“A little bit more, a little bit more.”

Convert binary to decimal

11001011
= $1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
= $1 \times 128 + 1 \times 64 + 0 \times 32 + 1 \times 16 + 1 \times 8 + 1 \times 2 + 1 \times 1$
= 203 (base 10)

10110110
= $1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
= $1 \times 128 + 0 \times 64 + 1 \times 32 + 1 \times 16 + 1 \times 8 + 1 \times 2 + 0 \times 1$
= 182 (base 10)
... in 2 bits
\[ 1 \ 1 = 3 \text{ (base 10)} \]

... in 3 bits
\[ 1 \ 1 \ 1 = \_ \text{ (base 10)} \]

... in 4 bits
\[ 1 \ 1 \ 1 \ 1 = \_ \text{ (base 10)} \]

... in 5 bits
\[ 1 \ 1 \ 1 \ 1 \ 1 = \_ \text{ (base 10)} \]

... in 8 bits (one byte)
\[ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 = \_ \text{ (base 10)} \]
\[ = 2^8 - 1 \]
Remember that binary numbers have positions that grow in powers of two.

\[
\begin{array}{cccccccc}
\text{positions} & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
\text{value} & 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}
\]

Using the powers of two ...
Repeatedly subtract the largest possible power of two.

\[
\begin{array}{cccccccc}
\text{decimal number} & 183 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}
\]

A second example…

Using the powers of two …
Repeatedly subtract the largest possible power of two.

decimal number 82

82 - ? = ?

binary number

Data FAQ

FAQ7 Why are real number data sometimes inaccurate?

... in my computer?
... in my calculator?
The usual approach for storing a positive real number as data also relies upon base-2 numbers.

**Base-10 Real Numbers**
- positions right of dec. point are negative powers of ten

  - example: \( 9.235 = 9 \times 10^0 + 2 \times 10^{-1} + 3 \times 10^{-2} + 5 \times 10^{-3} \)

**Base-2 Numbers**
- positions right of “.” are negative powers of two
  - example: \( 1.101 = 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \)
    \[ = 1 \times 1 + 1 \times \frac{1}{2} + 0 \times \frac{1}{4} + 1 \times \frac{1}{8} \]
    \[ = 1.625 \text{ (base 10)} \]

**Real binary numbers aren’t always exact.**

Even base-10 numbers can’t always be stored in a fixed number of digits.
  - example: \( \frac{1}{3} = \)

The problem gets worse with binary number storage.
  - example: \( \frac{1}{5} = 0.2 \text{ (base 10)} \)
    \[ = 0.0011001100110011\ldots \text{ (base 2)} \]

**This is one reason that computers/calculators can be imprecise!**
FAQ 8 How is text represented as data?

Text is made of symbols, called characters; each can be assigned a number.

<table>
<thead>
<tr>
<th>character</th>
<th>number</th>
<th>bit sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>00000</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>00001</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>00010</td>
</tr>
<tr>
<td>d</td>
<td>3</td>
<td>00011</td>
</tr>
<tr>
<td>e</td>
<td>4</td>
<td>00100</td>
</tr>
<tr>
<td>f</td>
<td>5</td>
<td>00101</td>
</tr>
<tr>
<td>g</td>
<td>6</td>
<td>00110</td>
</tr>
<tr>
<td>h</td>
<td>7</td>
<td>00111</td>
</tr>
<tr>
<td>i</td>
<td>8</td>
<td>01000</td>
</tr>
<tr>
<td>j</td>
<td>9</td>
<td>01001</td>
</tr>
<tr>
<td>k</td>
<td>10</td>
<td>01010</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>z</td>
<td>25</td>
<td>11001</td>
</tr>
</tbody>
</table>

More bits are required to store all capital letters, and punctuation marks (‘.’, ‘?’), and special symbols (‘*’, ‘&’, ‘#’). One byte (8 bits) is enough for all symbols on your keyboard. Two bytes (16 bits) allows for symbols from many other languages (‘ñ’, ‘â’).
FAQ 9 What about image and sound data?

Pixel -- short for picture element
-- one dot in a 2-dimensional grid of such dots; an image is constructed out of such a grid.

Consider the following black & white 8 by 8 picture:

```
01111100
01000010
01000010
01111100
01001000
01000100
01000010
00000000
```

Storing color typically uses the following per pixel:

- **greenness** ... 1 byte
- **redness** ... 1 byte
- **blueness** ... 1 byte
Consider another way to store images.

<table>
<thead>
<tr>
<th>Raw pixels</th>
<th>Run Length Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111100</td>
<td>1, 5, 2</td>
</tr>
<tr>
<td>01000010</td>
<td>1, 1, 4, 1, 1</td>
</tr>
<tr>
<td>01000100</td>
<td>1, 1, 4, 1, 1</td>
</tr>
<tr>
<td>01001000</td>
<td>1, 5, 2</td>
</tr>
<tr>
<td>01000100</td>
<td>1, 1, 2, 1, 3</td>
</tr>
<tr>
<td>01000010</td>
<td>1, 1, 3, 1, 2</td>
</tr>
<tr>
<td>00000000</td>
<td>1, 4, 1, 1</td>
</tr>
</tbody>
</table>

a total of 64 numbers  a total of 31 numbers

**Compression**
-a way to represent data in more compact form

**Lossless vs. Lossy**

RLE is **lossless compression**, i.e. no information is lost.
Lossless compression is used in **GIF**, **TIFF**, and **PNG**.

**Lossy Compression**:
approximate the original data to reduce its size.
Lossy used in **JPG**
A typical CD contains two channels of 44,100 samples per second and 16 bits (2 bytes) per sample.

**WAV**
- standard sound file form using lossless compression

**MP3**
- standard sound file form using lossy compression