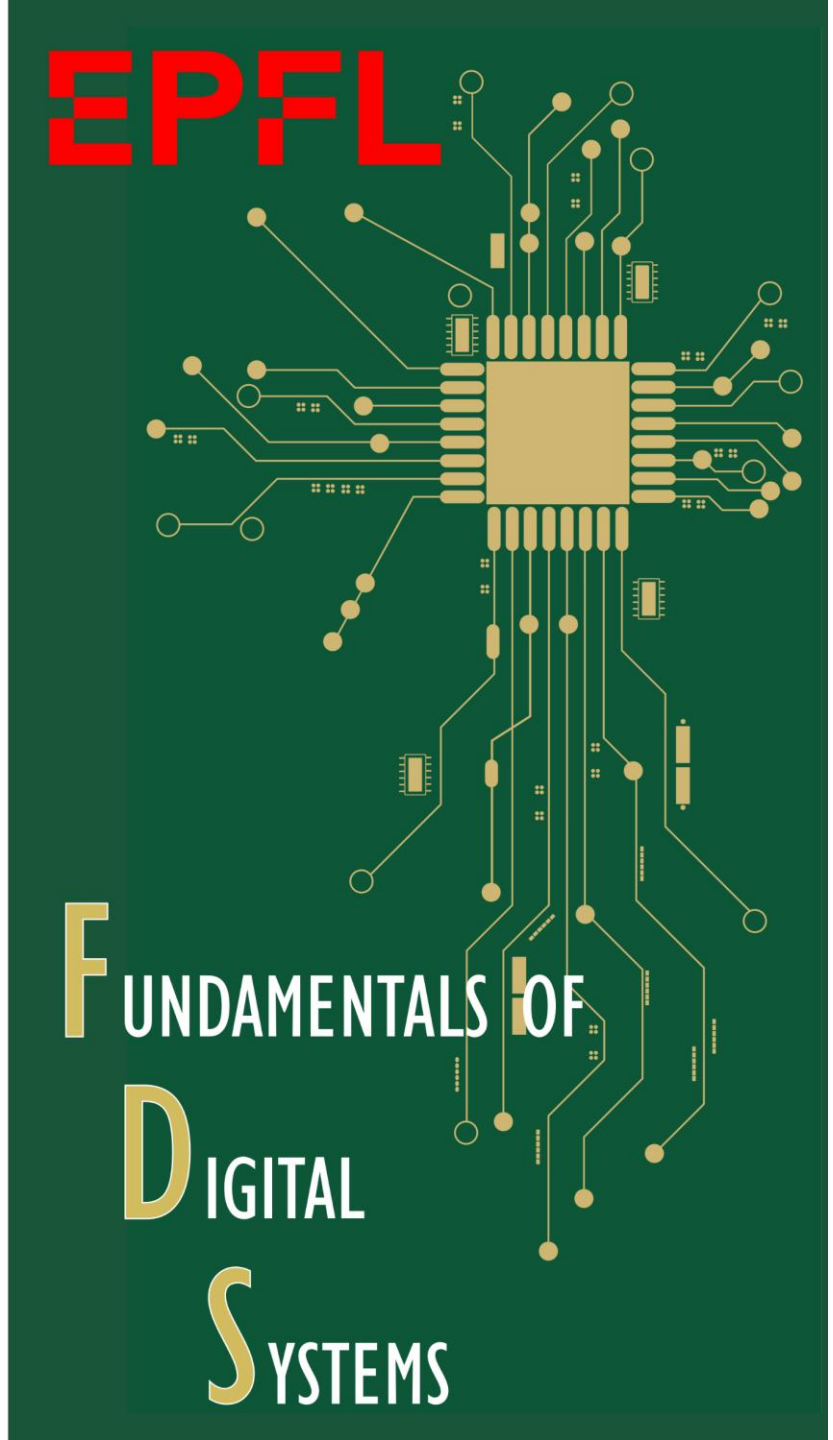


Number Systems

Arithmetic Operations with Integers

CS-173 Fundamentals of Digital Systems

Mirjana Stojilović

February 2025

Previously on FDS

…Number systems and codes

Previously…

- We saw how to represent integers in binary, octal, and hexadecimal number systems and perform conversions from one system to another

Decimal	Unsigned Binary 4-digit vector	Octal 2-digit vector	Hexadecimal 1-digit vector
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	**A**
11	1011	13	**B**
12	1100	14	**C**
13	1101	15	**D**
14	1110	16	**E**
15	1111	17	**F**

Previously…

Binary number systems

- Binary number representations of n-bit digit vector X
  - Unsigned integers

$$x = \sum_{i=0}^{n-1} X_i 2^i$$

  - Sign-and-magnitude

$$x_s = X_{n-1}, \quad x_m = \sum_{i=0}^{n-2} X_i 2^i$$

  - Two's complement

$$x = -X_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} X_i 2^i$$

Previously…
Binary number systems

EXAMPLES

- Binary number representations on n bits
  - Unsigned integers

$$x = \sum_{i=0}^{n-1} X_i 2^i$$

Examples:

$$01100_2 = 1 \cdot 2^3 + 1 \cdot 2^2 = 8 + 4 = 12_{10} = C_{16}$$
$$10011_2 = 1 \cdot 2^4 + 1 \cdot 2^1 + 1 \cdot 2^0 = 19_{10} = 13_{16}$$

  - Sign-and-magnitude

$$x_s = X_{n-1}, \quad x_m = \sum_{i=0}^{n-2} X_i 2^i$$

Examples:

$$01100_2 = +(1 \cdot 2^3 + 1 \cdot 2^2) = 12_{10}$$
$$10011_2 = -(1 \cdot 2^1 + 1 \cdot 2^0) = -3_{10}$$

  - Two's complement

$$x = -X_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} X_i 2^i$$

Examples:

$$01100_2 = 1 \cdot 2^3 + 1 \cdot 2^2 = 12_{10}$$
$$10011_2 = -1 \cdot 2^4 + 1 \cdot 2^1 + 1 \cdot 2^0$$
$$= -16 + 2 + 1 = -13_{10}$$

Previously…

Range (Sign) Extension in Two's Complement

- In two's complement, range (sign) extension boils down to simply filling the most significant bits with the sign bit

$$X = 10101_2 = -1 \cdot 2^4 + 1 \cdot 2^2 + 1 \cdot 2^0 = -16 + 4 + 1 = -11_{10}$$

Equal

$$\begin{aligned} X = 11110101_2 &= -1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^2 + 1 \cdot 2^0 \\ &= -128 + 64 + 32 + 16 + 4 + 1 \\ &= -11_{10} \end{aligned}$$

Previously…

Change of Sign in Two's Complement

- In two's complement, negating (i.e., changing the sign/polarity) can simply be done by, first complementing every bit and then adding +1
- Example:

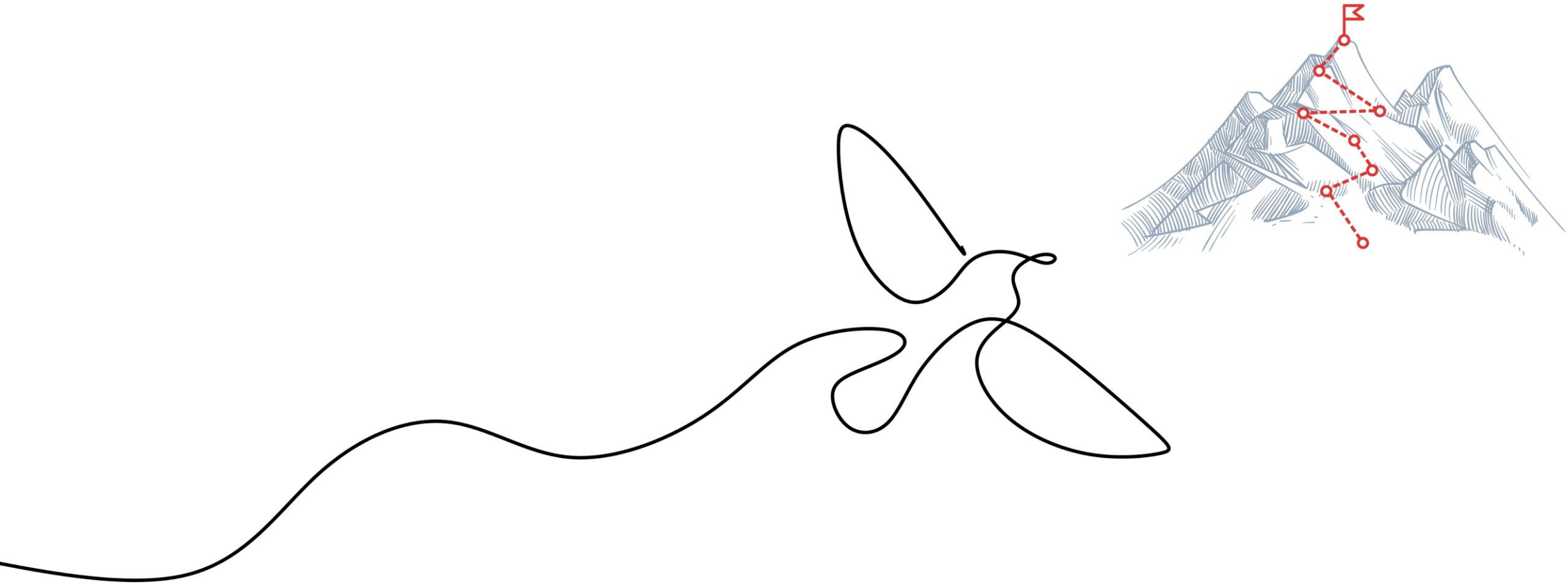
Bitwise complement

Add +1

$$-9_{10} = -(01001_2) = (10110_2) + 1 = 10111_2 = -16 + 4 + 2 + 1 = -9_{10}$$

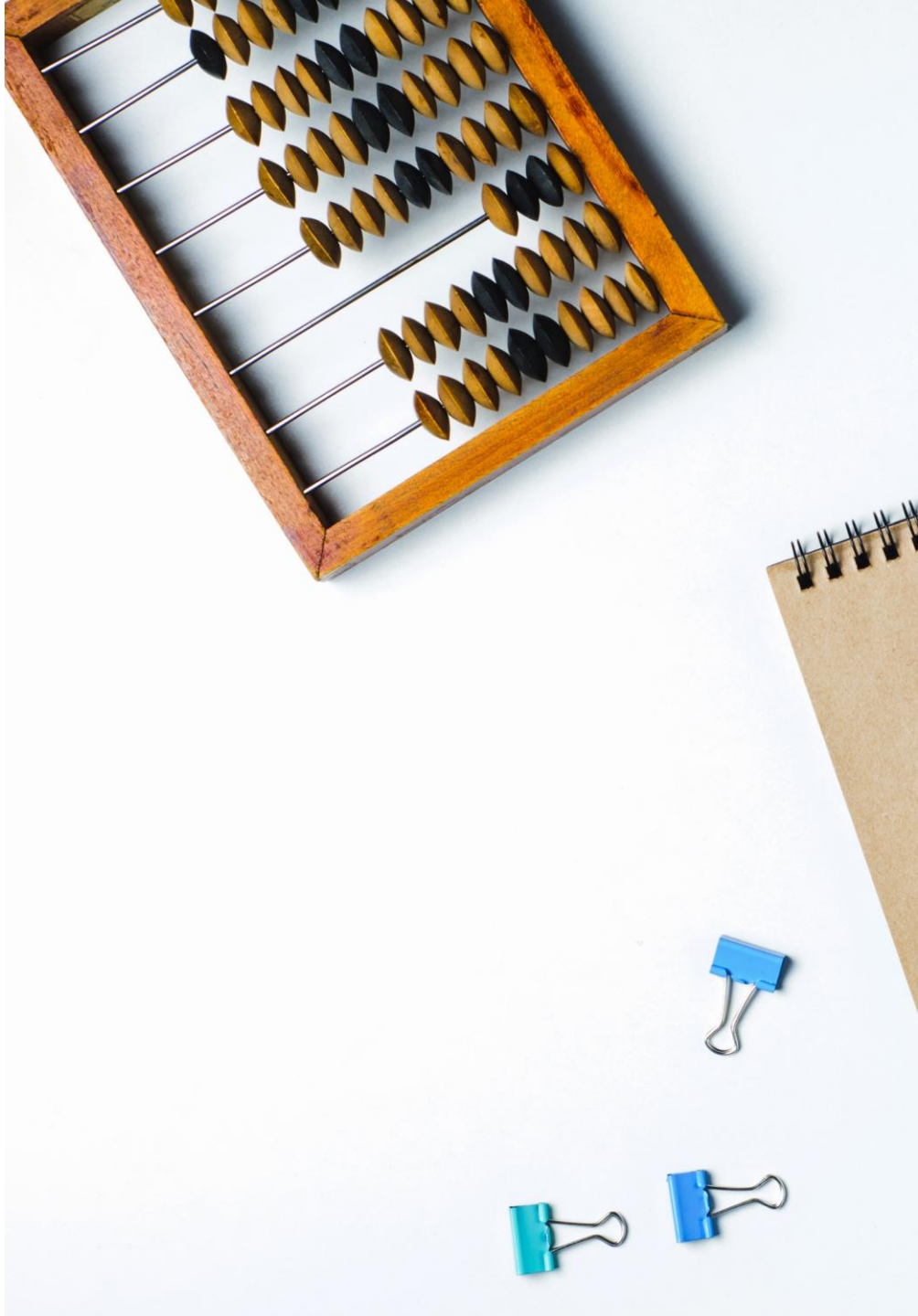
$$-(-9_{10}) = -(10111_2) = (01000_2) + 1 = 01001_2 = 8 + 1 = 9_{10}$$

$$-16 + 4 + 2 + 1$$

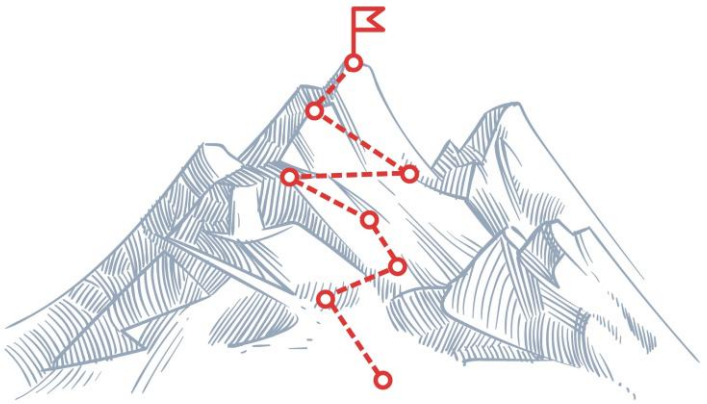


Let’s Talk About…

…Performing arithmetic operations on integers, signed and unsigned



Learning Outcomes

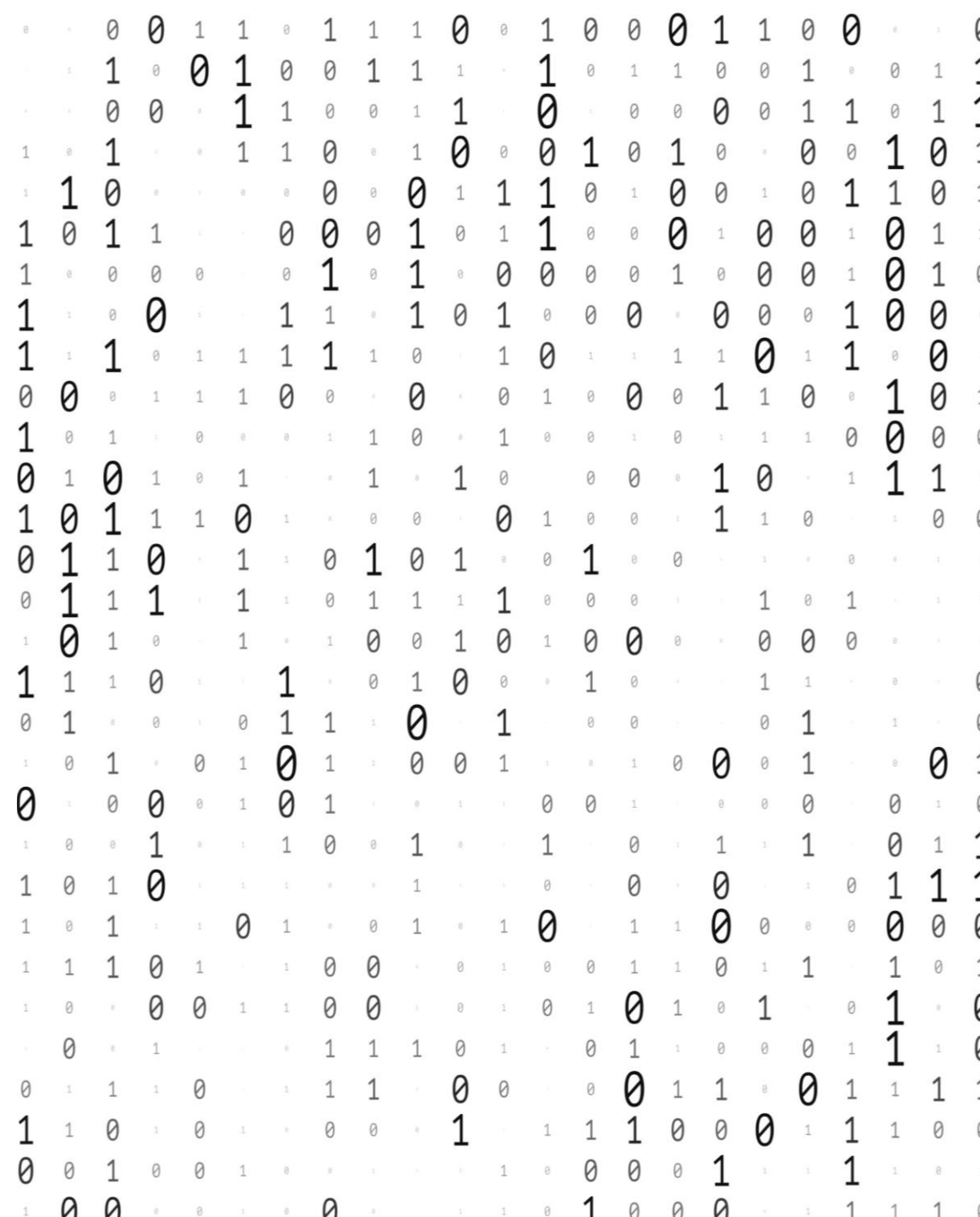


- Note the similarity between arithmetic operations with decimal and binary numbers
- Learn algorithms for +/-/×
  - Existence of carry and borrow
- Perform +/-/× pen-and-paper style
  - Note the similarity between arithmetic algorithms on binary numbers in unsigned and two's complement representation
- Detect exceptional cases when the result is too large to be represented with the given number of digits (overflow)

Quick Outline

- Addition and subtraction of unsigned integers
  - Addition; Overflow;
  - Subtraction;
- Two’s complement addition and subtraction
  - Addition/subtraction wheel
  - Addition
  - Subtraction
- Binary multiplication
  - Unsigned
  - Two’s complement
- Literature

Addition and Subtraction of Unsigned Integers



Decimal Numbers

Recall the Addition by Hand

- Adding two decimal numbers by hand involves a simple algorithm
  - Place the two numbers on top of one another
  - **Right-align** the least-significant digits
  - Add **leading zeros**, if needed
  - Starting from the least significant (rightmost) column, add digits one column at a time
    - If the sum requires two digits, propagate the **extra digit,** called **carry**, to the next column to the left
  - Continue until there is nothing left to add

$$\begin{array}{rr} & X \\ + & Y \\ \hline & X+Y \end{array} \qquad \begin{array}{r} 1\ 9\ 0_{10} \\ +\ 1\ 4\ 1_{10} \\ \hline 3\ 3\ 1_{10} \end{array}$$

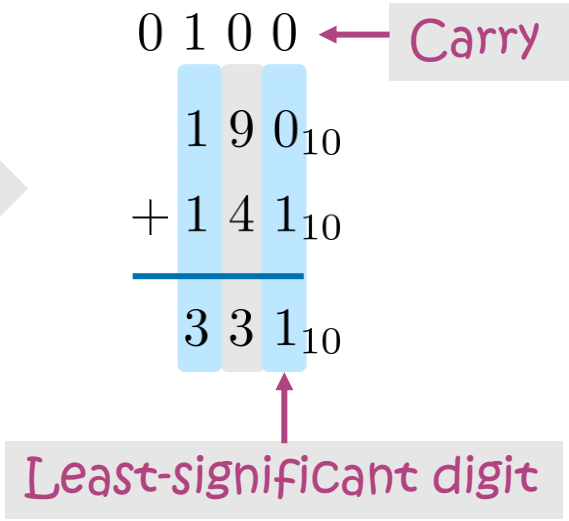
- Carry taking part in the sum is typically called **carry-in** (input carry); carry produced as part of the partial result is called **carry-out** (output carry)

Decimal Numbers

Example: Addition by Hand

$$s_i + 10 \cdot c_{\text{out}} = X_i + Y_i + c_{\text{in}}$$

$$\begin{array}{rr} & X \\ + & Y \\ \hline & X + Y \end{array}$$



Sum (one digit)

Carry-in

Carry-out

Index i

Direction

	X_i	Y_i	c_{in}	s_i	c_{out}	
$i = 0$	0	1	0	1	0	
$i = 1$	9	4	0	3	1	
$i = 2$	1	1	1	3	0	
$i = 3$	0	0	0	0	0	END

$X_i + Y_i + c_{\text{in}} > 9$ Carry occurred

Decimal Numbers

Example: Addition by Hand

$$s_i + 10 \cdot c_{\text{out}} = X_i + Y_i + c_{\text{in}}$$

$$\begin{array}{r} X \\ +\ Y \\ \hline X + Y \end{array}$$

$$\begin{array}{r} +\ 0\ 1\ 0\ 0 \\ 1\ 7\ 3_{10} \\ +\ 0\ 4\ 4_{10} \\ \hline 2\ 1\ 7_{10} \end{array}$$

Carry

Least-significant digit

Sum (one digit)

Carry-in

Carry-out

Index i

Direction

	X_i	Y_i	c_{in}	s_i	c_{out}	
$i = 0$	3	4	0	7	0	
$i = 1$	7	4	0	1	1	
$i = 2$	1	0	1	2	0	
$i = 3$	0	0	0	0	0	END

$X_i + Y_i + c_{\text{in}} > 9$ Carry occurred

Example: Decimal Numbers

Example: Subtraction by Hand

$$d_i - 10 \cdot b_{\text{out}} = X_i - Y_i - b_{in}$$

- Subtraction is performed similarly to addition
  - Instead of carries, using **borrows**

$$\begin{array}{rr} & X \\ - & Y \\ \hline & X - Y \end{array}$$

$$\begin{array}{rr} & 2\ 2\ 9_{10} \\ - & 0\ 4\ 6_{10} \\ \hline & 1\ 8\ 3_{10} \end{array}$$

$-\ 0\ 1\ 0\ 0$ ← Borrow

Least-significant digit

Difference (one digit)

Borrow-in

Borrow-out

Index i	X_i	Y_i	b_{in}	d_i	b_{out}	
$i = 0$	9	6	0	3	0	
$i = 1$	2	4	0	8	1	
$i = 2$	2	0	1	1	0	
$i = 3$	0	0	0	0	0	END

Direction

$X_i - Y_i - b_{\text{in}} < 0$ Borrow occurred

Decimal Numbers

Example: Subtraction by Hand

$$d_i - 10 \cdot b_{\text{out}} = X_i - Y_i - b_{in}$$

- Subtraction is performed similarly to addition
  - Instead of carries, using **borrows**

$$\begin{array}{rr} & X \\ - & Y \\ \hline & X - Y \end{array}$$

$$\begin{array}{rr} & 2\ 1\ 0_{10} \\ - & 1\ 0\ 9_{10} \\ \hline & 1\ 0\ 1_{10} \end{array}$$

$-\ 0\ 0\ 1\ 0$ ← Borrow

Least-significant digit

Difference (one digit)

Borrow-in

Borrow-out

Index i	X_i	Y_i	b_{in}	d_i	b_{out}	
$i = 0$	0	9	0	1	1	
$i = 1$	1	0	1	0	0	
$i = 2$	2	1	0	1	0	
$i = 3$	0	0	0	0	0	END

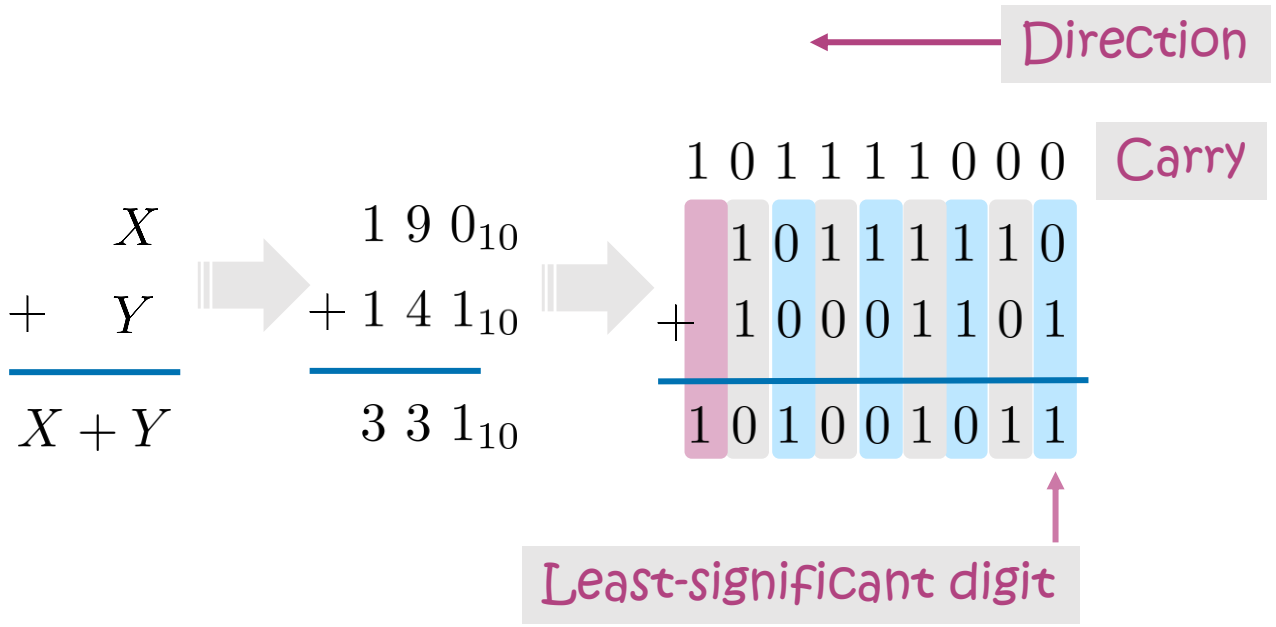
Direction

$X_i - Y_i - b_{\text{in}} < 0$ Borrow occurred

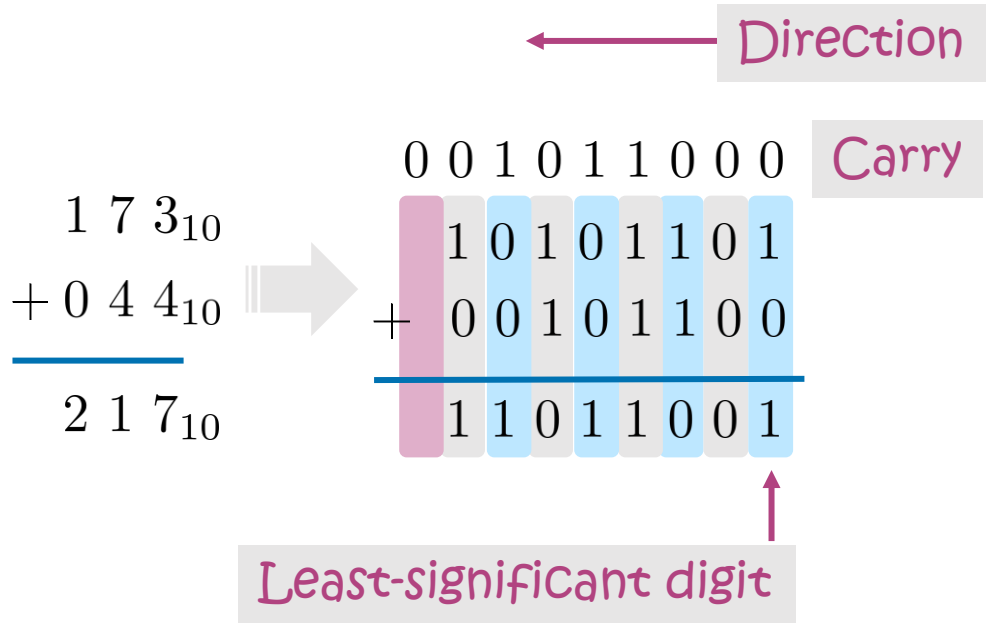
Addition of Unsigned Integers

By Hand

- The same idea as for decimal numbers
- Example 1:



- Example 2:



How Many Bits Are Needed…

- …to represent the sum of two n-bit unsigned numbers?

A: $n + 1$

$$s_{\min} = 0 + 0 = 0$$

$$s_{\max} = (2^n - 1) + (2^n - 1) = 2 \cdot 2^n - 2 = 2^{n+1} - 2$$

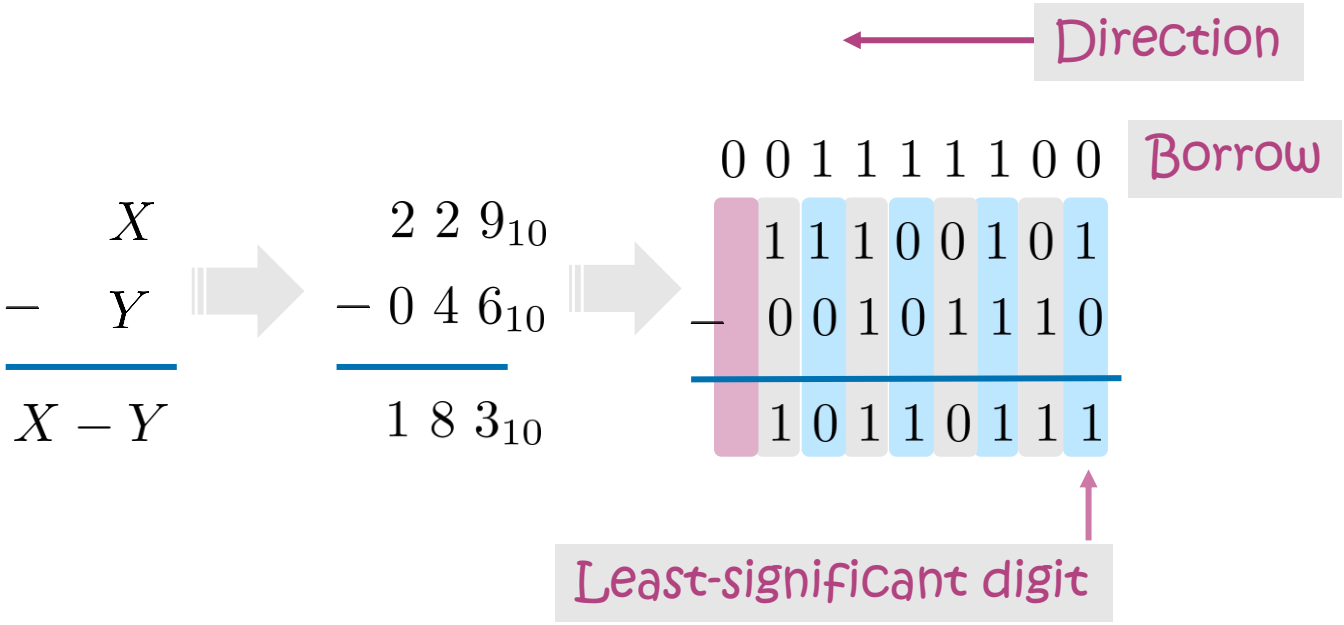
n+1 bits for the sum

- But we do not always have the extra bit in hardware!
- When the magnitude of the result exceeds the largest representable value, we say an **overflow** occurs and the result is incorrect

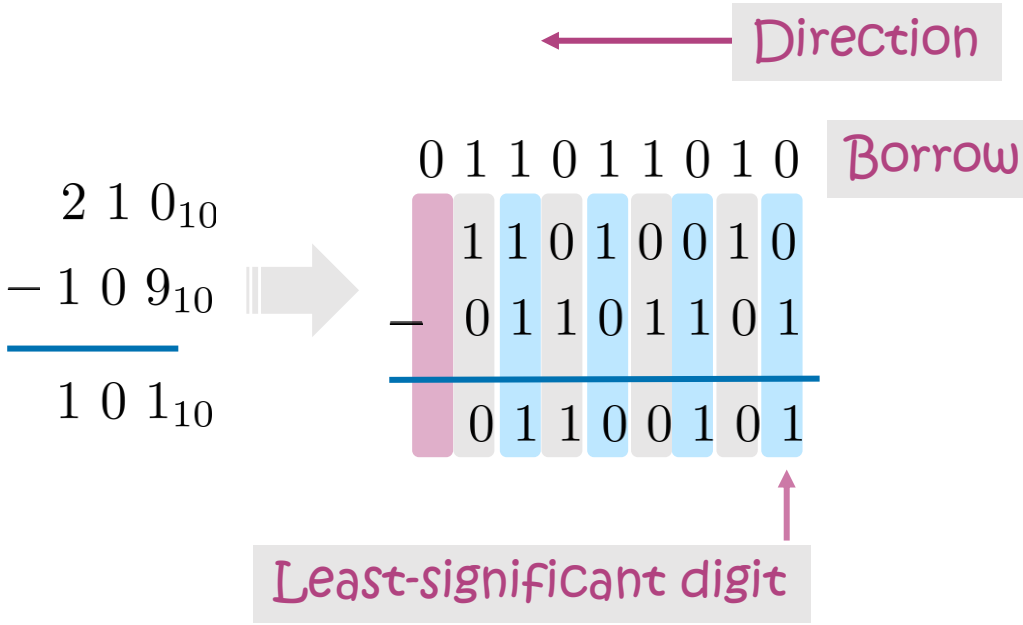
Subtraction of Unsigned Integers

By Hand

- The same idea as for decimal numbers
- Example 1:



- Example 2:



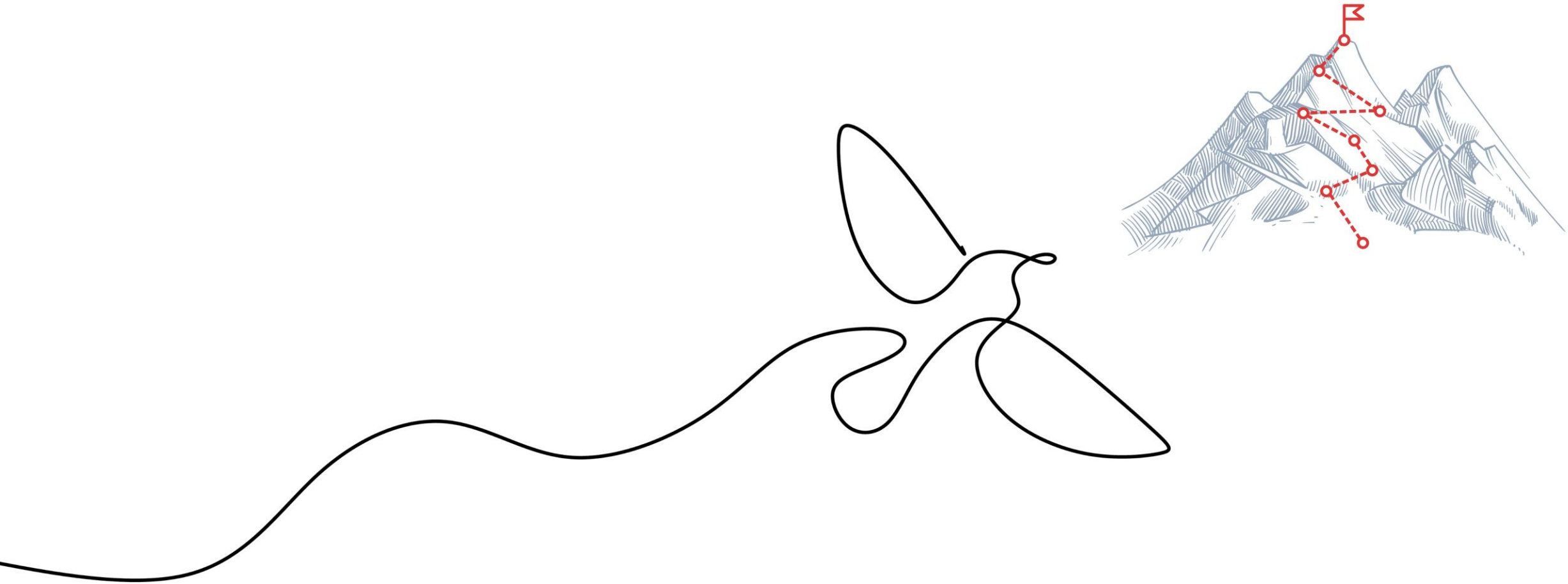
What If the Result Should be Negative?!

$$
\begin{array}{rr}
 & 1\ 0_{10} \\
- & 1\ 2\ 7_{10} \\
\hline
 & -1\ 1\ 7_{10}
\end{array}
$$

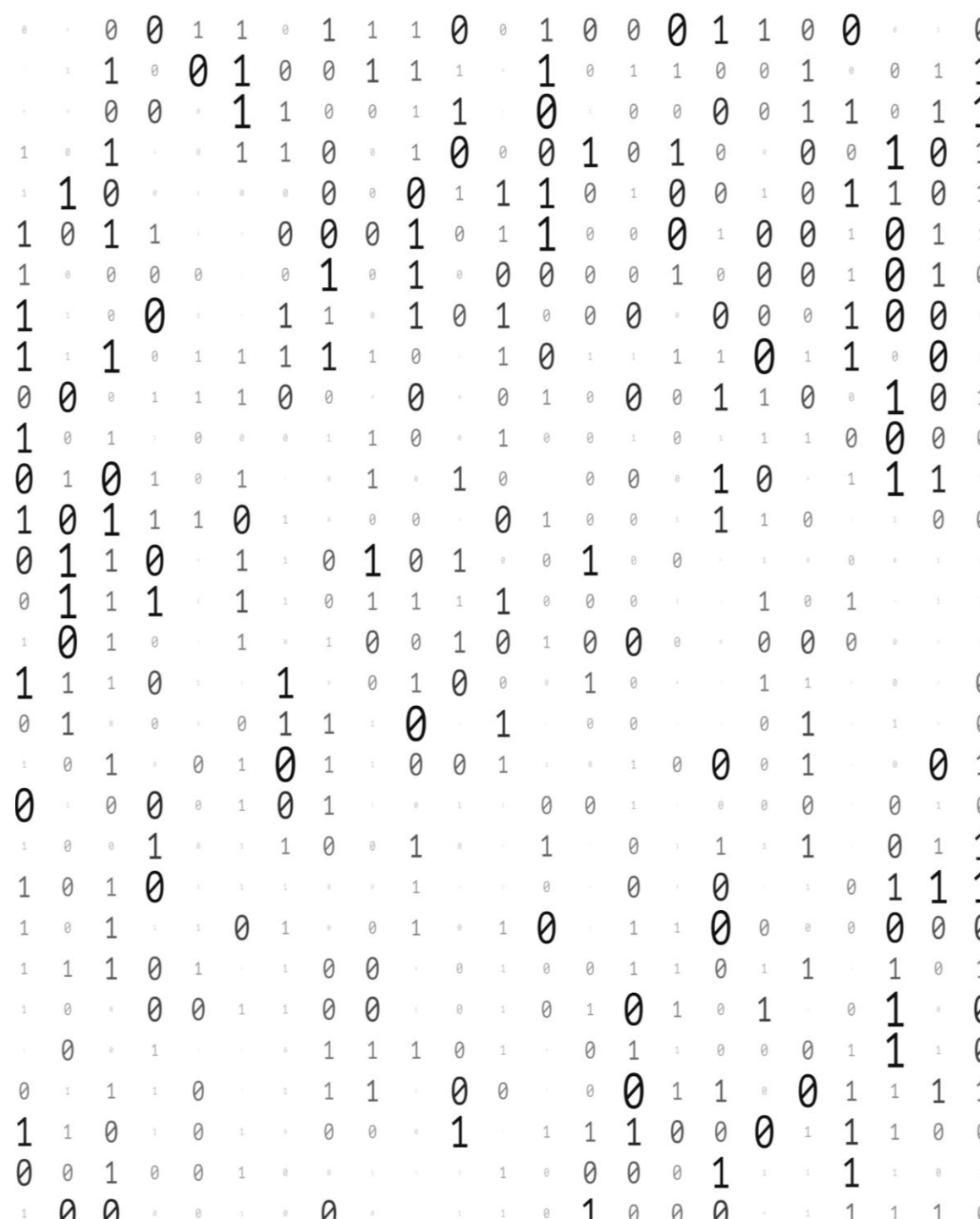
$$
\begin{array}{rr}
 & \ldots 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0 \\
 & 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0 \\
- & 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
\hline
 & \ldots 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1
\end{array} = 139_{10}
$$

Borrow

- Negative results cannot be represented using an unsigned system
- Trying to represent a value smaller than the minimum representable by the given number of bits n, gives an incorrect result



Two’s Complement Addition and Subtraction



Two's Complement Arithmetic

Graphical Representation

- Clockwise—addition of positive numbers
- Counterclockwise—subtraction of positive numbers

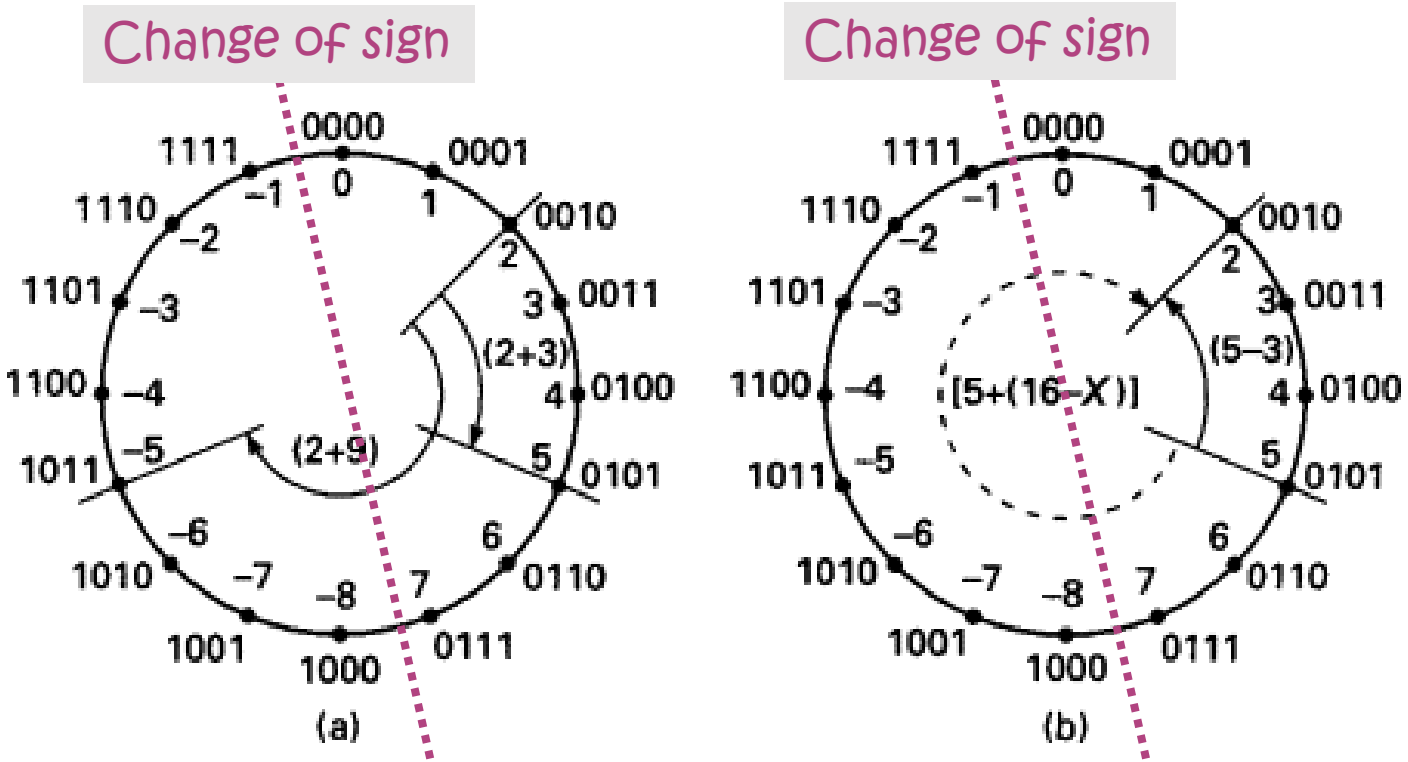
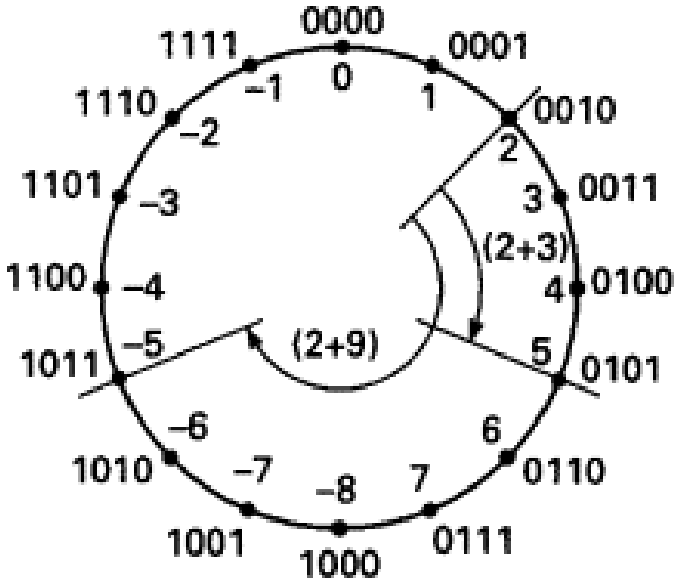


Image source: sciencedirect.com

Two's Complement is …

…the Preferred Representation in Digital Systems

- If we start with the smallest (most negative) number $1000_2 = -8_{10}$ and count up all successive numbers up to $0111_2 = 7_{10}$ can be obtained by adding 1 to the previous one
  - The result will always be correct as long as the range is not exceeded
  - Simple operation
  - Not as simple for sign-and-magnitude
  - Good for hardware implementation
    - **Win-win: the same hardware can perform the addition of unsigned numbers**



Two's Complement Addition

Overflow Detection Rules

- Same algorithm as for the unsigned numbers
- If the result exceeds the range, **overflow** occurs

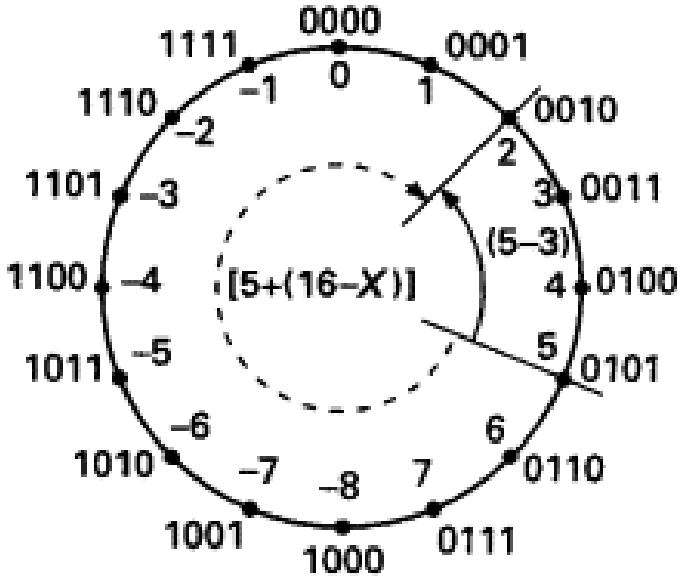
- **Overflow detection rules**
  - If the signs of the two numbers are the same but different from the sign of the sum, the overflow occurred
  - Alternative formulation: if c_{in} into and c_{out} out of the sign position are different, the overflow occurred
  - Adding two numbers of different signs never produces an overflow

Two's Complement Subtraction

Graphical Representation

- To subtract x, we **add** the two's complement of x
  - In the case of n=4, the two's complement is

$$2^4 - x = 16 - x$$



- Example:

$$\begin{aligned}(5-3)_{10} &= 5_{10} + (-3)_{10} = 0101_2 + (-(0011)_2) \\ &= 0101_2 + (1100_2 + 1_2) = 0101_2 + 1101_2 \\ &= 0010_2 = 2_{10}\end{aligned}$$

Image source: sciencedirect.com

Two's Complement Addition and Subtraction

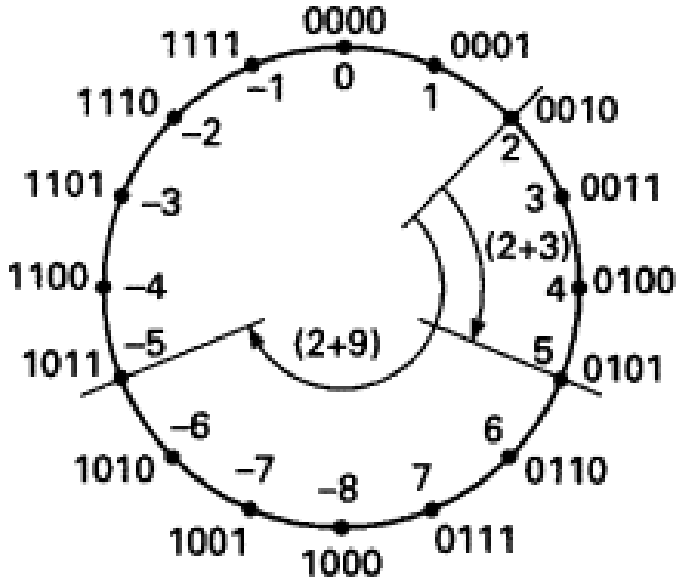
Examples

$-3_{10} = -(0011_2) = 1100_2 + 1_2 = 1101_2$

$-6_{10} = -(0110_2) = 1001_2 + 1_2 = 1010_2$

Overflow

$$\begin{array}{rr} & -3_{10} \\ + & -6_{10} \\ \hline & -9_{10} \end{array} \quad \Rightarrow \quad \begin{array}{r} 1101_2 \\ + 1010_2 \\ \hline 10111_2 \end{array}$$



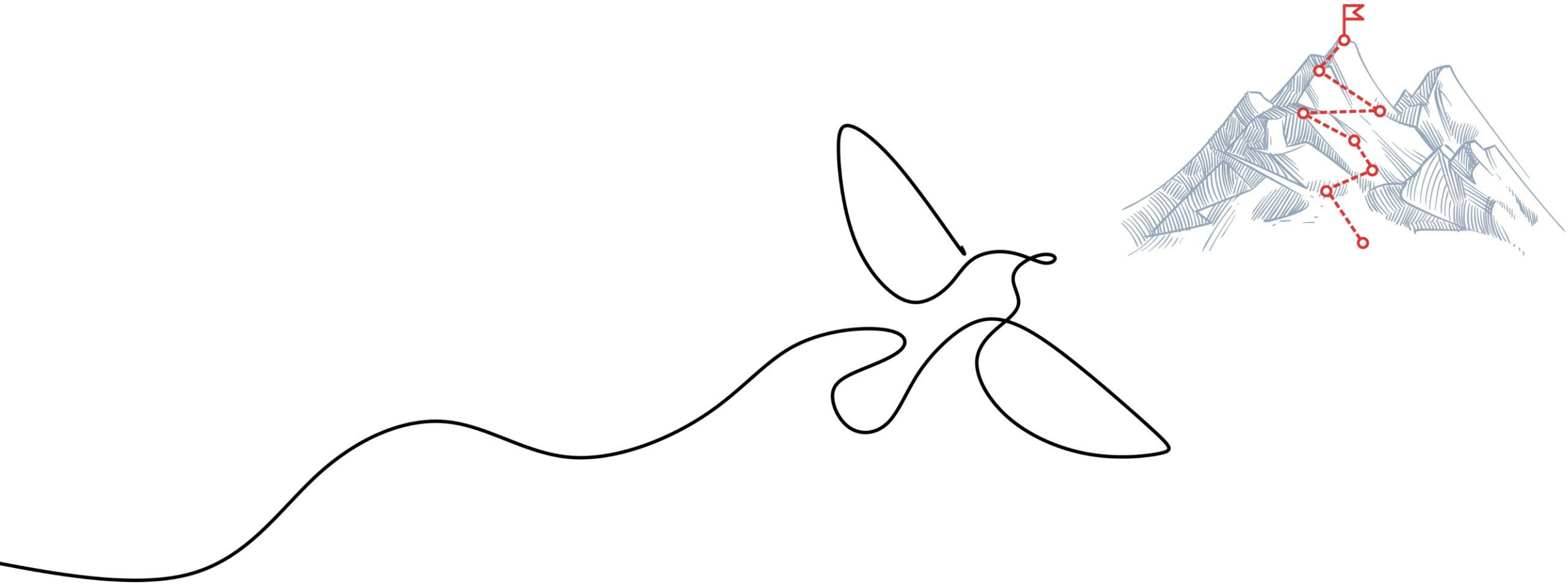
$$\begin{array}{rr} & 5_{10} \\ + & -6_{10} \\ \hline & -1_{10} \end{array} \quad \Rightarrow \quad \begin{array}{r} 0101_2 \\ + 1010_2 \\ \hline 1111_2 \end{array}$$

Overflow

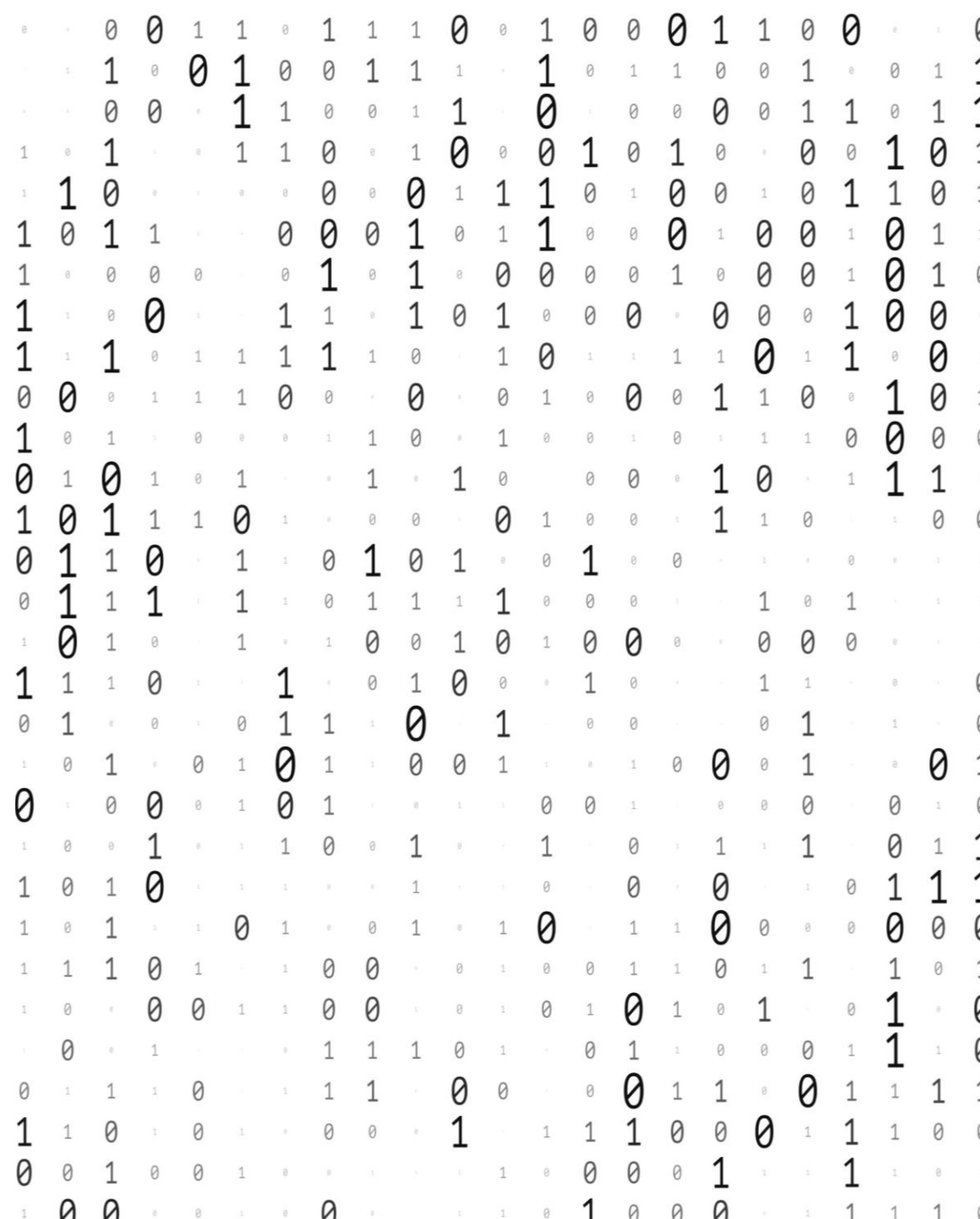
$$\begin{array}{rr} & -8_{10} \\ + & -8_{10} \\ \hline & -16_{10} \end{array} \quad \Rightarrow \quad \begin{array}{r} 1000_2 \\ + 1000_2 \\ \hline 10000_2 \end{array}$$

Overflow

$$\begin{array}{rr} & 5_{10} \\ + & 6_{10} \\ \hline & 11_{10} \end{array} \quad \Rightarrow \quad \begin{array}{r} 0101_2 \\ + 0110_2 \\ \hline 1011_2 \end{array}$$

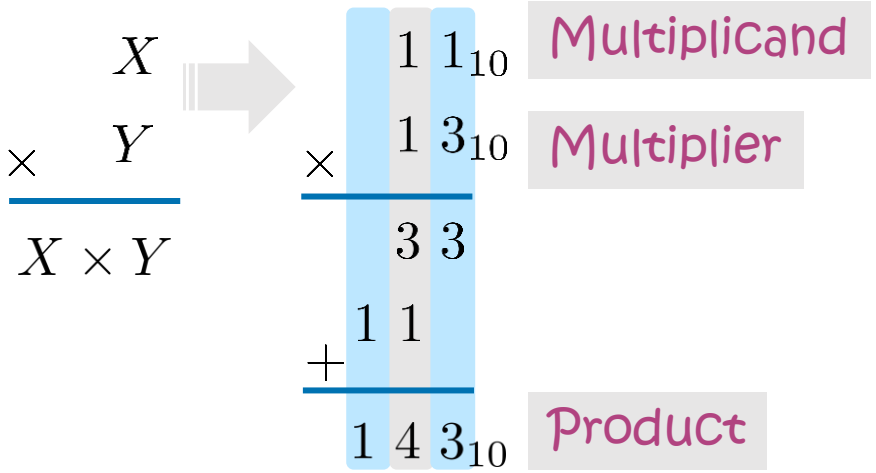


Binary Multiplication



Decimal Numbers

Recall the Multiplication by Hand



- Final product is the sum of shifted multiplicands

$$X \cdot Y = X \cdot \sum_{i=0}^{n-1} Y_i \cdot 10^i$$

$$= \sum_{i=0}^{n-1} Y_i \cdot X \cdot 10^i$$

- Example:

$$\begin{aligned} 11_{10} \cdot 13_{10} &= 11 \cdot (3 \cdot 10^0 + 1 \cdot 10^1) \\ &= 3 \cdot 11 \cdot 10^0 + 1 \cdot 11 \cdot 10^1 \\ &= 33 + 110 \\ &= 143_{10} \end{aligned}$$

Multiplication of Unsigned Integers

- Applying the same technique to binary representation

$$X \cdot Y = X \cdot \sum_{i=0}^{n-1} Y_i \cdot 2^i$$

$$= \sum_{i=0}^{n-1} X \cdot Y_i \cdot 2^i$$

$$= Y_{n-1} \cdot X \cdot 2^{n-1} + \ldots + Y_2 \cdot X \cdot 2^2 + Y_1 \cdot X \cdot 2^1 + Y_0 \cdot X \cdot 2^0$$

Multiplicand Left-shifted by n-1

Multiplicand Left-shifted by 2

Multiplicand Left-shifted by 1

Multiplicand

Multiplication of Unsigned Integers

Example

- Same idea, except we add each shifted multiplicand once at a time; so-called **shift-and-add algorithm**; hardware-friendly

$$
\begin{array}{rl}
 & 1\ 1_{10} \quad \text{Multiplicand} \\
\times & 1\ 3_{10} \quad \text{Multiplier} \\
\hline
 & 1\ 4\ 3_{10} \quad \text{Result}
\end{array}
$$

$$
\begin{aligned}
143_{10} &= 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1_2 \\
&= 128 + 8 + 4 + 2 + 1
\end{aligned}
$$

$$
\begin{array}{rll}
 & 1\ 0\ 1\ 1 & \text{Multiplicand} \\
\times & 1\ 1\ 0\ 1 & \text{Multiplier} \\
\hline
 & 0\ 0\ 0\ 0 & \text{First partial product (always zero)} \\
+ & 1\ 0\ 1\ 1 & 1 \text{ x multiplicand} \\
\hline
 & 0\ 1\ 0\ 1\ 1 & \text{Partial product} \\
+ & 0\ 0\ 0\ 0 & 0 \text{ x multiplicand, then left-shifted by 1 place} \\
\hline
 & 0\ 0\ 1\ 0\ 1\ 1 & \text{Partial product} \\
+ & 1\ 0\ 1\ 1 & 1 \text{ x multiplicand, then left-shifted by 2 places} \\
\hline
 & 0\ 1\ 1\ 0\ 1\ 1\ 1 & \text{Partial product} \\
+ & 1\ 0\ 1\ 1 & 1 \text{ x multiplicand, then left-shifted by 3 places} \\
\hline
 & 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1 & \text{Result}
\end{array}
$$

When Multiplying Two Integers…

- …one with n-bits and one with m-bits, at most how many bits are required to represent the resulting product?

- **A**: n + m

  Hint: multiplication is a sequence of m-additions (the number of bits of the multiplier) with an n-bit number (the number of bits of the multiplicand). Every intermediate result requires one bit more than the previous.

Two's Complement Multiplication

Recall the value in two's complement

$$x = -X_{n-1}2^{n-1} + \sum_{i=0}^{n-2} X_i 2^i$$

- Inspired by the previous algorithm

$$X \cdot Y = X \cdot (-Y_{n-1} \cdot 2^{n-1}) + X \sum_{i=0}^{n-2} Y_i \cdot 2^i$$

$$= -X \cdot Y_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} X \cdot Y_i \cdot 2^i$$

$$= -Y_{n-1} \cdot X \cdot 2^{n-1} + Y_{n-2} \cdot X \cdot 2^{n-2} \ldots + Y_2 \cdot X \cdot 2^2 + Y_1 \cdot X \cdot 2^1 + Y_0 \cdot X \cdot 2^0$$

Multiplicand Left-shifted by n-1

Multiplicand Left-shifted by n-2

Multiplicand Left-shifted by 2

Multiplicand Left-shifted by 1

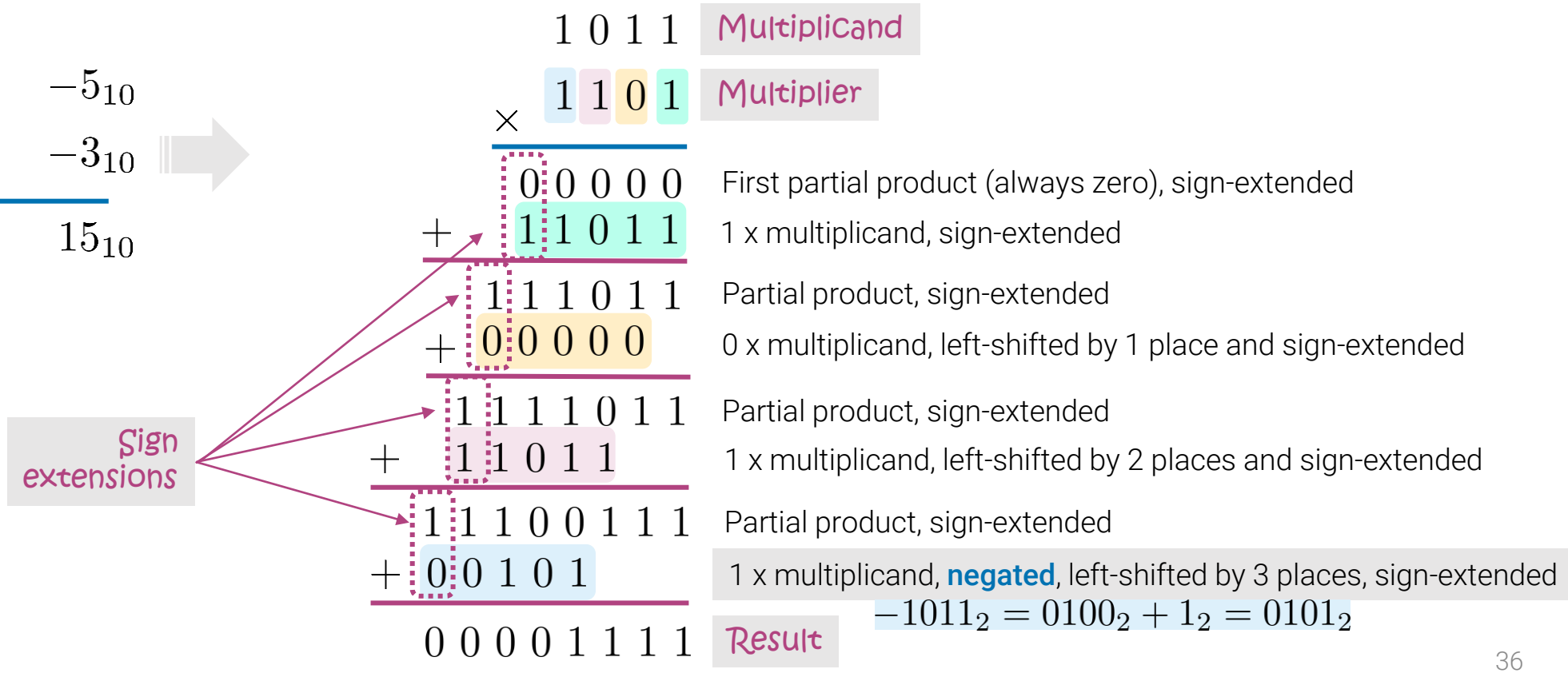
- **Sign**: let us not forget to sign-extend the partial results

Two's Complement Multiplication

Example

- n+m bits are kept; any higher-order bits are discarded

$$\begin{array}{r} X \\ \times\ Y \\ \hline X \times Y \end{array} \qquad \begin{array}{r} -5_{10} \\ \times\ -3_{10} \\ \hline 15_{10} \end{array}$$

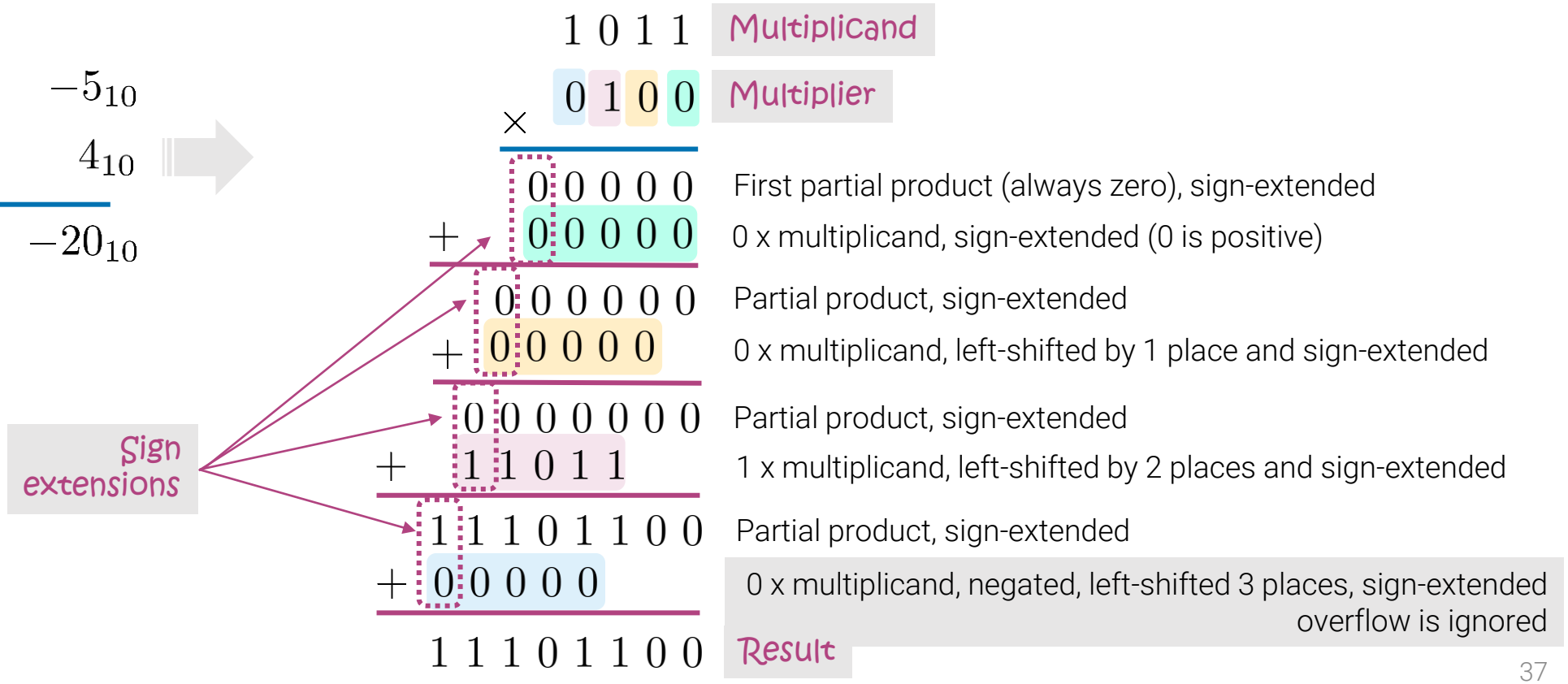


Two's Complement Multiplication

Example

- Every intermediate result must have the correct sign

$$\begin{array}{r} X \\ \times\ Y \\ \hline X \times Y \end{array} \qquad \begin{array}{r} -5_{10} \\ \times\ 4_{10} \\ \hline -20_{10} \end{array}$$

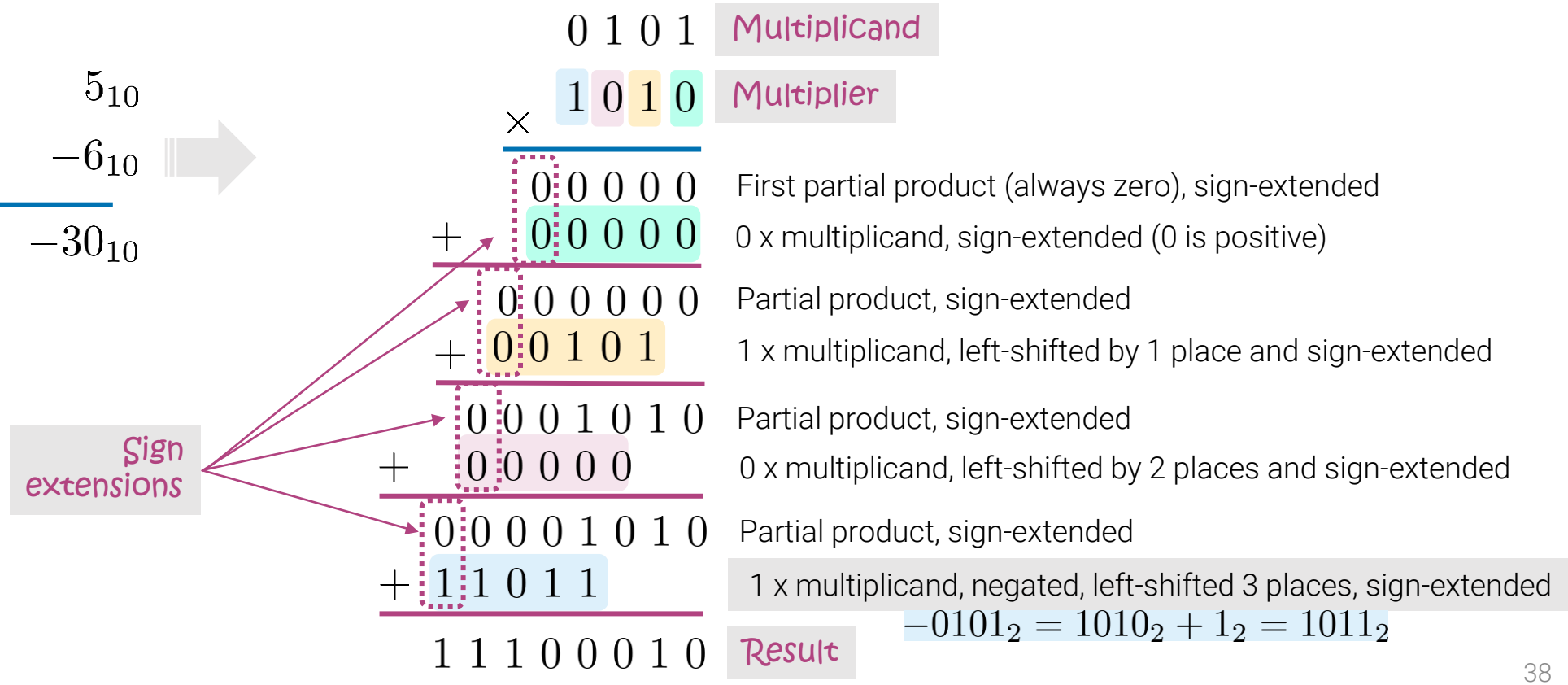


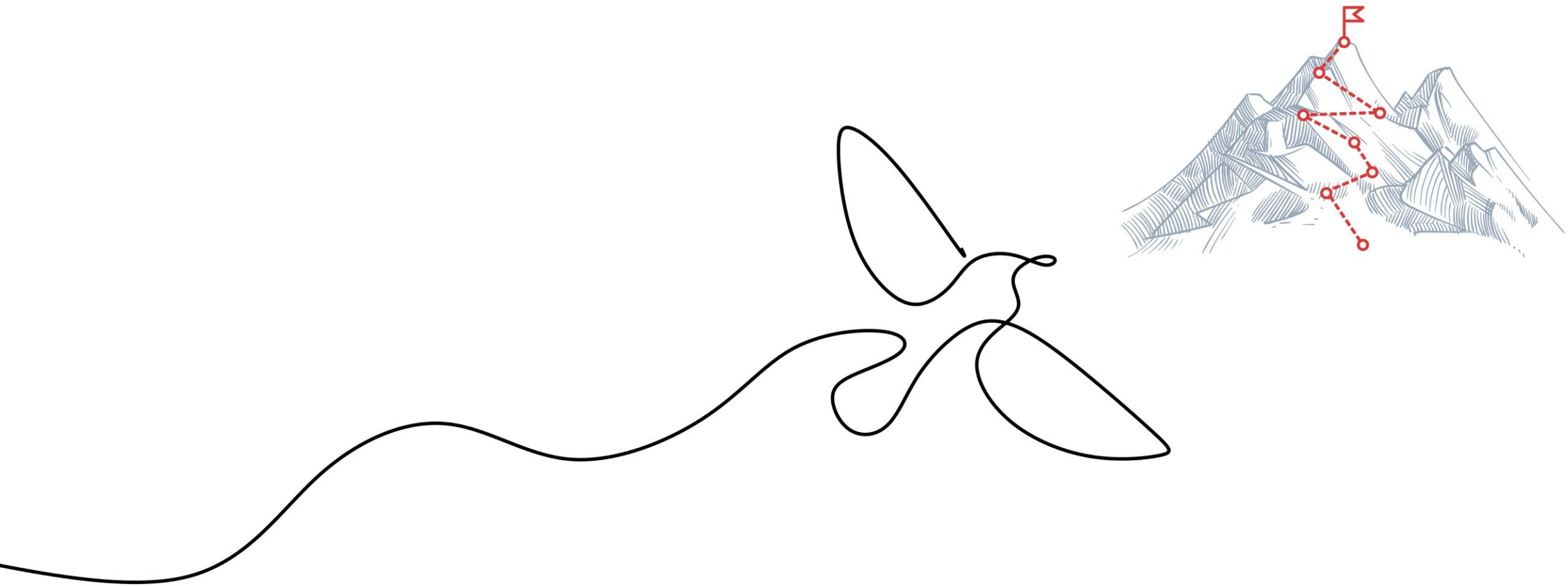
Two's Complement Multiplication

Example

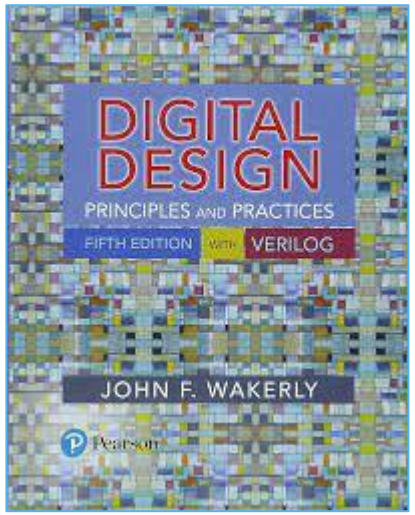
- Every intermediate result must have the correct sign

$$\begin{array}{r} X \\ \times\ Y \\ \hline X \times Y \end{array} \qquad \begin{array}{r} 5_{10} \\ \times\ -6_{10} \\ \hline -30_{10} \end{array}$$

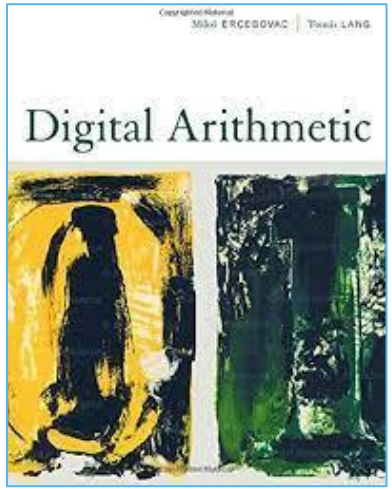




Literature



- Chapter 2: Number Systems and Codes
  - 2.6
  - 2.8



- Chapter 1: Preview of Basic Number Representations and Arithmetic Algorithms
  - 1.3
  - 1.5