



# CMPT125 TUTORIAL 4: BINARY REPRESENTATION

TA: Jamal ([jrahim@sfu.ca](mailto:jrahim@sfu.ca))

# BINARY ENCODING OF INTEGERS

SFU

- The values of digits in a number are positional:
- Decimal numbers:  $582 = 500 + 80 + 2 = 5*10^2 + 8*10^1 + 2*10^0$
- Binary numbers:  $10110 = 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$
- Exercises:
  - Convert 10011011 from binary to decimal
  - Convert 29 from decimal to binary

# BINARY ENCODING OF INTEGERS

SFU

- Convert 10011011 from binary to decimal:
  - $2^7 + 2^4 + 2^3 + 2^1 + 2^0 = 128 + 16 + 8 + 2 + 1 = 155$
- Convert 29 from decimal to binary:
  - 29 is odd, hence the binary should end with 1. \*\*\*\*\*1
  - Let's subtract 1, we are left with 28.
  - Let's divide 28 by 2, and get 14.
  - 14 is even, so the next digit will be 0.... -> \*\*\*\*\*01
  - Let's divide 14 by 2, we get 7.
  - 7 is odd, so the next digit is 1 -> \*\*\*\*101

- Simple data types are usually fixed in width:
  - `int` is usually 4 bytes = 32 bits. Hence, its range is  $[-2^{31}, 2^{31}-1]$  (one of the digits is reserved for the sign). Larger numbers will result in an overflow.
  - `long int` is usually 8 bytes
  - `char` is 1 byte
  - `float` is usually 4 bytes, `double` 8 bytes, and `long double` 12 bytes

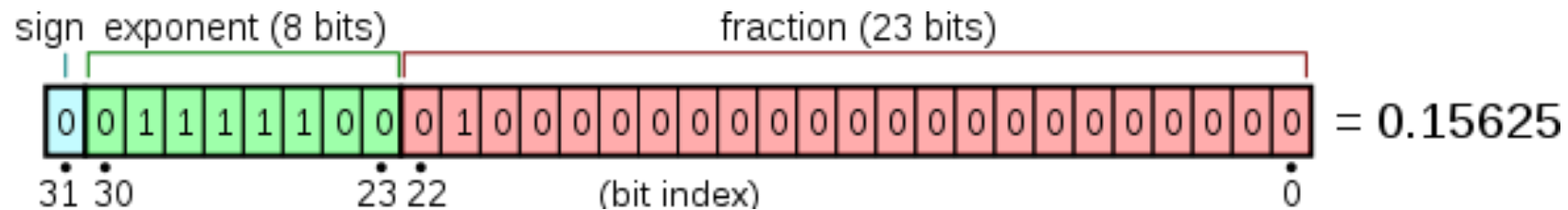
# FLOATING POINT ENCODING

SFU

- Scientific notation conventions to express number by their magnitude is used for binary. Examples:
  - Speed of light =  $2.99792458 \times 10^8$  m/s
  - One gigabyte =  $1.073741824 \times 10^9$  bytes
  - $\frac{1}{3} = 3.33333333333 \times 10^{-1}$

# FLOATING POINT ENCODING

SFU

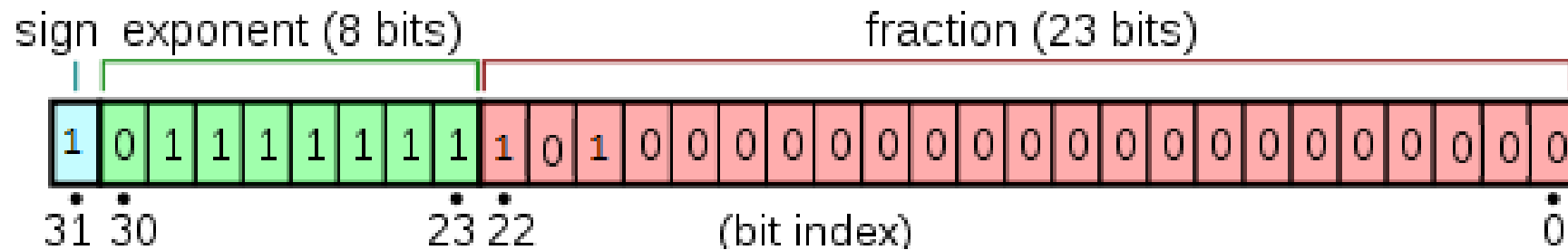


- Floating point composed of 4 bytes = 32 bits:
  - 1 bit for sign ( 0 – positive, 1 – negative)
  - 23 bits for the significand (*significant digits* of the number)  $1.b_{22}b_{21}\dots b_0$  :
    - $b_{22}$  represents the digit of  $\frac{1}{2}$
    - $b_{21}$  represents the digit of  $\frac{1}{4}$  ...
  - 8 bits for the exponent – ranges from -127 to 128
  - Range of the representation:  $1.b_{22}b_{21}\dots b_0 \times 2^{(\text{exp}-127)}$ 
    - between  $\approx 2^{128} \approx 3.40 \times 10^{38}$  and  $\approx 2^{-127} \approx 1.17 \times 10^{-38}$

**The number is  $(-1)^{\text{sign}} \times (1 + \text{fraction}) \times 2^{\text{exp}-127}$**

# FLOATING POINT ENCODING – EXAMPLE

SFU



- $5/8 = 1/2 + 1/8$
- Set 1 for the minus sign
- Set  $b_0 = 1$  (for  $1/2$ ) and  $b_2 = 1$  (for  $1/8$ )
- The 8 bits of exponent are 127
- $-1.625 = - (1 + 1/2 + 1/8) \times 2^0$

# EXERCISES

SFU

1. Convert 94 and 60 to binary.
2. Convert 10001 and 1010000 to Decimal.
3. Play around with the provided code and try to understand the concepts:
  - i. enc.c in the zip file has a function that shows how int is actually represented in the memory (least significant digits go first)
  - ii. large\_int.c show what happens if you start from INT\_MAX and increase it by 1.
  - iii. look at different float values in debug mode in VS code. For example, 0.2 will have precision issues, but 0.25 will be perfectly represented. This is because float are represented using  $1/2$ ,  $1/4$ ,  $1/8$ ... so you can't represent 0.2 perfectly.