Tutorial Data representation: solutions

Solutions

1.

a) $37 = 2^5 + 2^2 + 2^0 = \texttt{0b00100101}$
b) $89 = \texttt{0b01011001}$
c) $4 = \texttt{0b00000100}$
d) $126 = \texttt{0b01111110}$
e) This can't be represented in 8 bits.

2.

a) $\texttt{0b1100} = 2^3 + 2^2 = 8 + 4 = 12$
b) $\texttt{0b100100} = 36$
c) $\texttt{0b11111111} = 255$

3.

a) 4 bits
b) 8 bits
c) 9 bits
d) 11 bits
e) 12 bits

Hint: You can find the answer using logarithm base 2 ($\log_2$)

4.

a) $67 = (4 \times 16) + 3 = \texttt{0x43}$
b) $142 = \texttt{0x8E}$
c) $1348 = \texttt{0x544}$

5.

a) $\texttt{0x1B} = (1 \times 16) + 11 = 27$
b) $\texttt{0xA7} = 167$
c) $\texttt{0x8CE} = 2254$

6.

`0xBE9C8`

Make groups of 4 bits (ensure group from the least significant bit) and convert directly to hexadecimal. Alternatively, convert to decimal and then hexadecimal.

7.

`0b1011 0010 0101 1101 0110`

Use the same principle as the previous question (Q6)

8.

a) `0b11111010`
b) `0b10111000`
c) `0b11111111`

These answers can be checked by converting to decimal notation.

9.

a) `0b00110110`
b) `0b10010110`
c) `0b00110010`

These answers can be checked by converting to decimal notation.

10. In the sign-magnitude representation, positive integers remain the same. For negative numbers the high bit becomes 1.

a) `0b00010111`
b) `0b10010111`
c) `0b10110000`
d) `0b11000001`

11. For positive numbers there is no change. For negative numbers, we invert the bits, add 1 and ignore any overflow bits.

a) `0b00010111`
b) `0b11101001`
c) `0b11010000`
d) `0b10111111`

These answers can be checked by adding the positive and negative values together, the result should be zero.

12.

a) `1 10000010 01010100000000000000000` = -10.625
b) `0 10001000 00100000001000000000000` = 800.25

13.

a) 1024.0 = `0 10001001 00000000000000000000000`
b) -4.75 = `1 10000001 00110000000000000000000`

The process here is to convert the decimal number into a binary fraction with a decimal point (for example 100.11 for part b). The decimal point is then moved to after the leading 1 bit, so you get 1.0011, for example, and then keep track of how many positions the decimal point moved. If it moved to the right, then that is a positive exponent (in this case 2 so the multiplier is 2^2), to the left means a negative exponent. From here, set the sign bit, add 127 to the exponent and represent that value in 8 bits, and use the portion after the decimal place as the mantissa (recall the 1 is implied).

14. For this you might need to do some independent research to find an ASCII table.

a) 51 or `0x33`
b) 102 or `0x66`
c) 38 or `0x26`
d) 59 or `0x3B`

15.

a) `U+0043` LATIN CAPITAL LETTER C

- Convert `0x0043` to binary = `0b0000 0000 0100 0011`
- Note that codepoint is less than `0x7F`, so only one byte is needed. Represent using 7 bits (underlined) preceded by a 0 = `0100 0011` = `0x43`.

b) `U+1F305` SUNRISE

- Convert `0x1F305` to binary = `0b0001 1111 0011 0000 0101`
- Note that codepoint is greater than `0x10000`, so four bytes are needed, using the format `11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`.
- There are 21 bits to include, so add an extra 0 at the start of the binary string = `0 0001 1111 0011 0000 0101` and divide into groups of three = `000 011 111 001 100 000 101`.
- Represent in `UTF-8` using 4 bytes using the format = `1111 0000 1001 1111 1000 1100 1000 0101` = `0x F0 9F 8C 85`.

c) `U+00F3` LATIN SMALL LETTER O WITH ACUTE

- `0x C3 B3`