Representation of numbers

course “Essentials of computing systems”

Feb.2022

contents

1. Positional numeral systems
2. Octal and hexadecimal numbers
3. Conversions between different bases
4. Negative numbers
5. Two's-complement addition
6. Floating-point numbers
7. Binary codes for decimal numbers

contents

1. Positional numeral systems
2. Octal and hexadecimal numbers
3. Conversions between different bases
4. Negative numbers
5. Two's-complement addition
6. Floating-point numbers
7. Binary codes for decimal numbers

numeral system

- A numeral system (or number system) is a writing system for expressing numbers.
- It is a notation for representing numbers of a given set, using digits or other symbols.
- Computers use the binary arithmetic system, and not the decimal (Hindu–Arabic) numeral system used by humans.
- They both are positional numeral systems, which represent any number by a sequence of juxtaposed digits.

digit and radix

- A digit is a symbol used alone or in combinations to represent numbers in a positional numeral system.
- The position of each digit has a corresponding weight.
- The mathematical value of a number is provided by the weighted sum of all its digits.
- For decimal integer numbers, the weights are powers of ten equal to 10^i (i is the position of the digit).
- The value of a decimal integer number D of the form $d_3 d_2 d_1 d_0$ is $d_3 \cdot 10^3 + d_2 \cdot 10^2 + d_1 \cdot 10^1 + d_0 \cdot 10^0$.
- $4682 = 4{\cdot}1000 + 6 \cdot 100 + 8 \cdot 10 + 2 \cdot 1$
- 10 is called the radix (or base) of the numeral system.

natural numbers

- Positional numeral systems can use any integer $r \geq 2$ for the base.
- The digit in position i has weight r^i.
- The general format of a n-digit number D in such a system is: $d_{n-1}d_{n-2}\dots d_1 d_0$.
- The value of D (natural number) is $\sum_{i=0}^{n-1} d_i \cdot r^i$.
- Digital devices adopt the binary base ($r = 2$).
- The general format of a natural number B in base 2 is $b_{n-1}b_{n-2}\dots b_1 b_0$.
- The value of B is $\sum_{i=0}^{n-1} b_i \cdot 2^i$.
- The representation for natural numbers involves a fixed number of bits (8, 16, ...), and is known as the natural binary numeral system.

decimal numbers

- The previous concepts all apply to natural numbers.
- Fractions are considered by using negative powers of a radix.
- The integer part of a number is separated from its fractional part by a radix point.
- $4682.51 = 4 \cdot 1000 + 6 \cdot 100 + 8 \cdot 10 + 2 \cdot 1 + 5 \cdot 0.1 + 1 \cdot 0.01$.
- The value of a decimal number D of the form $d_2 d_1 d_0 . d_{-1} d_{-2}$ is $d_2 \cdot 10^2 + d_1 \cdot 10^1 + d_0 \cdot 10^0 + d_{-1} \cdot 10^{-1} + d_{-2} \cdot 10^{-2}$.
- Binary fractions have a binary point.
- The general format of a number B in base b is $b_{n-1} b_{n-2} \ldots b_1 b_0 . b_{-1} b_{-2} \ldots b_{-m}$.
- The value of B is $\sum_{i=-m}^{n-1} b_i \cdot 2^i$.

least / most significant digits

- In positional numeral systems:
  - the rightmost digit is designated as least significant digit (LSD)
  - the rightmost digit is the most significant digit (MSD)
  - binary numbers, LSD is least significant bit (LSB)
  - binary numbers, MSD is most significant bit (MSB)

$$\begin{aligned}
11001_2 &= 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 &&= 25_{10} \\
111001_2 &= 1 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 1 \cdot 1 &&= 57_{10} \\
11.101_2 &= 2 + 1 + 0.5 + 0.125 &&= 3.625_{10}
\end{aligned}$$

contents

1. Positional numeral systems
2. Octal and hexadecimal numbers
3. Conversions between different bases
4. Negative numbers
5. Two's-complement addition
6. Floating-point numbers
7. Binary codes for decimal numbers

Bases 8 and 16

- Base 10 is important and useful, since humans use it in everyday activities.
- Base 2 is also relevant, as binary numbers are processed by digital devices.
- The binary notation is too verbose, so bases 8 (octal) and 16 (hexadecimal) are used instead.
- They offer a shorthand representation for binary numbers.
- The octal system uses eight different digits: 0 to 7 (0, 1, 2, 3, 4, 5, 6, 7).
- The hexadecimal numeral system uses 16 digits: 0 to 9 and the letters A to F (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Bases 8 and 16

			3-bit	hexa-	4-bit
binary	decimal	octal	string	decimal	string
0	0	0	000	0	0000
1	1	1	001	1	0001
10	2	2	010	2	0010
11	3	3	011	3	0011
100	4	4	100	4	0100
101	5	5	101	5	0101
110	6	6	110	6	0110
111	7	7	111	7	0111
1000	8	10	-	8	1000
1001	9	11	-	9	1001
1010	10	12	-	A	1010
1011	11	13	-	B	1011
1100	12	14	-	C	1100
1101	13	15	-	D	1101
1110	14	16	-	E	1110
1111	15	17	-	F	1111
10000	16	20	-	10	-
10001	17	21	-	11	-

converting from base 2 to bases 8-16

- Bases 8 and 16 are useful for representing multi-bit numbers, since they are integer powers of 2.
- Octal and hexadecimal digits are directly represented by 3-bit and 4-bit strings, respectively.
- Converting a binary number to octal is straightforward.
- Starting at the binary point and moving to the left, the bits are separated into groups of three and each group is replaced by the corresponding octal digit.
- 0s can be added to the left, since they do not change the value.

$$\begin{aligned}
100010101001_2 &= 100\;\;010\;\;101\;\;001_2 &&= 4251_8\\
100010101011_2 &= 1000\;\;1010\;\;1011_2 &&= 8AB_{16}\\
1111100011101110_2 &= 011\;\;111\;\;100\;\;011\;\;101\;\;110_2 &&= 374356_8\\
1111100011101110_2 &= 0001\;\;1111\;\;1000\;\;1110\;\;1110_2 &&= 1F8EE_{16}
\end{aligned}$$

converting from base 2 to bases 8-16

- Whenever a binary number contains digits to the right of the binary point, converting into octal/hexadecimal is also easy.
- Starting at the binary point and moving to the right, the bits are again separated in groups of three (or four) and are replaced by the corresponding octal (hexadecimal) digit.
- 0s can be added to the right of the rightmost bit, as this operation does not change the value of the number.

$$
\begin{aligned}
.11001_2 &= .110\ \ 010_2 &&= .62_8 \\
&= .1100\ \ 1000_2 &&= .C8_{16} \\
1234_8 &= 001\ \ 010\ \ 011\ \ 100_2 \\
234.05_8 &= 010\ \ 011\ \ 100.000\ \ 101_2 \\
1234_{16} &= 0001\ \ 0010\ \ 0011\ \ 0100_2 \\
AB3.05_{16} &= 1010\ \ 1011\ \ 0011.0000\ \ 0101_2
\end{aligned}
$$

contents

1 Positional numeral systems

2 Octal and hexadecimal numbers

3 Conversions between different bases

4 Negative numbers

5 Two's-complement addition

6 Floating-point numbers

7 Binary codes for decimal numbers

general conversion

- In general, the conversion from one base to another one cannot be accomplished by just replacing digits in a base to the corresponding digits in the other base.
- This only occurs when both bases are integer powers of the same number.
- The value of a number in base r, that has m digits to the right of the radix point and n digits to the left, is calculated by $D = \sum_{i=-m}^{n-1} d_i \cdot r^i$.
- The value can be calculated by converting each digit to the respective base-10 equivalent and by adding all these values.

$$
\begin{aligned}
\mathrm{AB3}_{16} &= 10 \cdot 16^2 + 11 \cdot 16^1 + 3 \cdot 16^0 &&= 2739_{10} \\
2345_8 &= 2 \cdot 8^3 + 3 \cdot 8^2 + 4 \cdot 8^1 + 5 \cdot 8^0 &&= 1253_{10} \\
124.7_8 &= 1 \cdot 8^2 + 2 \cdot 8^1 + 4 \cdot 8^0 + 7 \cdot 8^{-1} &&= 84.875_{10} \\
120.01_3 &= 1 \cdot 3^2 + 2 \cdot 3^1 + 0 \cdot 3^0 + 0 \cdot 3^{-1} + 1 \cdot 3^{-2} &&= 15.(1)_{10}
\end{aligned}
$$

general conversion

- The formula $D = \sum_{i=0}^{n-1} d_i \cdot r^i$ can be rewritten to highlight a simple procedure for converting numbers to base 10: $D = ((\ldots((d_{n-1}) \cdot r + d_{n-2}) \cdot r + \ldots) \cdot r + d_1) \cdot r + d_0$.
- When this formula is applied to hexadecimal number $A835_{16}$, the following expression can be used to calculate its value:

$$A835_{16} = (((10) \cdot 16 + 8) \cdot 16 + 3) \cdot 16 + 5$$

- The rewritten formula can be used to convert a decimal number D to the equivalent number in base r.
- If D is divided by r, d_0 is the remainder and the part inside parentheses corresponds to the quotient Q: $Q = (\ldots((d_{n-1}) \cdot r + d_{n-2}) \cdot r + \ldots) \cdot r + d_1$
- Since Q follows the same format as D, successive divisions by r yield successive digits of D, from right to left.

general conversion

$$\begin{aligned}
177 &\div 2 = 88 && \text{remainder } 1\\
88 &\div 2 = 44 && \text{remainder } 0\\
44 &\div 2 = 22 && \text{remainder } 0\\
22 &\div 2 = 11 && \text{remainder } 0\\
11 &\div 2 = 5 && \text{remainder } 1\\
5 &\div 2 = 2 && \text{remainder } 1\\
2 &\div 2 = 1 && \text{remainder } 0\\
1 &\div 2 = 0 && \text{remainder } 1
\end{aligned}$$

$$\boxed{177_{10} = 10110001_2}$$

$$\begin{aligned}
177 &\div 8 = 22 && \text{remainder } 1\\
22 &\div 8 = 2 && \text{remainder } 6\\
2 &\div 8 = 0 && \text{remainder } 2
\end{aligned}$$

$$\boxed{177_{10} = 261_8}$$

$$\begin{aligned}
177 &\div 16 = 11 && \text{remainder } 1\\
11 &\div 16 = 0 && \text{remainder } 11
\end{aligned}$$

$$\boxed{177_{10} = B1_{16}}$$

general conversion

- The methods described can be followed to directly convert any number in a given base to any other base.
- Often, it is easier to first convert to the decimal base and then to the target base.
- Convert 3410_5 to base 3:

$$3410_5 = 3 \cdot 5^3 + 4 \cdot 5^2 + 1 \cdot 5^1 + 0 \cdot 5^0 = 480_{10}$$

$$\begin{aligned} 480 \div 3 &= 160 && \text{remainder } 0 \\ 160 \div 3 &= 53 && \text{remainder } 1 \\ 53 \div 3 &= 17 && \text{remainder } 2 \\ 17 \div 3 &= 5 && \text{remainder } 2 \\ 5 \div 3 &= 1 && \text{remainder } 2 \\ 1 \div 3 &= 0 && \text{remainder } 1 \end{aligned}$$

$$\boxed{3410_5 = 122210_3}$$

contents

1. Positional numeral systems
2. Octal and hexadecimal numbers
3. Conversions between different bases
4. Negative numbers
5. Two's-complement addition
6. Floating-point numbers
7. Binary codes for decimal numbers

representations

- Representing signed numbers, together with positive ones, requires additional issues to be addressed, namely the inclusion of a sign bit.
- There are many ways to represent negative numbers and four alternatives are next discussed:
    1. sign-magnitude
    2. one's-complement
    3. two's-complement
    4. excess representations

sign-magnitude

- The most intuitive method, sign-magnitude, uses:
    - the MSB for the sign bit
    - the remaining bits for the magnitude of the number (its absolute value).
- By convention, a ‘1’ in the sign bit indicates a negative number, whereas a ‘0’ indicates a positive number (or zero).
- In a 8-bit representation, two symmetrical values just differ in the sign bit, as next illustrated:

$$+120_{10} = 01111000_2$$
$$-120_{10} = 11111000_2$$

- It is mandatory to know how many bits are used to represent the numbers.
- The number 1100_2 is negative if four bits are used, but positive if instead six bits are used.

sign-magnitude

- A disadvantage of this method is the existence of two possible representations of zero (“+0” and “-0”).
- The sign-magnitude contains the same number of positive and negative numbers.
- A sign-magnitude integer representation with n bits ranges from -(2^{n-1}-1) to +(2^{n-1}-1).
- With 8 bits, this range goes from -127 to +127.
- Signed-magnitude calculations are performed basically with the same method as humans use with pencil and paper.
- As an example, consider the rules for addition:
  1. If the signs are the same, add the magnitudes and use that same sign for the result;
  2. If the signs differ, determine which operand has the larger magnitude. The sign of the result is the same as the sign of the operand with the larger magnitude, and the magnitude must be obtained by subtracting the smaller one from the larger one.

one's-complement

- Complement numeral systems make additions/subtractions faster and easier.
- Let us analyse how this approach generally works for decimals.
- One decimal number can be subtracted from another by adding the difference of the subtrahend from all nines and adding back a carry.
- This is called taking the nine's complement of the subtrahend (or its diminished radix complement).
- Assume that one wants to calculate 165-43.
- The difference of 43 from 999 is 956 and, in nine's complement arithmetic, 165-43 = 165+956 = (1)121.
- The "carry" from the hundreds column is added back to the units place, yielding the correct result 165-43 = 121+1 = 122.

one's-complement

- This process works similarly in the one's-complement numeral system.
- The one's-complements of binary numbers are computed the same way as for natural numbers, except that the weight of the MSB is $-2^{n-1}+1$ instead of 2^{n-1}.

$$\begin{aligned} 011001_2 &= 1\cdot 16 + 1\cdot 8 + 1\cdot 1 &&= 25_{10} \\ 111001_2 &= 1\cdot(-31) + 1\cdot 16 + 1\cdot 8 + 1\cdot 1 &&= -6_{10} \end{aligned}$$

- Given a number V in base r having n digits, the diminished radix complement of V is $r^n - 1 - V$.
- The range of representable integer numbers with n bits in the diminished radix complement representation is the same as for sign-magnitude one.

one's-complement

- For a binary number, its one's-complement is obtained by subtracting from all ones.
- The one's-complement of 0101_2 is 1111-0101 $= 1010_2$.
- The one's complement of a binary number can be obtained by just toggling all the bits.
- Complement notation simplifies subtraction by turning it into addition.
- Additionally, it provides a method to represent negative numbers.
- The idea is that a negative number needs to be converted to its complement, which should have a '1' in the MSB.
- Positive numbers, which have a '0' in the MSB, are used as is.

$$
\begin{aligned}
+24_{10} &= +(00011000_2) = 00011000_2 \\
-24_{10} &= -(00011000_2) = 11100111_2 \\
-10_{10} &= -(00001010_2) = 11110101_2
\end{aligned}
$$

one's-complement

- To subtract 10 from 24, one needs first to express the subtrahend (10) in one's-complement and then add it to the minuend (24).
- This effectively adds -10 to 24.
- The MSB will have a '0' or a '1' carry, which needs to be added to the LSB of the sum.
- This operation is designated end carry-around and results from the use of the diminished radix complement, in this case the one's-complement.

one's-complement

$$\begin{array}{r cccccccc c r}
1 \leftarrow & 1 & 1 & 1 & & & & & & \Leftarrow & \text{carries} \\
 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & & (+24_{10}) \\
 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & & +(-10_{10}) \\ \cline{2-9}
 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & & \\
 & & & & & & & + & 1 & & \\ \cline{2-9} \cline{11-11}
 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & & (+14_{10})
\end{array}$$

$$\begin{array}{r cccccccc c r}
0 \leftarrow & & & & 1 & 1 & 1 & & & \Leftarrow & \text{carries} \\
 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & & (+10_{10}) \\
 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & & +(-24_{10}) \\ \cline{2-9}
 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & & \\
 & & & & & & & + & 0 & & \\ \cline{2-9} \cline{11-11}
 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & & (-14_{10})
\end{array}$$

- The range of representable integer numbers with n bits is -($2^{n\text{-}1}$-1) through +($2^{n\text{-}1}$-1).
- With 8 bits, this range goes from -127 to +127.

two's-complement

- Most computers use the two's-complement numeral system.
- The radix complement is considered more intuitive than the diminished radix complement.
- Given a numeric value V in base r having n digits, the radix complement of V is defined to be r^n-V for $V \neq 0$, and 0 for $V = 0$.
- With three decimal digits, the ten's complement of 43 is 957 (10^3-43).
- The decimal value for a two's-complement number is computed the same way as for a natural number, except that the weight of the MSB is -$2^{n\text{-}1}$ instead of +$2^{n\text{-}1}$.

$$\begin{aligned} 011001_2 &= 1 \cdot 16 + 1 \cdot 8 + 1 \cdot 1 &&= 25_{10} \\ 111001_2 &= 1 \cdot (-32) + 1 \cdot 16 + 1 \cdot 8 + 1 \cdot 1 &&= -7_{10} \end{aligned}$$

two's-complement

- Two's complement is just one's complement incremented by 1.
- To find the two's complement of a binary number, one just needs to flip bits and add 1.
- Any carries involving the MSB are simply discarded.
- Only negative numbers need to be converted to two's complement notation.

$$
\begin{aligned}
&+24_{10} = +00011000_2 = 00011000_2\\
&-24_{10} = -00011000_2 = 11100111_2 + 1 = 11101000_2\\
&-10_{10} = -00001010_2 = 11110101_2 + 1 = 11110110_2
\end{aligned}
$$

two's-complement

$$
\begin{array}{rccccccccccr}
1 \leftarrow & 1 & 1 & 1 & & & & & & & \Leftarrow & \text{carries} \\
& 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & & & (+24_{10}) \\
& 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & & & +(-10_{10}) \\
\cline{2-9} \cline{12-12}
& 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & & & (+14_{10})
\end{array}
$$

$$
\begin{array}{rccccccccccr}
0 \leftarrow & & & & 1 & & & & & & \Leftarrow & \text{carries} \\
& 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & & & (+10_{10}) \\
& 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & & & +(-24_{10}) \\
\cline{2-9} \cline{12-12}
& 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & & & (-14_{10})
\end{array}
$$

- The discarded carry did not cause an erroneous result.
- An overflow occurs if two positive (negative) numbers are added and the result is negative (positive).
- When using two's complement notation, it is impossible to have overflow whenever adding a positive number with a negative one.

two's-complement

- Two's-complement is the preferred choice for digitally representing signed numbers.
- It does not require the signs of the operands to be checked to determine whether to add or subtract.
- It has the best representation for 0 (all 0 bits), is self-inverting, and can be extended to larger numbers of bits.
- Two's-complement numbers are added and subtracted by the same methods as unsigned numbers with the same number of bits, so the same hardware can handle numbers in both cases.
- Its biggest drawback is the asymmetry in the range of values that can be represented.
- The range of representable integer numbers with n bits is -($2^{n\text{-}1}$) through +($2^{n\text{-}1}$-1).
- With 8 bits, this range goes from -128 to +127.

excess representations

- In an excess-b representation, an n-bit pattern, whose unsigned integer value is V ($0 \leq V < 2^n$) represents the signed integer V-b, where b is the bias (or offset) of the numeral system.
- The representable numeric values range from -b to 2^n-1-b.
- With 8 bits and b=100, this range represents 256 different integer numbers, from -100 to +155.
- The main advantage lies in the fact that the all-zero pattern corresponds to the minimal negative value and the all-one pattern to the maximal positive value.
- This facilitates the comparison of values.
- It also permits to represent unbalanced sets of consecutive negative and positive numbers (e.g., from -10 to +245).
- From a mathematical point of view, the representation of natural numbers is an excess-0 representation.

different numeral systems

binary	un- signed	sign magn.	one's compl.	two's compl.	excess 3	excess 7
0000	0	0	0	0	-3	-7
0001	1	1	1	1	-2	-6
0010	2	2	2	2	-1	-5
0011	3	3	3	3	0	-4
0100	4	4	4	4	1	-3
0101	5	5	5	5	2	-2
0110	6	6	6	6	3	-1
0111	7	7	7	7	4	0
1000	8	-0	-7	-8	5	1
1001	9	-1	-6	-7	6	2
1010	10	-2	-5	-6	7	3
1011	11	-3	-4	-5	8	4
1100	12	-4	-3	-4	9	5
1101	13	-5	-2	-3	10	6
1110	14	-6	-1	-2	11	7
1111	15	-7	-0	-1	12	8

contents

1. Positional numeral systems
2. Octal and hexadecimal numbers
3. Conversions between different bases
4. Negative numbers
5. Two’s-complement addition
6. Floating-point numbers
7. Binary codes for decimal numbers

addition

- Adding values in the two's-complement numeral system requires rules similar to the ones used to add decimals.
- A difference is that the tables for binary numbers just contain 0s and 1s instead of decimal digits.
- Two decimal numbers are manually added, by adding each pair of digits at a time, starting with the LSDs of both numbers.
- If the sum is greater than 9, there is a carry that is transported to the next pair of digits.
- When adding two binary numbers $A = a_{n-1}a_{n-2} \dots a_0$ and $B = b_{n-1}b_{n-2} \dots b_0$, the same procedure is applied.
- We start by adding the LSBs a_0 and b_0, with an initial carry c_0 equal to 0.
- This results in output carry bit c_1 and the sum bit s_1.

addition

a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

addition overflow

- An addition overflows whenever the signs of the addends are the same and the sign of the sum is different from the addends' sign.

$$\begin{array}{ccccccccccc}
 & & & 1 & & & & & & \Leftarrow & \text{carries} \\
 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & & (90_{10}) \\
 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & & +(20_{10}) \\ \hline
 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & & (110_{10})
\end{array}$$

$$\begin{array}{ccccccccccc}
1 & & & & 1 & 1 & 1 & & & \Leftarrow & \text{carries} \\
 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & & (-90_{10}) \\
 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & & +(-50_{10}) \\ \hline
 & \boxed{0} & 1 & 1 & 1 & 0 & 1 & 0 & 0 & & \boxed{\text{overflow}}
\end{array}$$

contents

1 Positional numeral systems

2 Octal and hexadecimal numbers

3 Conversions between different bases

4 Negative numbers

5 Two's-complement addition

6 Floating-point numbers

7 Binary codes for decimal numbers

fixed-point numbers

- Many numeral systems assume that the radix point has a fixed position.
- Each number is a fixed-point number.
- With integers, the radix point is located on the right.
- So with five decimal digits, the values from 0 to 99,999 can be represented.
- This system represents 100 000 different numbers, with every two consecutive numbers differing by 1.
- One can consider the radix point to be in a different position.
- For example, after the third decimal digit. In this case, the values range from 0 to 999.99.
- Every two consecutive numbers differ by 0.01.
- The value of a fixed-point number is an integer that is scaled by a factor determined by the position of the radix point.

generic format

- In several calculations, this type of equally separated numbers is not useful.
- An alternative is a floating-point numeral system, whose numbers have the generic format:

$$\text{mantissa} \cdot \text{radix}^{\text{exponent}}$$

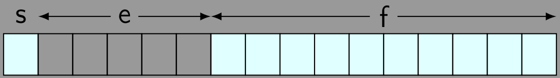
- This format is useful for performing computations involving very large numbers ($V \gg 0$), numbers very close to 0 ($V \ll 1$), and generally as an approximation to real arithmetic.
- A floating-point number is a fixed-point number, indicated by the mantissa, whose radix point position is regulated by the exponent.
- The term floating-point is related to the fact that the radix point of a number can "float".

generic format

- This format is used in the decimal scientific notation, in which numbers are expressed in two parts:
  1. a mantissa,
  2. an exponential part that indicates an integer power of ten.
- The value of the number is given by the product of these parts.
- $2.2538 \cdot 10^4$ expresses 22,538 in the scientific notation.
- Scientific notation simplifies any arithmetic calculation that deals with very large or very small numbers.
- It serves also as the basis for floating-point computation in digital computers.
- In general, floating-point representations offer more range and less precision than fixed-point ones.

sign, exponent, mantissa

- In digital computers, floating-point numbers use bits and consist of three parts:
    1. a sign bit,
    2. an exponent part (representing a power of 2),
    3. and a mantissa (or significand).
- Let us consider a 16-bit floating-point numeral system, with a sign bit, a 5-bit exponent, and a 10-bit fractional part of the mantissa.



normal numbers

- The value encoded by a bit pattern can be divided into three different cases, depending on the value of the exponent:
  1. normal numbers,
  2. subnormal numbers,
  3. special values.
- The value of a normal number (or normalised number) is calculated by: $V = (-1)^s \cdot (1 + f) \cdot 2^{e-\text{bias}}$
- The sign bit s can obviously assume two possible values.
  - If s=0, then (-1)^s=(-1)^0=+1, then V is positive.
  - If s=1, then (-1)^s=(-1)^1=-1, then V is negative.
- The other factors are always positive.

bias

- The possible values for the exponents must include both positive numbers (to represent large numbers) and negative ones (to represent small numbers).
- The exponent e is encoded in an excess format.
- The bias value is a number near the middle of the range of possible values that is selected to represent zero.
- The bias typically equals $2^{k\text{-}1}$-1, where k is the number of bits in the exponent.
- For our 16-bit floating-point numeral system, the bias is $2^{5\text{-}1}$-1=15, which is midway between 0 and 31.
- Any number larger than 15 in the exponent field represents a positive value for the real exponent.
- Values less than 15 indicate negative values.

normal numbers

- For the numeral system here discussed, the value V of a normal number is given by: $V = (-1)^s \cdot (1 + f) \cdot 2^{e-15}$
- This equation only applies for exponents from 1 to 30.
- Exponents of all zeros ($00000_2 = 0_{10}$) or all ones ($11111_2 = 31_{10}$) are reserved for zero, subnormal numbers, or special values, like infinity.

normalisation of the mantissa

- To avoid different possible representations of the same number, the mantissa $M=1+f$ must be normalised.
- This normalisation requires the mantissa M to obey the restriction $1 \leq M < 2$.
- This implies that the mantissa must always start with a nonzero bit.
- The only nonzero bit is '1', so this constant is hidden.
- This bit is also the only one that is to the left of the binary point.
- The only part of the mantissa that needs to be represented in the bit pattern is its fractional part (f).
- Since there are 10 bits for this part, the mantissas are 11 bits long (they all have a '1' to the left of the binary point).

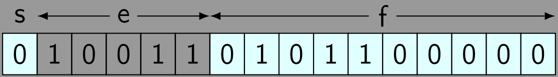
example

- Representing the decimal number $+21.5_{10}$ in the 16-bit floating-point representation:

$$+21.5_{10} = +10101.1_2 = +10101.1_2 \cdot 2^0 =$$

$$+1.01011_2 \cdot 2^4 = +1.0101100000_2 \cdot 2^{19-15}$$

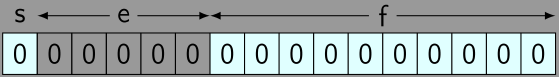
- We can fill in the 16-bit pattern:
  - $s = 0$ (indicates non-negative numbers),
  - $e = 19_{10} = 10011_2$,
  - $f = 0101100000$ ($M = 1.0101100000_2$).



s	e					f									
0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	0

subnormal numbers

- The all zeros (00000_2) exponent is reserved to represent subnormal numbers and zero.
- A subnormal number (or denormalised number) is a non-zero number with magnitude smaller than the smallest positive normal number.
- Its exponent value is fixed to be 1-bias (-14 in the example).
- The mantissa M is restricted by the condition $0 \leq M < 1$.
- For the all-zeros e exponent, the value V of a subnormal number is given by: $V = (-1)^s \cdot f \cdot 2^{e-14}$
- Since f can be all-zeros ($0000\,0000\,00_2$), this allows $V = 0$.
- The zero value is represented by an all-zeros bit pattern.



s	e					f									
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

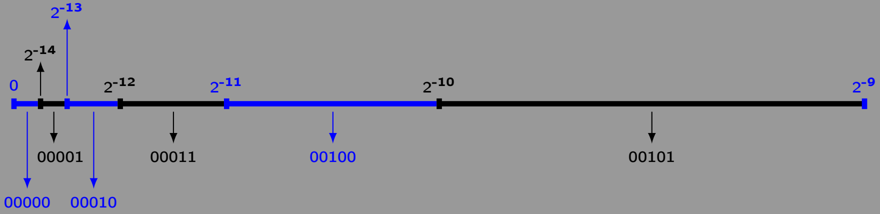
- Zero cannot be represented by a normal number. It is considered a special value in a floating-point numeral system.

special values

- The all ones ($11111_2 = 31_{10}$) exponent is reserved for other special values.
- A special value can be used to indicate an error or to represent non-initialised data.
- If the fraction field f is equal to all zeros, the value represents infinity (V=-∞, if s=1; V=+∞, if s=0).
- Infinity can represent results that overflow.
- When f is nonzero, the resulting value is a not a number (NaN).
- Such value can be the result of an operation that cannot be given as a real number or as infinity, e.g., $\sqrt{-1}$.

exponent	M = 0	M $\neq$ 0
$000\ldots00$	$V = \pm 0$	subnormal
$111\ldots11$	$V = \pm\infty$	NaN

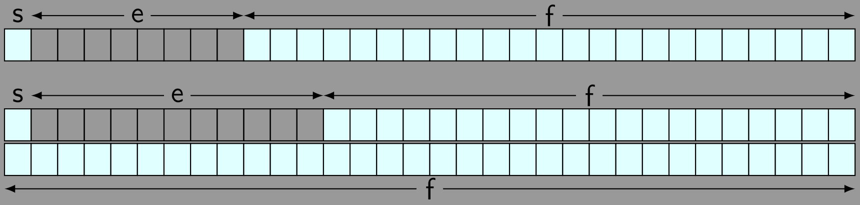
special values



exponent	interval	smallest value	2nd smallest value	largest value	distance
$00000_2=0$	2^{-14}	0	2^{-24}	$1023 \cdot 2^{-24}$	2^{-24}
$00001_2=1$	2^{-14}	2^{-14}	$(1+2^{-10}) \cdot 2^{-14}$	$(1+1023 \cdot 2^{-10}) \cdot 2^{-14}$	2^{-24}
$00001_2=1$	2^{-13}	2^{-13}	$(1+2^{-10}) \cdot 2^{-13}$	$(1+1023 \cdot 2^{-10}) \cdot 2^{-13}$	2^{-23}
$00011_2=3$	2^{-12}	2^{-12}	$(1+2^{-10}) \cdot 2^{-12}$	$(1+1023 \cdot 2^{-10}) \cdot 2^{-12}$	2^{-22}
$00100_2=4$	2^{-11}	2^{-11}	$(1+2^{-10}) \cdot 2^{-11}$	$(1+1023 \cdot 2^{-10}) \cdot 2^{-11}$	2^{-21}
$00101_2=5$	2^{-10}	2^{-10}	$(1+2^{-10}) \cdot 2^{-10}$	$(1+1023 \cdot 2^{-10}) \cdot 2^{-10}$	2^{-20}
			...		
$11100_2=28$	2^{13}	2^{13}	$(1+2^{-10}) \cdot 2^{13}$	$(1+1023 \cdot 2^{-10}) \cdot 2^{13}$	2^{3}
$11101_2=29$	2^{14}	2^{14}	$(1+2^{-10}) \cdot 2^{14}$	$(1+1023 \cdot 2^{-10}) \cdot 2^{14}$	2^{4}
$11110_2=30$	2^{15}	2^{15}	$(1+2^{-10}) \cdot 2^{15}$	$(1+1023 \cdot 2^{-10}) \cdot 2^{15}$	2^{5}

IEEE 754 floating-point standard

- The 16-bit floating-point numeral system used in this section was introduced for simplicity and conceptual understanding.
- It corresponds to the half precision form that is part of the IEEE 754 floating-point standard, published in 1985.
- This standard is also used for both single- and double-precision floating-point numbers.
- These formats correspond to the C `float` and `double` datatypes.



some important floating-point numbers

			single precision		double precision	
description	e	f	value	decimal	value	decimal
zero	00..00	0..00	0	0.0	0	0.0
smallest subn.	00..00	0..01	$2^{-23} \times 2^{-126}$	1.4×10^{-45}	$2^{-52} \times 2^{-1022}$	4.9×10^{-324}
largest subn.	00..00	1..11	$(1\text{-}\epsilon) \times 2^{-126}$	1.2×10^{-38}	$(1\text{-}\epsilon) \times 2^{-1022}$	2.2×10^{-308}
smallest norm.	00..01	0..00	1×2^{-126}	1.2×10^{-38}	1×2^{-1022}	2.2×10^{-308}
one	01..11	0..00	1×2^{0}	1.0	1×2^{0}	1.0
largest norm.	11..10	0..00	$(2\text{-}\epsilon) \times 2^{127}$	3.4×10^{38}	$(2\text{-}\epsilon) \times 2^{1023}$	1.8×10^{308}

contents

1 Positional numeral systems

2 Octal and hexadecimal numbers

3 Conversions between different bases

4 Negative numbers

5 Two's-complement addition

6 Floating-point numbers

7 Binary codes for decimal numbers

BCD

- Some binary codes exist to facilitate the process of representing decimal numbers.
- A decimal number is represented by a string of bits, where each group of contiguous bits represent one of its decimal digits.
- For example, the sequence ‘010100001001’ represents the decimal number 509.
- There are 29 059 430 400 different codes to represent the 10 decimal digits with 4 bits.
- The most obvious code is called binary-coded decimal (BCD).
- It encodes the decimal digits 0 to 9 by the respective 4-bit unsigned binary representations 0000 through 1001.
- The code patterns 1010 through 1111 are not used.
- This code allows one byte to store two decimal digits in a packed BCD representation.
- 93 is represented by 10010011.

digit	BCD	2421	2-out-of-5	biquinary	1-out-of-10
0	0000	0000	01100	0100001	1000000000
1	0001	0001	11000	0100010	0100000000
2	0010	0010	10100	0100100	0010000000
3	0011	0011	10010	0101000	0001000000
4	0100	0100	01010	0110000	0000100000
5	0101	1011	00110	1000001	0000010000
6	0110	1100	10001	1000010	0000001000
7	0111	1101	01001	1000100	0000000100
8	1000	1110	00101	1001000	0000000010
9	1001	1111	00011	1010000	0000000001
Unused bit patterns					
	1010	0101	00000	0000000	0000000000
	1011	0110	00001	0000001	0000000011
	1100	0111	00010	0000010	0000000101
	1101	1000	00100	0000011	0000000110
	1110	1001	00111	0000100	0000000111
	1111	1010	...	...	...