Computer Science Basics
Floating Point Numbers

Fall Term 2025/2026

Emmanuel Benoist | BFH-TI

Floating Point Numbers

Last Week

Representing negative numbers

- **Numbers are stored in binary in computers**
  - Easy for positive numbers
- **Two's Complement Notation**
  - Works for positive and negative numbers
  - Uses the same addition for signed and unsigned numbers
- **How to represent real numbers?**
  - Topic of this course.

Floating points

At the beginning of computer science, computer only provided integer arithmetic. But, for scienfical uses, it was needed to allows programmers to use "real numbers". The data types used to model $\mathbb{R}$, the real number of mathematicians, have special properties that every programmer should know. Most of programming languages (and computer hardware) now provide the IEEE 754 norm.

Unsigned integer with bias

Unsigned integer with bias

- **How to represent numbers between -127 and 128?**
  - First solution : 2's complement (seen last week)
- **Or write a positive number**
  - If $-127 \leq x \leq 128$
  - then $0 \leq (x + 127) \leq 255$
  - $(x + 127)$ can be represented on one byte.
- **Examples**
  - x=0 is represented by 127+x = 127
    0 is represented by 0x7F
  - x = 5 is represented by 127 + x = 132
    5 is represented by 0x84
  - 0x90 = 144 represents 144-127 = 17
  - 0x79 = 121 represents 121 - 127 = -6

Floating point binary number

Convert decimal into binary

- **For integers**
  Divide by 2, and each time remember the remainder
- **Example: 543**

```
543   |  1
271   |  1
135   |  1
67    |  1
33    |  1
16    |  0
8     |  0
4     |  0
2     |  0
1     |  1
```

543 = 1000011111B

Convert decimal into binary (Cont.)

- **For floating points numbers**
  For the decimal part: multiply each time by two.
- **Example: write 0.25 in binary**

```
0.25 | 0.
0.5  | 0
1    | 1
```

0.25 = 0b 0.01

Convert decimal into binary (Cont.)

- **Example 2: write 0.1 in binary**

```
0.1 | 0.
0.2 | 0
0.4 | 0
0.8 | 0
1.6 | 1 -> 0.6
1.2 | 1 -> 0.2
0.4 | 0
0.8 | 0
1.6 | 1 -> 0.6
1.2 | 1 -> 0.2
...
```

$$0.1 = 0b0.0001100110011\overline{0011}00$$

Fixed point numbers

Fixed point numbers

How to represent the decimal number 41.13 with fixed point arithmetic ? Remember that every number in base b can be written as:

$$x = \sum_{i=-\infty}^{\infty} d_i b^i$$

where the d_i are the digits of the numbers and b the base. So the number $(41.13)_{10}$ may be written as the sum:

$$41.13 = 4 * 10^1 + 1 * 10^0 + 1 * 10^{-1} + 3 * 10^{-2}$$

Binary fixed point numbers

As the number 41.13 exist independantly of its representation, it should be possible to write this number using the binary notation.

1. First code the integer part of the number in binary. The decimal value 41 is written $(101001)_2$.
2. Compute the decimal part of the number. The value 0.13 is written as $(0.0010000101000111110\ldots)_2$

Finally one have that:

$$(41.13)_{10} = (101001.0010000101000111110\ldots)_2$$

Some mathematical considerations

- A rationnal number may be written written either with a finite decimal form or with a periodic form. The property **"to be rational"** is independant of the base used to write the number.
- A rational number may have a finite representation in one base and a periodic infinite representation in another base. For example, the number (1/3) has an infinite periodic represention in base 10, but may be written as $(0.1)_3$ in base 3. The decimal number (0.1) has an infinite periodic representation in base 2. $(0.1)_{10} = (0.0\overline{0011})_2$
- Irrational numbers have infinite and non-periodic form. Theses numbers does not have a finite representation **in any** base.

Floating points

Floating points

- The main drawback of fixed point numbers is that their size depends on their magnitude and their precision. But, in engineering, most of the time, one can reduce the precision as the magnitude augment. Engineers use mostly the concept of ***significant digits***.
- This notation is usually known as "engineering notation" in pocket calculators. For example one can write :
  - $1.344 \cdot 10^4$ for the number 13440
  - $2.342 \cdot 10^{-5}$ for the number 0.00002342
  - $4.430 \cdot 10^0$ for the number 4.430

  All theses number have 4 significant digits (the mantissa). The "scaling" is done by the exponent of 10. The mantissa is always a number between 1.000 and 9.999.

IEEE 754 representation

The norm IEEE 754 used for binary representation of floating point numbers is based on the same idea.

- the mantissa is a number between 1.0 and 2.0 (smaller than 2.0). The number of significant digits is given by the type of floating point numbers used (`float`, `double`)
- The exponent is now an exponent of 2. It may take positive or negative values.
- The norm introduces special values (negative infinity, positive infinity, not a number, zero) that would be studied later in this chapter.
- The IEEE 754 norm introduce also rounding rules for numbers.

Floating numbers : a first example

For this example, the number is coded on 32 bits, which correspond to the `float` format of Java for example.
How would be the decimal number 0.5 represented ?
$(0.5)_{10} =$ 0 0111 1110 0000 0000 0000 0000 0000 000

- The first bit (red) is the sign bit. Its value is 0 for positive number and 1 for negative numbers.
- The second group (blue) is the exponent. Here one have the value -1. The exponent is coded as an unsigned integer with a bias. The value of the exponent is computed by the formula valueOfTheField - 127.
- The third field contains the mantissa which here is 0 (there is a hidden bit!)

The exponent

- As explained above, the exponent is coded as an unsigned integer (no two-complement) with a bias which correspond to the half of the magnitude of the exponent. For example if the exponent is coded on 8 bits, its magnitude is $2^8 = 256$ and the bias would be $2^{8-1} - 1 = 127$
- Some values of the exponent are reserved to represent special values of numbers. These values are $0x00$ and $0xFF$.

The mantissa

- As explained before the matissa contain a value into the interval $1 \leq$ mantissa < 2. Remark that the value 2 does not belong the the intervall.
- On this interval, one may see that the first bit is always 1 and therefore is not represented explicitly (it is called the **hidden bit**). The norm IEEE 754 define a special format (called denormalized numbers) to change this behavior.
  This hidden bit cause the value of the mantissa to 0 on the example above.
- The bits of the mantissa represent the sum of negative power of 2.

Special values

As it was mentionned above, the norm IEEE 754 introduced some special values.

Exponent	Mantissa	Value	Description
$0x00$	= 0	0	Zero
$0x00$	$\neq 0$	$\pm 0.m \cdot 2^{-126}$	Denormalized
$0x01$ to $0xFE$	any	$\pm 1.m \cdot 2^{e-127}$	Normalized
$0xFF$	= 0	$\pm\infty$	$\pm$ Infinity
$0xFF$	$\neq 0$	NaN	Not a Number

Remark : There are two possible representations for the value 0 (+0 and -0)

Comments on special values

- The value zero needs a special representation as it is not possible to represent it using the standard model (see example above)
- Denormalized values are not provided by standard programming language. Be extremly careful if you need them (perfomance issues)
- Infinity represents numbers whose magnitude may not be represented into the model. Arithmetical operations where one operand is infinity are well defined by IEEE norm (see next slide).
- Operation where an operand has the value NaN cause an error.

Operation with infinity

Operation	Result
$x/\pm\infty$	0
$\pm\infty \cdot \pm\infty$	$\pm\infty$
$\pm$ non zero / 0	$\pm\infty$
$\infty + \infty$	∞
$\pm 0/\pm 0$	NaN
$\infty - \infty$	NaN
$\pm\infty/\pm\infty$	NaN
$\pm\infty \cdot 0$	NaN

Different type of floating points

The norm IEEE 754 defines two types of floating points : single precision and double precision. They differ only by the number of bits used to store them

	Sign	Exponent	Mantissa	Bias
Single precision	1	8	23	127
Double precision	1	11	52	1023

Rounding

IEEE standard has four different rounding modes. The first is the default, the others are called *directed rounding*.

- **Round to nearest** : rounds to the nearest value. If the number fall in the midway it is rounded to the nearest value with an even (zero) least significant bit.
- **Round toward o** - directed rounding towards o
- **Round toward** $+\infty$
- **Round toward** $-\infty$

Example of rounding

Rounding mode	+11.5	+12.5	-11.5	-12.5
to nearest, ties to even	+12.0	+12.0	-12.0	-12.0
to nearest, ties away from zero	+12.0	+13.0	-12.0	-13.0
toward o	+11.0	+12.0	-11.0	-12.0
toward $+\infty$	+12.0	+13.0	-11.0	-12.0
toward $-\infty$	+11.0	+12.0	-12.0	-13.0

Conclusion

Conclusion

- **IEEE 754 has three parts**
  - sign bit
  - Exponent
  - Mantisse
- **First: convert the number in binary**
  - $10,5 = 0b1010,1$
- **Then : Write the number in "binary-scientific" notation.**
  - $10,5 = 0b1,0101 \times 2^3$
  - mantisse = ob 1,0101 , exponent = 3, sign=+
- **Use IEEE 754 notation**
  - Sign : + is 0; - is 1
  - Mantisse without the first 1 is 010 1000 0000 0000 0000 0000
  - Expenent +127 is 130=ob 1000 0010
  - Representation ob 0**100 0001 0** *010 1000 0000 0000 0000 0000* 0x41280000