▶ Claim: at least conceptually we could say that

$$123 \equiv \langle 3, 2, 1 \rangle,$$

i.e., the decimal literal 123 is basically just a sequence of digits.

▶ Question: given
  ▶ a **bit** is a single binary digit, i.e., 0 or 1,
  ▶ a **byte** is an 8-element sequence of bits, and
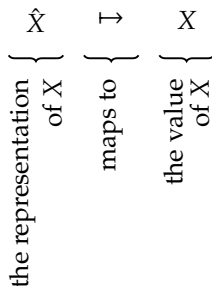 ▶ a **word** is a w-element sequence of bits

and so, e.g.,

$$01111011 \equiv \langle 1, 1, 0, 1, 1, 1, 1, 0 \rangle,$$

what do these things *mean* ... what do they *represent*?

▶ Answer: anything *we* decide they do!

► Concept:

$$\hat{X} \qquad \mapsto \qquad X$$

$\underbrace{\hat{X}}_{\text{the representation of X}}$ $\underbrace{\mapsto}_{\text{maps to}}$ $\underbrace{X}_{\text{the value of X}}$

i.e., we need

1. a concrete representation that we can write down, plus
2. a mapping that yields the correct value *and* is consistent (in both directions).

▶ Agenda:

1. useful properties of bit-sequences,
2. positional number systems $\rightsquigarrow$ standard integer representations.

Part 1: useful properties of bit-sequences

Definition

A given literal, say

$$X = 1111011,$$

can be interpreted in *two* ways:

1. A **little-endian** ordering is where we read bits in a literal from right-to-left, i.e.,

$$X_{LE} = \langle X_0, X_1, X_2, X_3, X_4, X_5, X_6 \rangle = \langle 1, 1, 0, 1, 1, 1, 1 \rangle,$$

   where

   - the Least-Significant Bit (LSB) is the right-most in the literal (i.e., X_0), and
   - the Most-Significant Bit (MSB) is the left-most in the literal (i.e., $X_{n-1} = X_6$).

2. A **big-endian** ordering is where we read bits in a literal from left-to-right, i.e.,

$$X_{BE} = \langle X_6, X_5, X_4, X_3, X_2, X_1, X_0 \rangle = \langle 1, 1, 1, 1, 0, 1, 1 \rangle,$$

   where

   - the Least-Significant Bit (LSB) is the left-most in the literal (i.e., $X_{n-1} = X_6$), and
   - the Most-Significant Bit (MSB) is the right-most in the literal (i.e., X_0).

Part 1: useful properties of bit-sequences

Definition

Following the idea of vectorial Boolean function, given an n-element bit-sequence X, and an m-element bit-sequence Y we can clearly

1. overload $\oslash \in \{\neg\}$, i.e., write

$$R = \oslash X,$$

   to mean

$$R_i = \oslash X_i$$

   for $0 \leq i < n$,

2. overload $\ominus \in \{\wedge, \vee, \oplus\}$, i.e., write

$$R = X \ominus Y,$$

   to mean

$$R_i = X_i \ominus Y_i$$

   for $0 \leq i < n = m$, where if $n \neq m$, we pad either X or Y with 0 until the $n = m$.

Part 1: useful properties of bit-sequences

<table>
<tr><td>

Definition

Following the idea of vectorial Boolean function, given an n-element bit-sequence X, and an m-element bit-sequence Y we can clearly

1. overload $\oslash \in \{\neg\}$, i.e., write

$$R = \oslash X,$$

   to mean

$$R_i = \oslash X_i$$

   for $0 \le i < n$,

2. overload $\ominus \in \{\wedge, \vee, \oplus\}$, i.e., write

$$R = X \ominus Y,$$

   to mean

$$R_i = X_i \ominus Y_i$$

   for $0 \le i < n = m$, where if $n \neq m$, we pad either X or Y with 0 until the $n = m$.

</td></tr>
</table>

▶ Example: in C, we use the computational (or **bit-wise**) operators ~, &, |, and ^ this way: they apply NOT, AND, OR, and XOR to corresponding bits in the operands.

Part 1: useful properties of bit-sequences

Definition

Given two n-bit sequences X and Y, we can define some important properties named after Richard Hamming, a researcher at Bell Labs:

▶ The **Hamming weight** of X is the number of bits in X that are equal to 1, i.e., the number of times $X_i = 1$. This can be expressed as

$$HW(X) = \sum_{i=0}^{n-1} X_i.$$

▶ The **Hamming distance** between X and Y is the number of bits in X that differ from the corresponding bit in Y, i.e., the number of times $X_i \neq Y_i$. This can be expressed as

$$HD(X, Y) = \sum_{i=0}^{n-1} X_i \oplus Y_i.$$

Note that both quantities naturally generalise to non-binary sequences.

Part 1: useful properties of bit-sequences

Definition

Given two n-bit sequences X and Y, we can define some important properties named after Richard Hamming, a researcher at Bell Labs:

▶ The **Hamming weight** of X is the number of bits in X that are equal to 1, i.e., the number of times $X_i = 1$. This can be expressed as

$$\mathrm{HW}(X) = \sum_{i=0}^{n-1} X_i.$$

▶ The **Hamming distance** between X and Y is the number of bits in X that differ from the corresponding bit in Y, i.e., the number of times $X_i \neq Y_i$. This can be expressed as

$$\mathrm{HD}(X, Y) = \sum_{i=0}^{n-1} X_i \oplus Y_i.$$

Note that both quantities naturally generalise to non-binary sequences.

▶ Example: given $X = \langle 1, 0, 0, 1 \rangle$ and $Y = \langle 0, 1, 1, 1 \rangle$ we find that

$$\mathrm{HW}(X) \quad = \sum_{i=0}^{n-1} X_i \qquad\qquad\qquad\qquad\qquad\qquad\qquad = 1 + 0 + 0 + 1 = 2$$

$$\mathrm{HD}(X, Y) = \sum_{i=0}^{n-1} X_i \oplus Y_i = (1 \oplus 0) + (0 \oplus 1) + (0 \oplus 1) + (1 \oplus 1) = 1 + 1 + 1 + 0 = 3$$

▶ Concept: a **positional number system** expresses the value of a number x using a base-b (or radix-b) expansion, i.e.,

$$
\begin{aligned}
\hat{x} \quad &= \quad \langle \hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1} \rangle \\[1em]
&\mapsto \quad x \\[1em]
&= \quad \pm \sum_{i=0}^{n-1} \hat{x}_i \cdot b^i
\end{aligned}
$$

where each $\hat{x}_i$
▶ is one of n digits taken from the digit set $X = \{0, 1, \ldots, b-1\}$,
▶ is "weighted" by some power of of the base b.

▶ Beware!
  ▶ for $b > 10$ we can't express $\hat{x}_i$ using a single Arabic numeral,
  ▶ for $b = 16$, for example, we use letters instead:

$$
\begin{array}{ccc}
A & \mapsto & 10 \\
B & \mapsto & 11 \\
C & \mapsto & 12 \\
D & \mapsto & 13 \\
E & \mapsto & 14 \\
F & \mapsto & 15
\end{array}
$$

Example

Consider an example where we

1. set $b = 10$, i.e., deal with **decimal** numbers, and
2. have $\hat{x}_i \in X = \{0, 1, \ldots, 10 - 1 = 9\}$.

This means we can write

$$
\begin{aligned}
\hat{x} = 123 \quad &= \quad \langle 3, 2, 1 \rangle_{(10)} \\
&\mapsto \quad x \\
&= \quad \sum_{i=0}^{n-1} \hat{x}_i \cdot 10^i \\
&= \quad 3 \cdot 10^0 + 2 \cdot 10^1 + 1 \cdot 10^2 \\
&= \quad 3 \cdot 1 \quad + 2 \cdot 10 \quad + 1 \cdot 100 \\
&= \quad 123_{(10)}
\end{aligned}
$$

i.e., represent the value "one hundred and twenty three" in a variety of ways using different bases.

Example

Consider an example where we

1. set $b = 2$, i.e., deal with **binary** numbers, and
2. have $\hat{x}_i \in X = \{0, 2 - 1 = 1\}$.

This means we can write

$$
\begin{aligned}
\hat{x} = 1111011 \quad &= \quad \langle 1, 1, 0, 1, 1, 1, 1 \rangle_{(2)} \\
&\mapsto \quad x \\
&= \quad \sum_{i=0}^{n-1} \hat{x}_i \cdot 2^i \\
&= \quad 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6 \\
&= \quad 1 \cdot 1 \; + 1 \cdot 2 \; + 0 \cdot 4 \; + 1 \cdot 8 \; + 1 \cdot 16 + 1 \cdot 32 + 1 \cdot 64 \\
&= \quad 123_{(10)}
\end{aligned}
$$

i.e., represent the value "one hundred and twenty three" in a variety of ways using different bases.

Example

Consider an example where we

1. set $b = 8$, i.e., deal with **octal** numbers, and
2. have $\hat{x}_i \in X = \{0, 1, \ldots, 8 - 1 = 7\}$.

This means we can write

$$
\begin{aligned}
\hat{x} = 173 \quad &= \quad \langle 3, 7, 1 \rangle_{(8)} \\
&\mapsto \quad x \\
&= \quad \sum_{i=0}^{n-1} \hat{x}_i \cdot 8^i \\
&= \quad 3 \cdot 8^0 + 7 \cdot 8^1 + 1 \cdot 8^2 \\
&= \quad 3 \cdot 1 + 7 \cdot 8 + 1 \cdot 64 \\
&= \quad 123_{(10)}
\end{aligned}
$$

i.e., represent the value "one hundred and twenty three" in a variety of ways using different bases.

Example

Consider an example where we

1. set $b = 16$, i.e., deal with **hexadecimal** numbers, and
2. have $\hat{x}_i \in X = \{0, 1, \ldots, 16 - 1 = 15\}$.

This means we can write

$$
\begin{aligned}
\hat{x} = 7B \quad &= \quad \langle B, 7 \rangle_{(16)} \\
&\mapsto \quad x \\
&= \quad \sum_{i=0}^{n-1} \hat{x}_i \cdot 16^i \\
&= \quad 11 \cdot 16^0 + 7 \cdot 16^1 \\
&= \quad 11 \cdot 1 + 7 \cdot 16 \\
&= \quad 123_{(10)}
\end{aligned}
$$

i.e., represent the value "one hundred and twenty three" in a variety of ways using different bases.

▶ Problem: we want to represent and perform various operations on elements of $\mathbb{Z}$, *but*
  1. it's an an infinite set, and
  2. so far we've ignored the issue of sign.

▶ Solution: in C, for example, we get

$$
\begin{array}{rcccl}
\texttt{unsigned char} & \simeq & \texttt{uint8\_t} & \mapsto & \{ \quad\quad\quad 0, \ldots, +2^8 - 1 \} \\
\texttt{char} & \simeq & \texttt{int8\_t} & \mapsto & \{ -2^7, \ldots, 0, \ldots, +2^7 - 1 \}
\end{array}
$$

but why *these*, and how do they work?

Definition

An unsigned integer can be represented in n bits by using the natural binary expansion. That is, we have

$$
\begin{aligned}
\hat{x} &= \langle \hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1} \rangle \\
&\mapsto x \\
&= \sum_{i=0}^{n-1} \hat{x}_i \cdot 2^i
\end{aligned}
$$

for $\hat{x}_i \in \{0, 1\}$, which yields

$$
0 \leq x \leq 2^n - 1.
$$

Example ($n = 8$)

$$11111111 \mapsto 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = +255_{(10)}$$

$$\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots$$

$$10000101 \mapsto 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = +133_{(10)}$$

$$\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots$$

$$10000000 \mapsto 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = +128_{(10)}$$
$$01111111 \mapsto 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = +127_{(10)}$$

$$\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots$$

$$01111011 \mapsto 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = +123_{(10)}$$

$$\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots$$

$$00000001 \mapsto 0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = +1_{(10)}$$
$$00000000 \mapsto 0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = +0_{(10)}$$

▶ Fact:
  ▶ each hexadecimal digit $x_i \in \{0, 1, \ldots, 15\}$,
  ▶ four bits gives $2^4 = 16$ possible combinations, so
  ▶ each hexadecimal digit can be thought of as a short-hand for four binary digits.

▶ Example: we can perform the following translation steps

$$
\begin{array}{rcllll}
8AC & = & \langle & C, & A, & 8, & \rangle_{(16)} \\
    & = & \langle & \langle\, 0,0,1,1 \,\rangle_{(2)}, & \langle\, 0,1,0,1 \,\rangle_{(2)}, & \langle\, 0,0,0,1 \,\rangle_{(2)} & \rangle_{(16)} \\
    & = & \langle & 0,0,1,1, & 0,1,0,1, & 0,0,0,1 & \rangle_{(16)} \\
    & \mapsto & 2220_{(10)} &&&&
\end{array}
$$

such that in C, for example,

$$
\texttt{0x8AC} \;=\; 2220_{(10)}.
$$

▶ Fact: left-shift (resp. right-shift) of some x by y digits is equivalent to multiplication (resp. division) by b^y.

▶ Example: taking $b = 2$ we find that

$$
\begin{array}{rcl}
x \times 2^y & = & (\sum_{i=0}^{n-1} x_i \cdot 2^i) \times 2^y \\
& = & \sum_{i=0}^{n-1} x_i \cdot 2^i \times 2^y \\
& = & \sum_{i=0}^{n-1} x_i \cdot 2^{i+y} \\
& = & x \ll y
\end{array}
$$

and

$$
\begin{array}{rcl}
x / 2^y & = & (\sum_{i=0}^{n-1} x_i \cdot 2^i) / 2^y \\
& = & \sum_{i=0}^{n-1} x_i \cdot 2^i / 2^y \\
& = & \sum_{i=0}^{n-1} x_i \cdot 2^{i-y} \\
& = & x \gg y
\end{array}
$$

such that in C, for example,

$$
\begin{array}{rclclcl}
\texttt{0x8AC} \ll 2 & \mapsto & 2220_{(10)} \times 2^2 & = & 8880_{(10)} & \mapsto & \texttt{0x22B0} \\
\texttt{0x8AC} \gg 2 & \mapsto & 2220_{(10)} / 2^2 & = & 555_{(10)} & \mapsto & \texttt{0x22B}
\end{array}
$$

Definition

A signed integer can be represented in n bits by using the **sign-magnitude** approach; 1 bit is reserved for the sign (0 means positive, 1 means negative) and $n-1$ for the magnitude. That is, we have

$$
\begin{aligned}
\hat{x} &= \langle \hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1} \rangle \\[1em]
&\mapsto x \\[1em]
&= (-1)^{\hat{x}_{n-1}} \cdot \sum_{i=0}^{n-2} \hat{x}_i \cdot 2^i
\end{aligned}
$$

for $\hat{x}_i \in \{0, 1\}$, which yields

$$
-2^{n-1} + 1 \le x \le +2^{n-1} - 1.
$$

Note that there are two representations of zero (i.e., $+0$ and -0).

Example ($n = 8$)

01111111	$\mapsto$	$(-1)^0$	$\cdot$	$($	$1 \cdot 2^6$	$+ 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$ $)$	$=$ $+127_{(10)}$
	$\vdots$						$\vdots$
01111011	$\mapsto$	$(-1)^0$	$\cdot$	$($	$1 \cdot 2^6$	$+ 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$ $)$	$=$ $+123_{(10)}$
	$\vdots$						$\vdots$
00000001	$\mapsto$	$(-1)^0$	$\cdot$	$($	$0 \cdot 2^6$	$+ 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$ $)$	$=$ $+1_{(10)}$
00000000	$\mapsto$	$(-1)^0$	$\cdot$	$($	$0 \cdot 2^6$	$+ 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$ $)$	$=$ $+0_{(10)}$
10000000	$\mapsto$	$(-1)^1$	$\cdot$	$($	$0 \cdot 2^6$	$+ 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$ $)$	$=$ $-0_{(10)}$
10000001	$\mapsto$	$(-1)^1$	$\cdot$	$($	$0 \cdot 2^6$	$+ 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$ $)$	$=$ $-1_{(10)}$
	$\vdots$						$\vdots$
11111011	$\mapsto$	$(-1)^1$	$\cdot$	$($	$1 \cdot 2^6$	$+ 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$ $)$	$=$ $-123_{(10)}$
	$\vdots$						$\vdots$
11111111	$\mapsto$	$(-1)^1$	$\cdot$	$($	$1 \cdot 2^6$	$+ 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$ $)$	$=$ $-127_{(10)}$

Example ($n = 8$)

reversed copy, non-contiguous number line
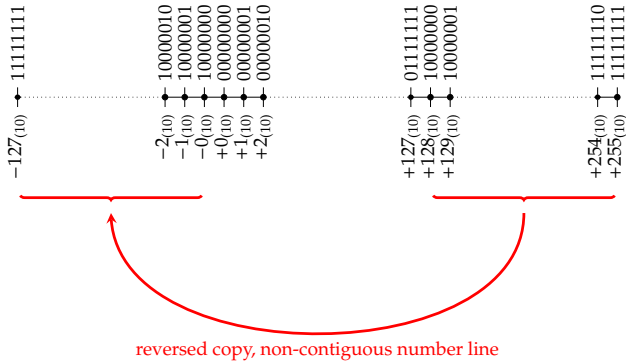
Definition

A signed integer can be represented in n bits by using the **two's-complement** approach; the basic idea is to weight the $(n-1)$-th bit using -2^{n-1} rather than $+2^{n-1}$, and all other bits as normal. That is, we have

$$
\begin{aligned}
\hat{x} \quad &= \quad \langle \hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1} \rangle \\[2mm]
&\mapsto \quad x \\[2mm]
&= \quad \hat{x}_{n-1} \cdot -2^{n-1} + \sum_{i=0}^{n-2} \hat{x}_i \cdot 2^i
\end{aligned}
$$

for $\hat{x}_i \in \{0, 1\}$, which yields
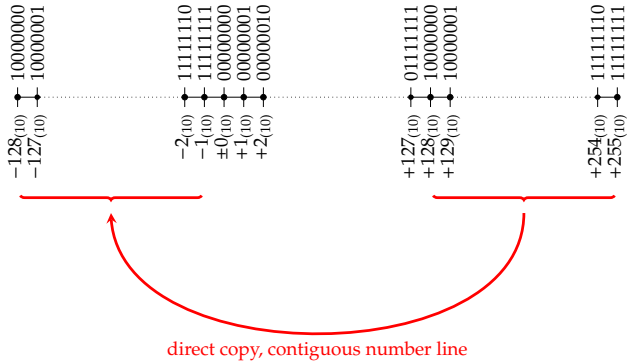
$$
-2^{n-1} \leq x \leq +2^{n-1} - 1.
$$

Example ($n = 8$)

$$01111111 \mapsto 0 \cdot -2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = +127_{(10)}$$

$$\vdots$$

$$01111011 \mapsto 0 \cdot -2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = +123_{(10)}$$

$$\vdots$$

$$00000001 \mapsto 0 \cdot -2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = +1_{(10)}$$

$$00000000 \mapsto 0 \cdot -2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = +0_{(10)}$$

$$11111111 \mapsto 1 \cdot -2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = -1_{(10)}$$

$$\vdots$$

$$10000101 \mapsto 1 \cdot -2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = -123_{(10)}$$

$$\vdots$$

$$10000000 \mapsto 1 \cdot -2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = -128_{(10)}$$

Example ($n = 8$)

direct copy, contiguous number line

▶ Take away points:

1. *We* control what bit-sequences mean: we can interpret an instance of the C `char` data-type as

   ▶ a signed 8-bit integer, *or*
   ▶ a generic object which can take one of 2^8 states,

   and, as a result, can represent *anything*, e.g.,

   ▶ a pixel within an image,
   ▶ a character within a document,
   ▶ a number within a matrix,
   ▶ ...

2. Beyond this, knowing about various standard representations is important and useful in a general sense.

Additional Reading

► *Wikipedia: Numeral system*. URL: `https://en.wikipedia.org/wiki/Numeral_system`.

► D. Page. "Chapter 1: Mathematical preliminaries". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer, 2009.

► B. Parhami. "Part 1: Number representation". In: *Computer Arithmetic: Algorithms and Hardware Designs*. 1st ed. Oxford University Press, 2000.

► W. Stallings. "Chapter 9: Number systems". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013.

► A.S. Tanenbaum and T. Austin. "Appendix A: Binary numbers". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012.

References

[1] *Wikipedia: Numeral system*. URL: `https://en.wikipedia.org/wiki/Numeral_system` (see p. 29).

[2] D. Page. "Chapter 1: Mathematical preliminaries". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer, 2009 (see p. 29).

[3] B. Parhami. "Part 1: Number representation". In: *Computer Arithmetic: Algorithms and Hardware Designs*. 1st ed. Oxford University Press, 2000 (see p. 29).

[4] W. Stallings. "Chapter 9: Number systems". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013 (see p. 29).

[5] A.S. Tanenbaum and T. Austin. "Appendix A: Binary numbers". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012 (see p. 29).