Data representations in computer

UniCode

- Bit means "binary digit" and is the smallest unit of computerized data.
- A bit is a 2-base number, i.e. it has either the value of 0 or 1.
- A byte is an amount of memory, a certain collection of bits, originally variable in size but now almost always eight bits.

byte =	1	2	3	4	5	6	7	8
	bit	bit	bit	bit	bit	bit	bit	bit
	1|0	1|0	1|0	1|0	1|0	1|0	1|0	1|0

Some example bytes could be 00000001 or 11111111 or 01010011.

byte =	1	2	3	4	5	6	7	8
	128 (27)	**64** (26)	**32** (25)	**16** (24)	**8** (23)	**4** (22)	**2** (21)	**1** (20)
	1|0	1|0	1|0	1|0	1|0	1|0	1|0	1|0

The calculation of the decimal equivalent of the binary value 00000001:

byte =	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**	
	0	0	0	0	0	0	0	1	= 1

UniCode

- The calculation of the decimal equivalent of the binary value 11111111:

byte =	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**	
	1	1	1	1	1	1	1	1	= 128+64+32+16+8+4+2+1 = 255

- The calculation of the decimal equivalent of the binary value 01010011:

byte =	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**	
	0	1	0	1	0	0	1	1	= 64+16+2+1 = 83

ASCII

- **ASCII** stands for American Standard Code for Information Interchange.
- It is a standard for assigning numerical values to the set of letters in the Roman alphabet and typographic characters.
- The ASCII character set can be represented by 7 bits. This makes 27 or 128 different values resp. characters
- As ASCII uses only 7 of the 8 bits available of an byte the first bit is always 0: 0xxxxxxx;

dec	bin	char	dec	bin	char
0	00000000	NUL (null)	64	01000000	@
1	00000001	SOH (start of heading)	65	01000001	A
2	00000010	STX (start of text)	66	01000010	B
3	00000011	ETX (end of text)	67	01000011	C
4	00000100	EOT (end of transmission)	68	01000100	D
5	00000101	ENQ (enquiry)	69	01000101	E
6	00000110	ACK (acknowledge)	70	01000110	F
7	00000111	BEL (bell)	71	01000111	G
8	00001000	BS (backspace)	72	01001000	H
9	00001001	TAB (horizontal tab)	73	01001001	I
10	00001010	LF (NL line feed, new line)	74	01001010	J
11	00001011	VT (vertical tab	75	01001011	K

- The first 32 characters are for control characters.

UniCode

- ASCII was an American-developed standard, so it only defined unaccented characters. There was an ‘e’, but no ‘é’ or ‘Í’.
- This meant that languages which required accented characters couldn’t be represented in ASCII.
- Unicode started out using 16-bit characters instead of 8-bit characters. 16 bits means you have 2^16 = 65,536 distinct values available.
- This made it possible to represent many different characters from many different alphabets

UniCode

- UTF-8 will encode a character with a single byte.
- UTF-16 will encode a character with a two bytes
-

character	encoding	bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001
あ	UTF-8	11100011 10000001 10000010
あ	UTF-16	00110000 01000010
あ	UTF-32	00000000 00000000 00110000 01000010

- Same binary data different interpretation by different encodings

bits	encoding	characters
11000100 01000010	Windows Latin 1	ÄB
11000100 01000010	Mac Roman	ƒB
11000100 01000010	GB18030	腂

UniCode

- A string of ASCII text is also valid UTF-8 text.
- UTF-8 uses the following rules:
- If the code point is < 128, it's represented by the corresponding byte value.
- If the code point is >= 128, it's turned into a sequence of two, three, or four bytes, where each byte of the sequence is between 128 and 255.
- Latin-1, also known as ISO-8859-1, is a similar encoding.
- Unicode code points 0–255 are identical to the Latin-1 values, so converting to this encoding simply requires converting code points to byte values; if a code point larger than 255 is encountered, the string can't be encoded into Latin-1.

```
In [59]: ord("A")
Out[59]: 65

In [60]: ord("a")
Out[60]: 97

In [61]: chr(50)
Out[61]: '2'

In [62]: chr(70)
Out[62]: 'F'
```

- One-character Unicode strings can also be created with the chr() built-in function, which takes integers and returns a Unicode string of length 1 that contains the corresponding code point.
- The reverse operation is the built-in ord() function that takes a one-character Unicode string and returns the code point value

Detecting Encodings

- **Chardet library:**
- Character encoding auto-detection in Python.

```
In [99]: from chardet.universaldetector import UniversalDetector

In [100]: moviefile=open(moviefilename,"rb")

In [101]: for line in moviefile:
     ...:     detector.feed(line)
     ...:     if detector.done: break
     ...: detector.close()
     ...:
Out[101]: {'confidence': 0.73, 'encoding': 'ISO-8859-1', 'language': ''}
```

- Latin-1 is also known as ISO-8859-1

Integer representation

- **Unsigned Integers:**
- Unsigned integers can represent zero and positive integers, but not negative integers.

byte =	128	64	32	16	8	4	2	1	
	1	1	1	1	1	1	1	1	= 128+64+32+16+8+4+2+1 = 255

n	Minimum	Maximum
8	0	(2^8)-1 (=255)
16	0	(2^16)-1 (=65,535)
32	0	(2^32)-1 (=4,294,967,295) (9+ digits)
64	0	(2^64)-1 (=18,446,744,073,709,551,615) (19+ digits)

Integer representation

- **Signed Integers:**
- Signed integers can represent zero, positive integers, as well as negative integers.
- Three representation types:

➢ Sign-Magnitude representation

➢ 1's Complement representation

➢ 2's Complement representation : Modern method

- the most significant bit (msb) is the *sign bit*, with value of 0 representing positive integers and 1 representing negative integers.

The remaining *n*-1 bits represents the magnitude of the integer, as follows:

- for positive integers, the absolute value of the integer is equal to "the magnitude of the (*n*-1)-bit binary pattern".
- for negative integers, the absolute value of the integer is equal to "the magnitude of the *complement* of the (*n*-1)-bit binary pattern *plus one*" (hence called 2's complement).

Integer representation

- 2's Complement representation
- Example 1: Suppose that *n*=8 and the binary representation 0 100 0001B
- Sign bit is 0 ⇒ positive

Absolute value is 100 0001B = 65D

Hence, the integer is +65

- Example 2: Suppose that *n*=8 and the binary representation 1 000 0001B.

Sign bit is 1 ⇒ negative

Absolute value is the complement of 000 0001B plus 1, i.e., 111 1110B + 1B = 127D

Hence, the integer is -127D

- Example 3: Suppose that *n*=8 and the binary representation 0 000 0000B.

Sign bit is 0 ⇒ positive

Absolute value is 000 0000B = 0D

Hence, the integer is +0D

- Example 4: Suppose that *n*=8 and the binary representation 1 111 1111B.

Sign bit is 1 ⇒ negative

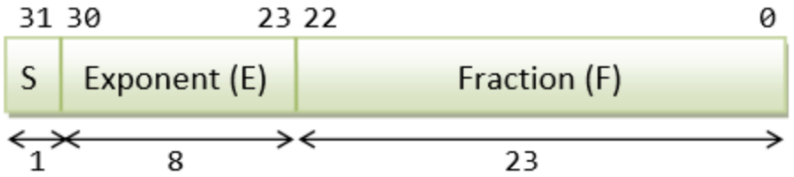
Absolute value is the complement of 111 1111B plus 1, i.e., 000 0000B + 1B = 1D

Hence, the integer is -1D

Floating Point representation

Normalized form

1 1000 0001 011 0000 0000 0000 0000 0000



32-bit Single-Precision Floating-point Number

- S = 1 (negative or positive)
- E = 1000 0001
- F = 011 0000 0000 0000 0000 0000

- N = (-1)^S × 1.F × 2^(E-127)

we need to represent both positive and negative exponent.
With an 8-bit E, ranging from 0 to 255, the excess-127 scheme could provide actual exponent of -127 to 128

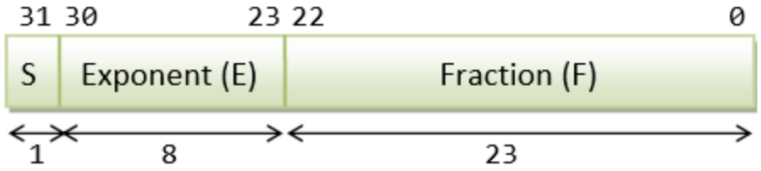
Fraction part: 1. 011 0000 0000 0000 0000 0000B
Here at "1" at the beginning is implicit
Fraction: $1 + 1 \times 2^{-2} + 1 \times 2^{-3} = 1.375D$.

Exponent part: 1000 0001B=129
So the number is -1.375×2^2=-5.5D

Floating Point representation

De-Normalized form: In normalized form implicit leading 1 for the fraction, it cannot represent the number zero!

•For E=0, the numbers are in the de-normalized form.
•An implicit leading 0 (instead of 1) is used for the fraction; and the actual exponent is always -126. Hence, the number zero can be represented with E=0 and F=0 (because 0.0×2^-126=0).

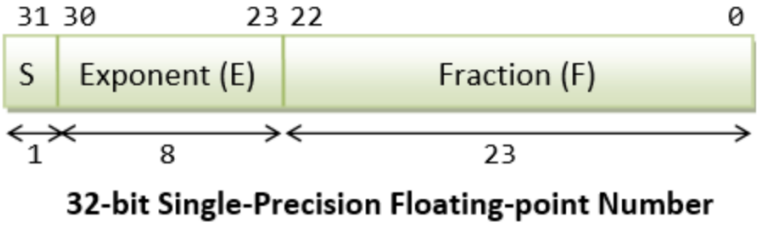


32-bit Single-Precision Floating-point Number

We can also represent very small positive and negative numbers in de-normalized form with E=0

For example, if S=1, E=0, and F=011 0000 0000 0000 0000 0000.
The actual fraction is 0.011=1×2^-2+1×2^-3=0.375D.
Since S=1, it is a negative number.
With E=0, the actual exponent is -126.
Hence the number is -0.375×2^-126 = -4.4×10^-39, which is an extremely small negative number (close to zero).

Floating Point representation



32-bit Single-Precision Floating-point Number

In summary:

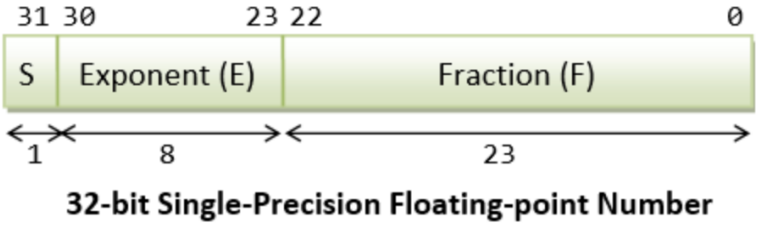
For $1 \leq E \leq 254$, $N = (-1)^S \times 1.F \times 2^{(E-127)}$.

- These numbers are in the so-called normalized form.
- The sign-bit represents the sign of the number.
- Fractional part (1.F) are normalized with an implicit leading 1.
- The exponent is bias (or in excess) of 127, so as to represent both positive and negative exponent.
- The range of exponent is -126 to +127

•For $E = 0$, $N = (-1)^S \times 0.F \times 2^{(-126)}$.
These numbers are in the so-called denormalized form.
The exponent of 2^-126 evaluates to a very small number.
Denormalized form is needed to represent zero (with F=0 and E=0).
It can also represents very small positive and negative number close to zero.

For E = 255, it represents special values, such as ±INF (positive and negative infinity) and NaN (not a number).

Floating Point representation



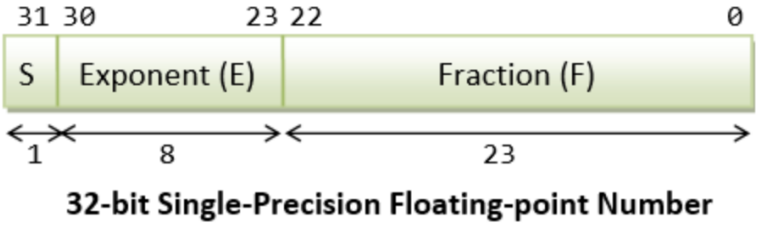
32-bit Single-Precision Floating-point Number

Example:

0 10000000 110 0000 0000 0000 0000 0000

- Sign bit S = 0 ⇒ positive number
- E = 1000 0000B = 128D (in normalized form)
- Fraction is 1.11B (with an implicit leading 1) = 1 + 1×2^-1 + 1×2^-2 = 1.75D
- The number is +1.75 × 2^(128-127) = +3.5D

Floating Point representation



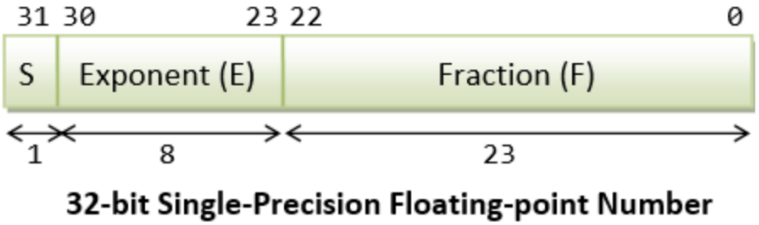
32-bit Single-Precision Floating-point Number

Example:

1 01111110 100 0000 0000 0000 0000 0000

- Sign bit S = 1 ⇒ Negative number
- E = 01111110B = 126D (in normalized form)
- Fraction is 1.1B (with an implicit leading 1) = 1 + 2^-1 = 1.5D
- The number is -1.5 × 2^(126-127) = -0.75D

Floating Point representation



32-bit Single-Precision Floating-point Number

Example:

1 00000000 000 0000 0000 0000 0000 0001

- E = 0 (in de-normalized form)
- Fraction is 0.000 0000 0000 0000 0000 0001B (with an implicit leading 0) = 1×2^-23
- The number is -2^-23 × 2^(-126) = -2×(-149) ≈ **-1.4**×10^-45

64-bit Double-Precision Floating-Point Numbers

64-bit Double-Precision Floating-point Number

- The most significant bit is the sign bit (S), with 0 for positive numbers and 1 for negative numbers.
- The following 11 bits represent exponent (E).
- The remaining 52 bits represents fraction (F).

- Normalized form: For $1 \leq E \leq 2046$, N = (-1)^S × 1.F × 2^(E-1023).
- Denormalized form: For E = 0, N = (-1)^S × 0.F × 2^(-1022).
- These are in the denormalized form.
- For E = 2047, N represents special values, such as ±INF (infinity), NaN (not a number)

Floating-Point Numbers Representations

Type	Sign	Exponent	Significand field	Total bits		Exponent bias	Bits precision	Number of decimal digits
Half (IEEE 754-2008)	1	5	10	16		15	11	~3.3
Single	1	8	23	32		127	24	~7.2
Double	1	11	52	64		1023	53	~15.9
x86 extended precision	1	15	64	80		16383	64	~19.2
Quad	1	15	112	128		16383	113	~34.0