractional part of $2F$ (the part after the decimal point)
 6. Replace k with $k - 1$
 7. If you have computed $> p$ bits of f , or if $F = 0$, stop. Otherwise go to 4 and continue.

FLOATING-POINT REPRESENTATIONS (13)

- Floating-point addition:
 1. Assume that both summands are positive and that $r' < r$
 2. Find the difference of the exponents, $e - e' > 0$
 3. Set bit 23 and clear bits 31–24 in both r and r'
$$r = r \ \& \ 0x00ffffff;$$
$$r = r \ | \ 0x00800000;$$
 4. Shift r' right $e - e'$ places (to align its binary point with that of r)
 5. Add r and r' (shifted) as unsigned integers; $u = r + r'$
 6. Compute $t = u \ \& \ 0xff000000$;
 7. **Normalization:** Shift u right t places and compute $e = e + t$
 8. Compute $f = u - 0x00800000$; f is the fraction of the sum.

FLOATING-POINT REPRESENTATIONS (14a)

- Example of floating-point addition: Compute $r + r_p$, where $r = 3.25 = 0x40500000$, $r_p = 0.25 = 0x3e800000$
 1. Both summands are positive and $r_p = 0.25 < r = 3.25$
 2. The difference of the exponents is $1 - (-2) = 3$
 3. Copy bits 22–0, clear bits 31–24 and set bit 23 in both r and r_p :

$$a1 = r \ \& \ 0x007ffffff = 0x00500000$$

$$u1 = a1 \mid 0x00800000 = 0x00d00000$$

$$a2 = r_p \ \& \ 0x007ffffff = 0x00000000$$

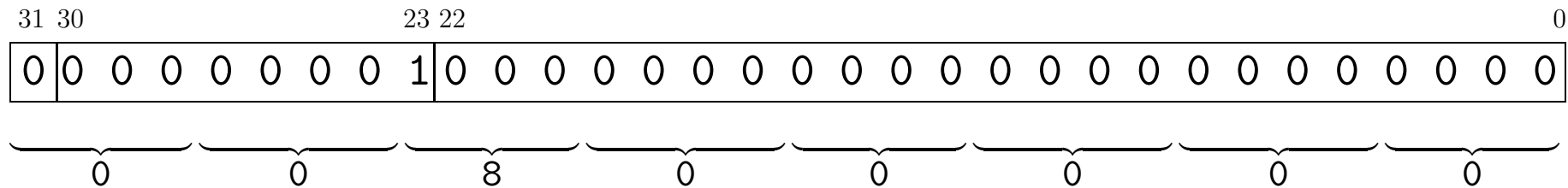
$$u2 = a2 \mid 0x00800000 = 0x00800000$$

FLOATING-POINT REPRESENTATIONS (14b)

- Example of floating-point addition: $3.25 + 0.25$ (continued)
 4. Shift u_2 right $e - e' = 3$ places to align its binary point:
 $u_2 \text{ (shifted)} = 0x00100000$
 5. Add u_1 and u_2 (shifted) as unsigned integers:
 $u = u_1 + u_2 \text{ (shifted)} = 0x00e00000$
 6. Compute $t = u \ \& \ 0xff000000 = 0x00000000$
 7. **Normalization:** Unnecessary in this example, because $t = 0$
 8. Subtract implicit 1 bit:
 $f = u - 0x00800000 = 0x00600000$
 Value of $1.f = 1 + \frac{1}{2} + \frac{1}{4}$
 Value of sum $= (1 + \frac{1}{2} + \frac{1}{4}) \times 2^1$
 9. **Answer:** $r + rp = (1 + \frac{1}{2} + \frac{1}{4}) \times 2^1$
 Check: $3.25 + 0.25 = 3.5 = 1.75 \times 2^1$

FLOATING-POINT REPRESENTATIONS (15)

- Find the smallest normalized positive floating-point number in the IEEE-754 single-precision representation:



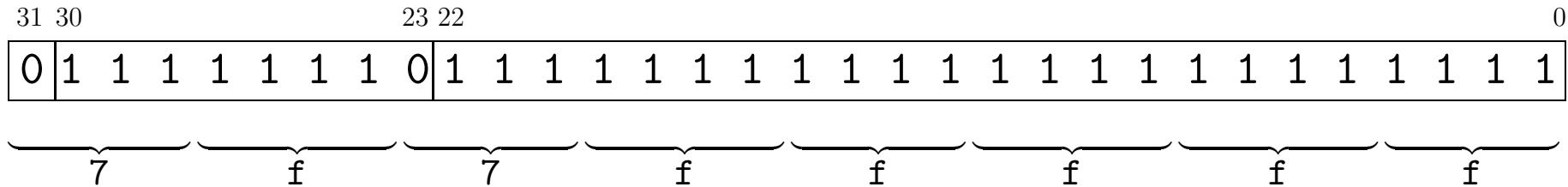
▷ Value of exponent = $k = 0x01-B = 1 - B = 1 - 127 = -126$

$$\text{Value of fraction} = 1.f = 1 + \frac{0}{2^{23}} = 1$$

Value of smallest normalizable number = $2^{-126} \times 1 \approx 1.175 \times 10^{-38}$

FLOATING-POINT REPRESENTATIONS (16)

- Find the largest normalized positive floating-point number in the IEEE-754 single-precision representation:



▷ Value of exponent = $0xf\text{e}-B = 254 - B = 254 - 127 = 127$

$$\begin{aligned}\text{Value of fraction} &= 1 + \frac{111111111111111111111111}{2^{23}} \\ &= 1 + \frac{111111111111111111111111}{100000000000000000000000} \approx 2\end{aligned}$$

Value of number $\approx 2^{127} \times 2 \approx 3.403 \times 10^{38}$

FLOATING-POINT REPRESENTATIONS (17)

- Find the smallest and largest normalized positive floating-point numbers in the IEEE-754 double-precision representation:
 - ▷ Smallest number is $1 \times 2^{-1022} \approx 10^{-307.65} \approx 4.4668 \times 10^{-308}$
 - ▷ Largest number $\approx 2 \times 2^{4095-1023} = 2^{1023} \approx 10^{307.95} \approx 8.9125 \times 10^{307}$

FLOATING-POINT REPRESENTATIONS (18)

- How many floating-point numbers are there in a given representation?
 - ▷ Let
 - β = base
 - p = number of significant digits
 - ▷ Number of values of exponent $e = e_{\max} - e_{\min} + 1$
 - ▷ Number of properly normalized values of the fraction = $2(\beta - 1)\beta^{p-1}$
(taking signs into account)
 - ▷ Total number of normalized floating-point numbers in a representation:

$$N(\beta, p, e_{\max}, e_{\min}) = 2(\beta - 1)\beta^{p-1}(e_{\max} - e_{\min} + 1) + \begin{cases} 1, & \text{if zero is unsigned, or} \\ 2, & \text{if zero is signed.} \end{cases}$$

FLOATING-POINT REPRESENTATIONS (19)

- **Machine epsilon**, ϵ_{mach} : The smallest positive floating-point number such that

$$1. + \epsilon_{\text{mach}} > 1.$$

▷ Generally

$$\epsilon_{\text{mach}} = \beta^{1-p}$$

where

- β is the base
- p is the number of significant digits

▷ For IEEE-754,

$$\epsilon_{\text{mach}} = \begin{cases} 2^{-23} \approx 1.19 \times 10^{-7} & \text{in single precision;} \\ 2^{-52} \approx 2.22 \times 10^{-16} & \text{in double precision.} \end{cases}$$

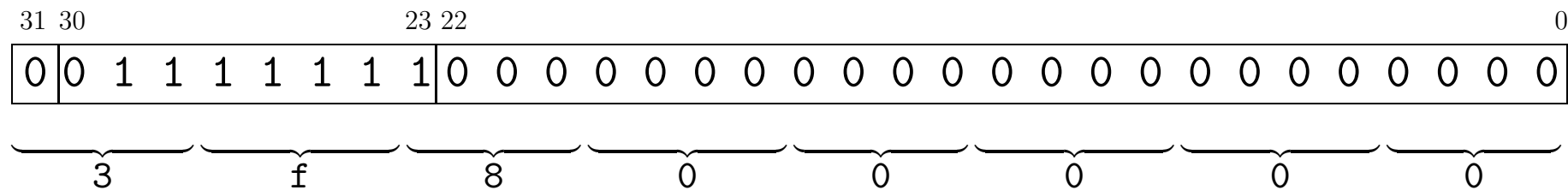
FLOATING-POINT REPRESENTATIONS (20)

- **Machine epsilon** in the IEEE-754 single-precision representation:
 1. Compute $r + rp = 1 + 2^{-23}$
 2. The difference of the exponents is $0 - (-23) = 23$
 3. Set bit 23 and clear bits 31–24 in both r and rp :
 $r \& 0x00ffffff = 0x00000000$, $r | 0x00800000 = 0x00800000$
 $rp \& 0x00ffffff = 0x00000000$, $rp | 0x00800000 = 0x00800000$
 4. Shift rp right $e - e' = 23$ places to align its binary point with that of r : $0x00000001$
 5. Add r and rp (shifted) as unsigned integers: $u = r + rp = 0x00800001$
 6. Compute $t = u \& 0xff000000 = 0x00000000$
 7. **Normalization:** Unnecessary in this example, because $t = 0$
 8. Compute $f = u - 0x00800000 = 0x00000001$; value $= 1 + 2^{-23}$
 9. **A smaller rp results in $u = r$**

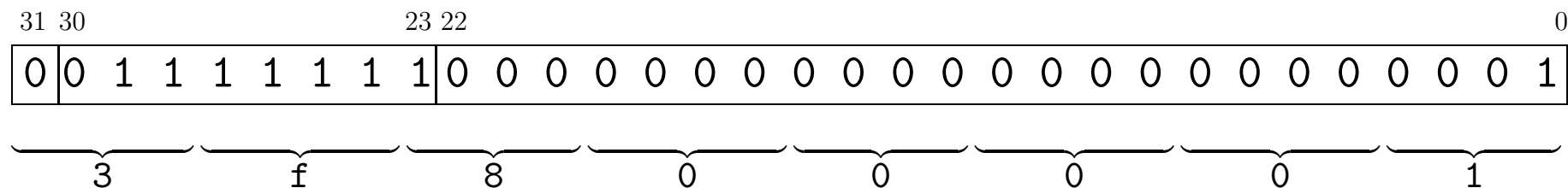
FLOATING-POINT REPRESENTATIONS (21)

- Consecutive floating-point numbers (IEEE-754 single precision):

▷ $1.000000000000000000000000 \times 2^0$:

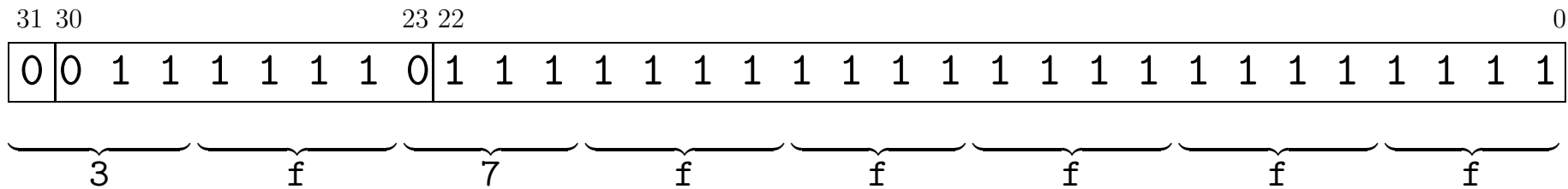
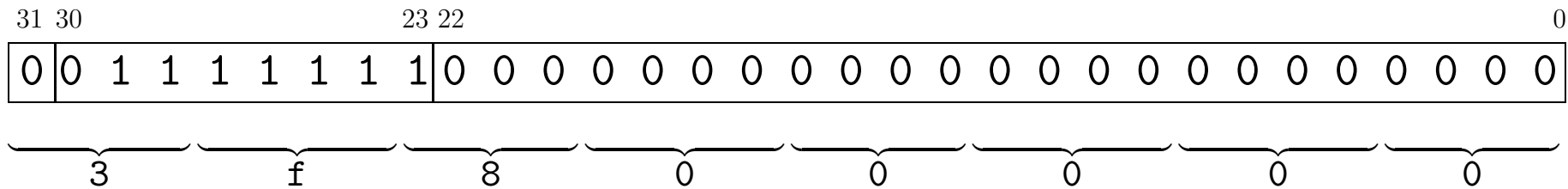


▷ $1.000000000000000000000001 \times 2^0$:



FLOATING-POINT REPRESENTATIONS (22)

- Consecutive floating-point numbers (IEEE-754 single precision):

 $\triangleright 1.1111111111111111111111111111 \times 2^{-1}:$ 
$$\triangleright 1.00000000000000000000000000000000 \times 2^0:$$


FLOATING-POINT REPRESENTATIONS (23)

- Differences between consecutive floating-point numbers (IEEE-754 single precision):

$$1.0000000000000000000000001 \times 2^0 - 1.0000000000000000000000001 \times 2^0 \\ = 2^{-23} = \beta^{1-p}$$

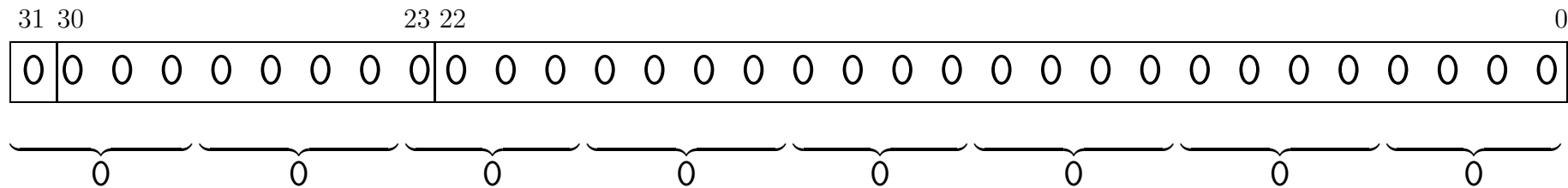
$$1.0000000000000000000000000 \times 2^0 - 1.111111111111111111111111 \times 2^{-1} \\ = 2^{-24} = \beta^{-p}$$

- There is a “wobble” of a factor of β ($= 2$ in binary floating-point representations) between the maximum and minimum relative change represented by 1 unit in the last place
 - ▷ The “wobble” is 16 in hexadecimal representations
 - ▷ Base 2 is the best for floating-point computation

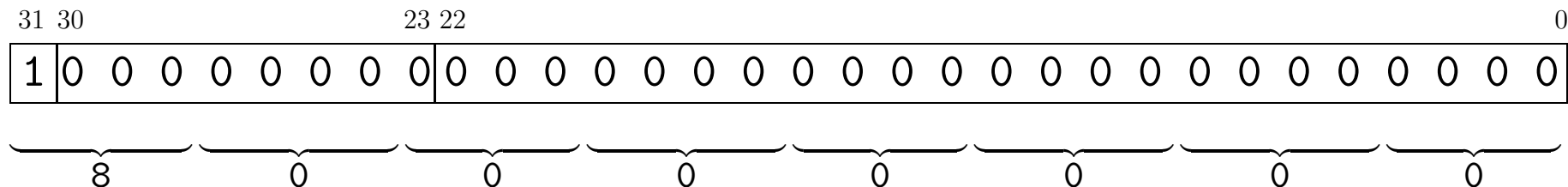
FLOATING-POINT REPRESENTATIONS (24)

- As for all sign-magnitude representations, there are two IEEE-754 single-precision representations for zero:

▷ $+0.000000000000000000000000$:



▷ $-0.000000000000000000000000$:



FLOATING-POINT REPRESENTATIONS (25)

- Invalid results of floating-point operations after normalization:
 - ▷ **Overflow:** $e > e_{\max}$
 - Single precision: $e_{\max} = 254$ (including a bias of 127)
 - Double precision: $e_{\max} = 2046$ (including a bias of 1023)
 - ▷ **Underflow:** $e < e_{\min}$
 - Single precision: $e_{\min} = 1$ (including a bias of 127)
 - Double precision: $e_{\min} = 1$ (including a bias of 1023)
- Action taken:
 - ▷ Overflow:
 - If $e = e_{\max} + 1$ and $f \neq 0$, result is a NaN
 - If $e = e_{\max} + 1$ and $f = 0$, result is $\pm\text{Inf}$
 - ▷ Underflow: Result is a denormalized floating-point number

FLOATING-POINT REPRESENTATIONS (26)

- Example of valid use of **Inf**: Evaluate

$$f(x) = \frac{1}{1 + 1/x}$$

starting at $x = 0.0$, in steps of 10^{-5}

- ▷ First evaluation: (at $x = 0.0$)

$$1/x = \text{Inf}$$

Value returned for $f(x)$ is $1/\text{Inf} = 0.0$

- ▷ Correct value is returned despite division by zero!
- ▷ Computation continues, giving correct result for all values of x

FLOATING-POINT REPRESENTATIONS (27)

- Example of invalid comparison of floating-point numbers:

```
if (x.ne.y) then
  z=1./(x-y)
else
  z=0.
  ierr=1
end if
```

▷ Consider

$$x = 1.00100 \dots 0 \times 2^{-126},$$

$$y = 1.00010 \dots 0 \times 2^{-126}$$

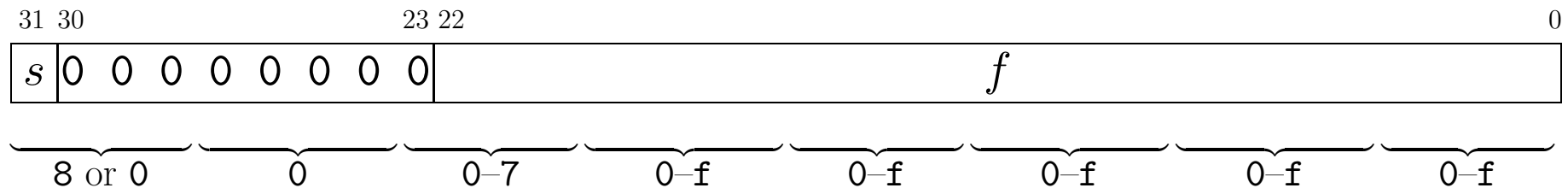
$$x - y = 0.00010 \dots 0 \times 2^{-126}$$

(**underflow** after normalization)

- ▷ If the underflowed result is “flushed” to zero, then the statement $z=1./(x-y)$ results in division by zero, even though x and y compare unequal!

FLOATING-POINT REPRESENTATIONS (28)

- **Denormalized** floating-point numbers (IEEE-754 single precision):



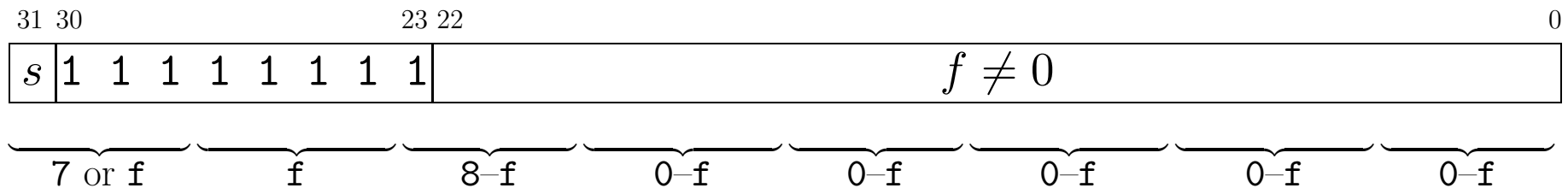
- Value assigned to a denormalized number:

$$d = (-1)^s 0.f \times 2^{-126}$$

- ▷ No implicit 1 bit
- ▷ Purpose: *gradual* loss of significance for results that are too small to represent in normalized form

FLOATING-POINT REPRESENTATIONS (29)

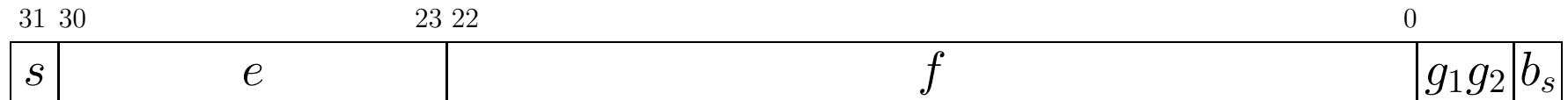
- NaNs in IEEE-754 single precision:



- The following operations generate the value NaN:
 - ▷ Addition: $\text{Inf} + (-\text{Inf})$
 - ▷ Multiplication: $0 \times \text{Inf}$
 - ▷ Division: $0/0$ or Inf/Inf
 - ▷ Computation of a remainder (REM): $x \text{ REM } 0$ or $\text{Inf REM } x$
 - ▷ Computation of a square root: $\sqrt{x}$ when $x < 0$
- Purpose: Allow computation to continue

FLOATING-POINT REPRESENTATIONS (30)

- **Rounding** in IEEE-754 single precision:



- ▷ Keep 2 additional (“guard”) bits, plus a “sticky bit”
- ▷ Common rounding modes:
 - Truncate
 - ◇ Round toward 0
 - ◇ Round toward $-\text{Inf}$
 - Round to nearest, with tie-breaking when $g_1 = 1$ and $g_2 = b_s = 0$
 - ◇ Tie-breaking method 1: Round up (biases result)
 - ◇ Tie-breaking method 2: Round so that bit 0 of the result is 0 (“round to even”; unbiased)

FLOATING-POINT REPRESENTATIONS (31)

- **Catastrophic cancellation** in subtraction:

- ▷ Occurs when the relative error of the result is large compared to machine epsilon:

$$\left| \frac{[\text{round}(x) \ominus \text{round}(y)] - \text{round}(x - y)}{\text{round}(x - y)} \right| \gg \epsilon_{\text{mach}}$$

- ▷ Example ($\beta = 10$, $p = 3$, round to even):

- Suppose we have computed results $x = 1.005$, $y = 1.000$
- Then $\text{round}(x) = 1.00$, $\text{round}(y) = 1.00$ but $\text{round}(x - y) = 5.00 \times 10^{-3}$
- Relative error of result is $1 \gg \epsilon_{\text{mach}}$

- This is the main reason for using double precision