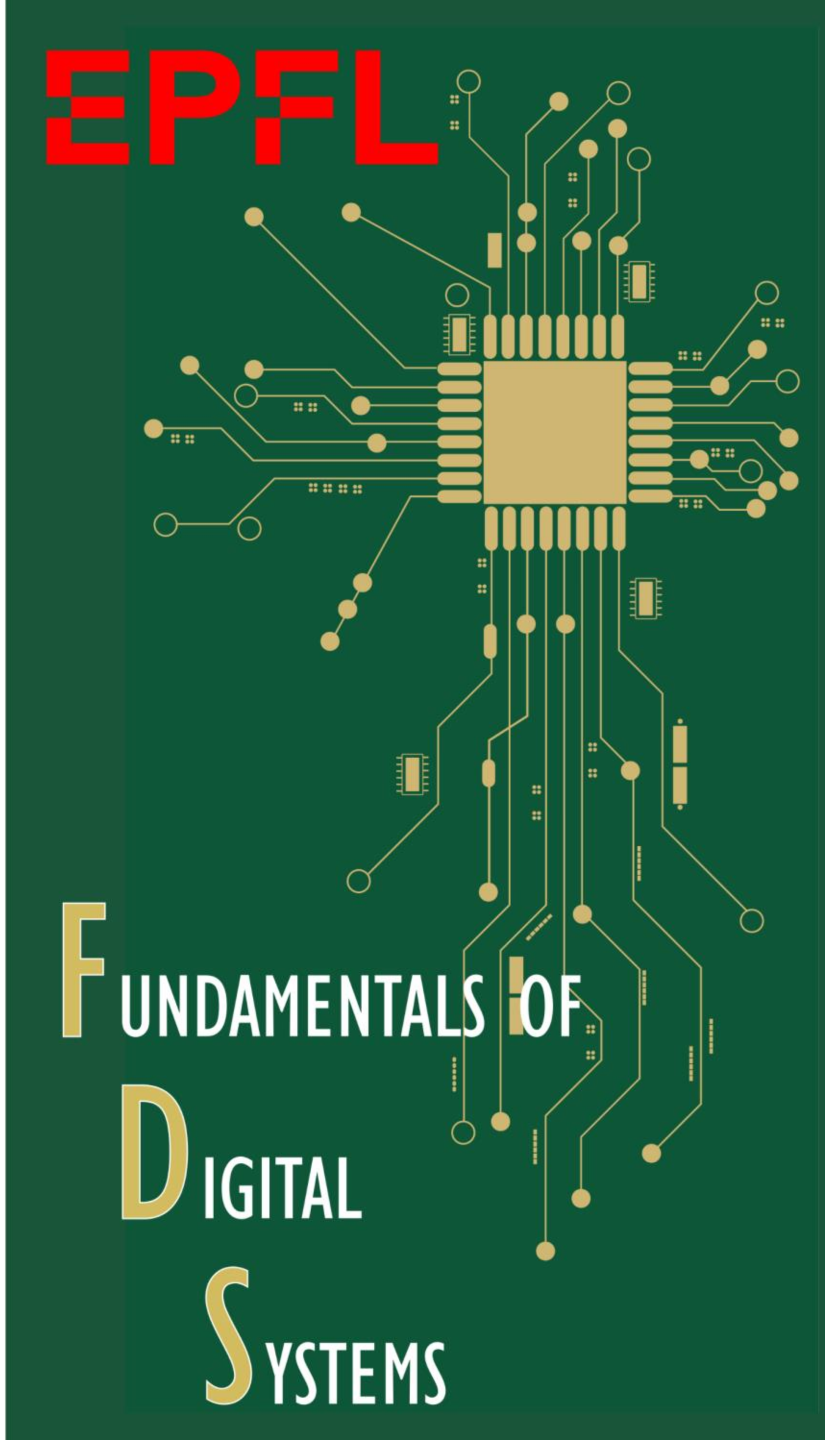


Number Systems

Integers, Bin/Oct/Hex, Codes

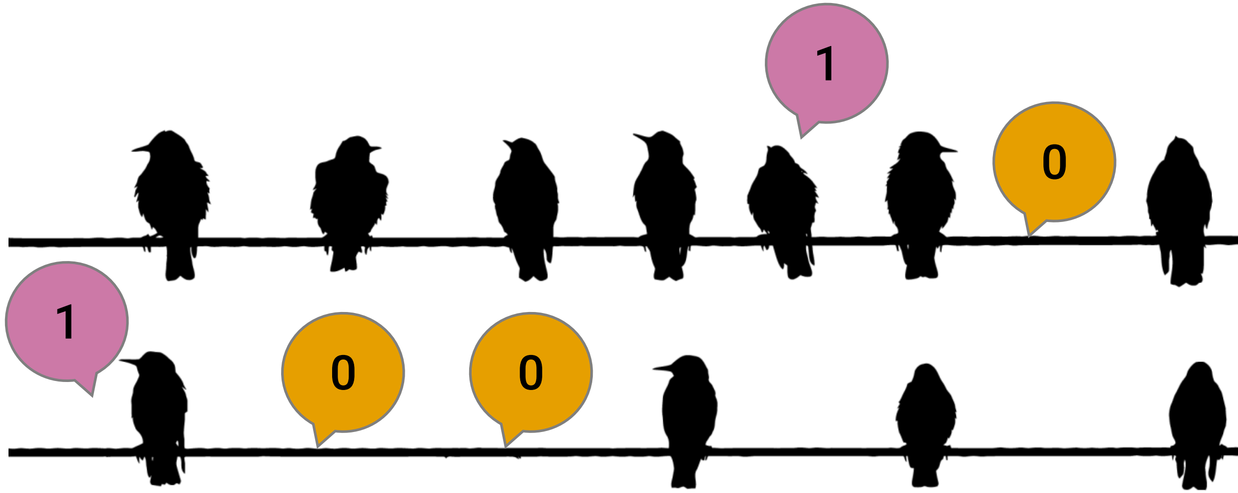
CS-173 Fundamentals of Digital Systems

Mirjana Stojilović

February 2025

A (Little) Bit of Information

- **A bit** is the most basic unit of information in digital computing and communication
- A logical state with one of two possible values (**binary**)

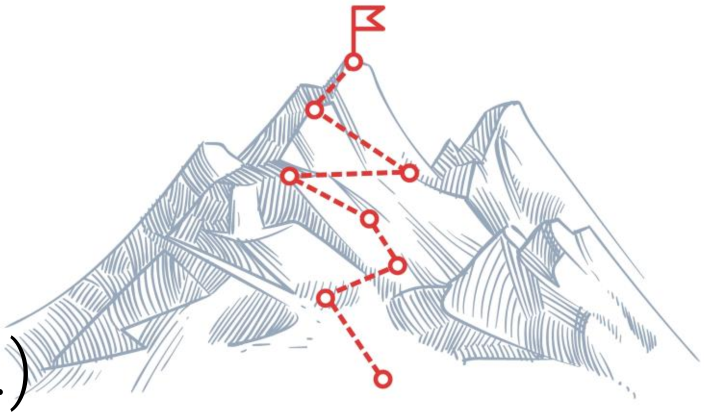


- In modern devices, a bit typically corresponds to an electrical state ON or OFF (charged or discharged, voltage high or low, etc.)
- Bits are small → so we group them into vectors or strings

Let’s Talk About…

…Number systems and codes

Learning Outcomes



- Familiarize with the general characteristics of number systems (radix, weights, digit vectors…)
- Learn to represent decimal numbers as binary numbers
- Discover octal/hex systems and their relation to binary
- Master representations of nonnegative and signed binary numbers
- Perform sign extension and arithmetic shifts
- Discover some alternative number codes

Quick Outline

- Representations of nonnegative integers
- Transformations binary/octal/hex to/from decimal
- Transformations octal/hex to/from binary
- Representations of signed integers
  - Sign-and-magnitude
  - Two's complement
- Range extension and arithmetic shifts
- Hamming, BCD, Gray codes

Digital Representations

- In mathematics, a **tuple** is a finite ordered sequence of elements
  - An **n-tuple** is a tuple of **n** elements, where **n** is a nonnegative integer

- In a **digital representation**, a number is represented by an **ordered n-tuple**
  - Each element of the n-tuple is called a **digit**
  - The n-tuple is called a **digit vector (or string of digits)**
  - The number of digits n is called the **precision** of the representation

Representation of Nonnegative Integers

Integer Digit-Vector

- **Digit-vector (string)** representing the integer x is denoted by

$$X = (X_{n-1}, X_{n-2}, \ldots, X_1, X_0)$$

zero-origin

Leftward-increasing indexing

- **Least-significant** digit (also called low-order digit): X_0
- **Most-significant** digit (also called high-order digit): X_{n-1}

Elements of a Number System

$$X = (X_{n-1}, X_{n-2}, ..., X_1, X_0)$$

- The number system to represent the integer x consists of
  - The number of **digits** n
  - A set of numerical **values** for the digits
    - If a **set of values for a digit** X_i is D_i, the cardinality of D_i is $|D_i|$
  - A rule of **interpretation**
    - Mapping between the set of digit-vector values and the set of integers
- **Set size**
  - The set of integers is a finite set of at most K elements

$$K = \prod_{i=0}^{n-1} |D_i|$$

Elements of a Number System

Example: Decimal Number System

$$X = (X_{n-1}, X_{n-2}, \dots, X_1, X_0)$$

- Number of digits n
  - Can be any, but let us consider n = 6 (e.g., 17, 9899, 676799, …)
  - Leading zeros are irrelevant
- Digit set in decimal number system
  - $D_i = \{0, 1, 2, \dots, 9\}$ of cardinality 10
- The corresponding set size K is one million values, from 0 to $K - 1$
  - $K = \prod_{i=0}^{n-1} 10 = 10^6$

(Non)Redundant Number Systems

- A number system is **nonredundant** if…
  - …each digit-vector represents a **different** integer
  - E.g., the decimal system is nonredundant as every number is unique
- Alternatively, a number system is **redundant** if…
  - …there are integers represented by **more than one** digit-vector

Weighted (Positional) Number Systems

- Most frequently used number systems are **weighted systems**
- The rule of representation:

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

where $W = (W_{n-1}, W_{n-2}, ..., W_1, W_0)$ is the **weight-vector** of size n

- Equivalent formulation

$$x = X_{n-1}W_{n-1} + X_{n-2}W_{n-2} + \cdots + X_1W_1 + X_0W_0$$

Weighted (Positional) Number Systems

Example: Decimal Number System

- Weights are a power of 10. Example:
  - Digit vector $X = (8, 5, 4, 7, 0, 3)$
  - Weight vector $W = (10^5, 10^4, 10^3, 10^2, 10^1, 10^0)$

$$x = 8 \times 10^5 + 5 \times 10^4 + 4 \times 10^3 + 7 \times 10^2 + 0 \times 10^1 + 3 \times 10^0$$
$$x = 854703_{10}$$

$$W_0 = 1$$
$$W_i = W_{i-1} \times R_{i-1}$$

- When weights are of the format
  - $W_0 = 1$ and
  - $W_i = W_{i-1}R_{i-1}, \; 1 \leq i \leq n-1$

we have a **radix number system**

Radix Number Systems

- …are weighted number system in which the weight vector is related to the **radix vector** $R = (R_{n-1}, R_{n-2}, \ldots, R_1, R_0)$ as follows

$$W_0 = 1; \quad W_i = W_{i-1} R_{i-1}, \; 1 \leq i \leq n-1$$

- Equivalent to

$$W_0 = 1; \;\; W_i = \prod_{j=0}^{i-1} R_j$$

- E.g., in the decimal number system $W_0 = 1; \, W_i = \prod_{j=0}^{i-1} 10$

Fixed- and Mixed-Radix Number Systems

- In a **fixed-radix** system, all elements of the radix-vector have the same value **r (the radix)**
- The weight vector in a fixed-radix system

$$W = (r^{n-1}, r^{n-2}, ..., r^2, r^1, 1)$$

and the integer x becomes

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

- In a mixed-radix system, the elements of the radix-vector differ

Radix Number Systems

Example: Decimal Number System

- Characteristics of the decimal number system
  - Radix $r = 10$
  - Fixed-radix system

$$W = (10^{n-1}, 10^{n-2}, ..., 10^2, 10^1, 1)$$

$$x = \sum_{i=0}^{n-1} X_i \times 10^i$$

$$854703 = 8 \times 10^5 + 5 \times 10^4 + 4 \times 10^3 + 7 \times 10^2 + 0 \times 10^1 + 3 \times 10^0$$

Q&A Number Systems

What fixed- and mixed-radix systems are most interesting to us?

Fixed

- Decimal – radix 10
- Binary – radix 2
- Octal – radix 8
- Hexadecimal – radix 16

Mixed

- E.g., time representation in terms of hours/minutes/seconds
  - Radix-vector

    $R = (24, 60, 60)$
  - Weight-vector

    $W = (3600, 60, 1)$

Canonical Number Systems

- In a **canonical** number system, the set of values for a digit D_i is

$$D_i = \{0, 1, ..., R_i - 1\}$$

with $|D_i| = R_i$, the corresponding element of the radix vector

- Canonical digit sets with fixed radix:
  - Decimal: {0, 1, ..., 9}; Binary: {0, 1}; Hexadecimal: {0, 1, 2, ..., 15}
- Range of values of x represented with **n fixed-radix-r digits:**

$$0 \leq x \leq r^n - 1$$

Conventional Number Systems

- A system with
  - fixed positive radix r and
  - a canonical set of digit values is called

a radix-r conventional number system

- These are by far the most commonly used number systems

Binary/Octal/Hexadecimal to/from Decimal

Transformations of nonnegative numbers

Conversion Table

Up to 15

- The hexadecimal system supplements 0-9 digits with the letters A-F
- Programming languages often use the prefix **0x** to denote a hexadecimal number

Decimal	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**

EXAMPLES

Transformations

Example: Binary/Decimal

- Converting from binary to decimal

$$10011_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 16 + 2 + 1 = 19_{10}$$

- Converting from decimal to binary
  - Digits can be computed as remainders of the long division by 2

$$179/2 = 89 \text{ remainder } 1$$
$$89/2 = 44 \text{ remainder } 1$$
$$44/2 = 22 \text{ remainder } 0$$
$$22/2 = 11 \text{ remainder } 0$$
$$11/2 = 5 \text{ remainder } 1$$
$$5/2 = 2 \text{ remainder } 1$$
$$2/2 = 1 \text{ remainder } 0$$
$$1/2 = 0 \text{ remainder } 1$$

Least-significant binary digit

Most-significant binary digit

Stop once zero

$$179_{10} = 10110011_2$$

Transformations

Example: Octal/Decimal

- Converting from octal to decimal

$$1357_8 = 1 \cdot 8^3 + 3 \cdot 8^2 + 5 \cdot 8^1 + 7 \cdot 8^0 = 512 + 192 + 40 + 7 = 751_{10}$$

- Converting from decimal to octal
  - Digits can be computed as remainders of the long division by 8

$$751/8 = 93 \text{ remainder } 7$$

Least-significant octal digit

$$93/8 = 11 \text{ remainder } 5$$

$$11/8 = 1 \text{ remainder } 3$$

$$1/8 = 0 \text{ remainder } 1$$

Most-significant octal digit

Stop once zero

$$751_{10} = 1357_8$$

Transformations

Example: Hexadecimal/Decimal

- Converting from hexadecimal to decimal

$$\begin{aligned} \mathrm{A0F52}_{16} &= 10 \cdot 16^4 + 0 \cdot 16^3 + 15 \cdot 16^2 + 5 \cdot 16^1 + 2 \cdot 16^0 \\ &= 655360 + 3840 + 80 + 2 = 659282_{10} \end{aligned}$$

- Converting from decimal to hexadecimal
  - Digits can be computed as remainders of the long division by 16

$$\begin{aligned} &659282/16 = 41205 \text{ remainder } 2 \\ &41205/16 = 2575 \text{ remainder } 5 \\ &2575/16 = 160 \text{ remainder } 15 \\ &160/16 = 10 \text{ remainder } 0 \\ &10/16 = 0 \text{ remainder } 10 \end{aligned}$$

Least-significant hexadecimal digit

Most-significant hexadecimal digit

Stop once zero

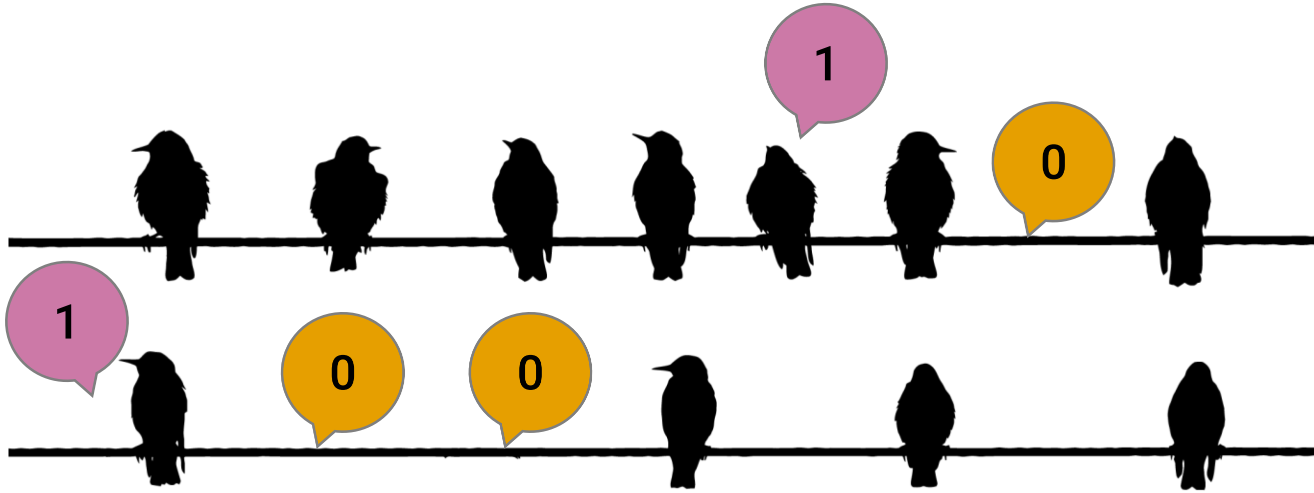
$$659282_{10} = \mathrm{A0F52}_{16}$$

Octal/Hexadecimal to/from Binary

Transformations of nonnegative numbers

Bit-Vector Representation

- For arithmetic operations in radix-2/8/16 digital systems, digit-vectors are represented by bit-vectors



- Methodology
  - A **code** for mapping **a digit to a bit-vector** is defined
  - Digit-vector is obtained by mapping each of its digits following the code

Codes
Bit-Vector Representation

- Binary
  - Digits 0 and 1 are represented by values 0 and 1, resp.
- Power-of-two radix r (octal, hex)
  - Digit d is represented by a bit-vector $(d_{k-1}, \ldots, d_0)$ where $k = log_2 r$ bits, such that

$$d = \sum_{i=0}^{k-1} d_i 2^i$$

  - E.g., digit D_{16} in the hexadecimal (radix-16) format is represented by a 4-bit binary vector/string 1101_2

Transformations

Example: Binary/Octal

- Converting from binary to octal, $k = \log_2 8 = 3$
  - Group every three binary digits into a single octal digit

$$010000100110_2 = 010\ 000\ 100\ 110_2 = 2046_8$$

- Converting from octal to binary
  - Exactly the reverse, expand each octal digit into three binary digits

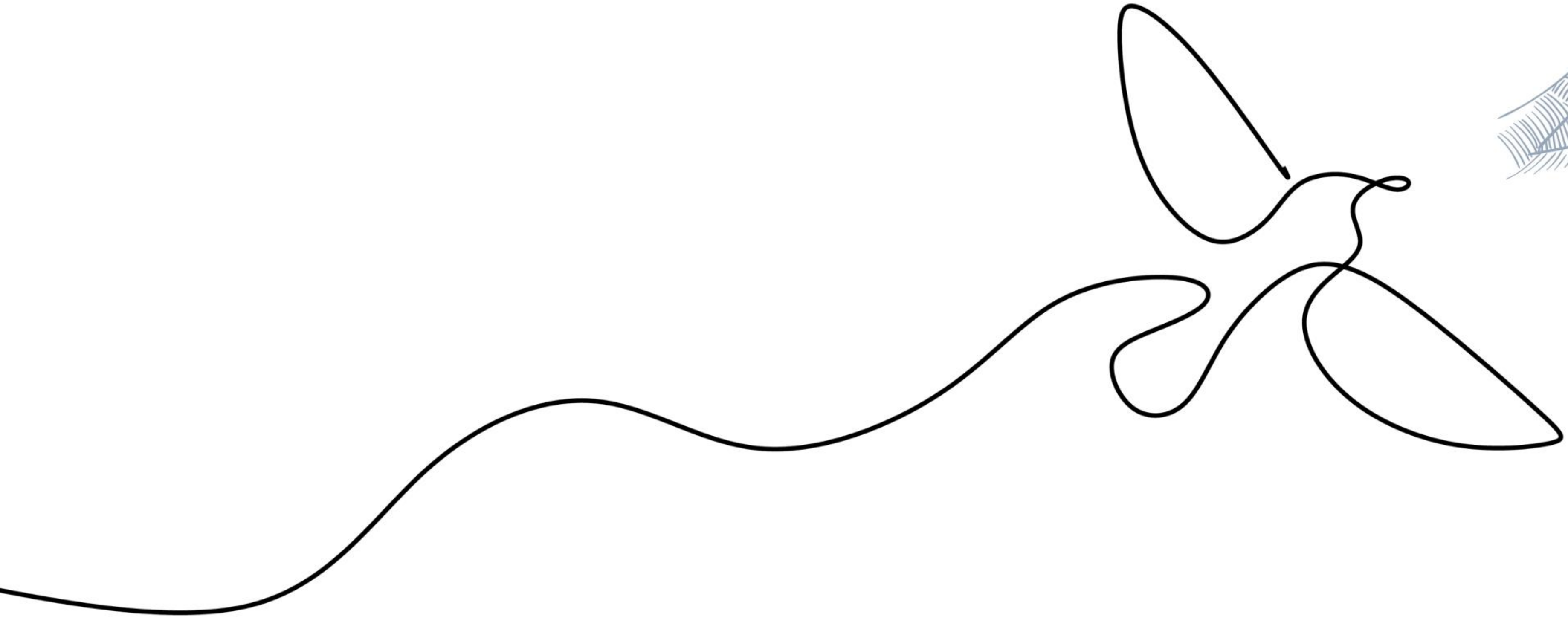
Transformations

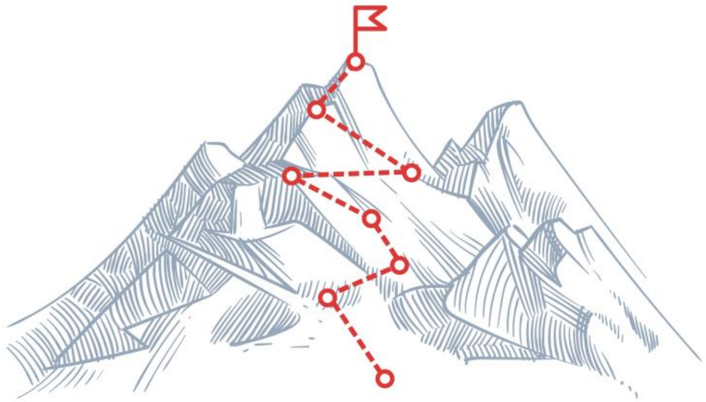
Example: Binary/Hexadecimal

- Converting from binary to hexadecimal, $k = \log_2 16 = 4$
  - Group every four binary digits into a single hexadecimal digit

$$1011111010101101_2 = 1011\ 1110\ 1010\ 1101_2 = \mathrm{BEAD}_{16}$$

- Converting from hexadecimal to binary
  - Exactly the reverse, expand each hex digit into four binary digits



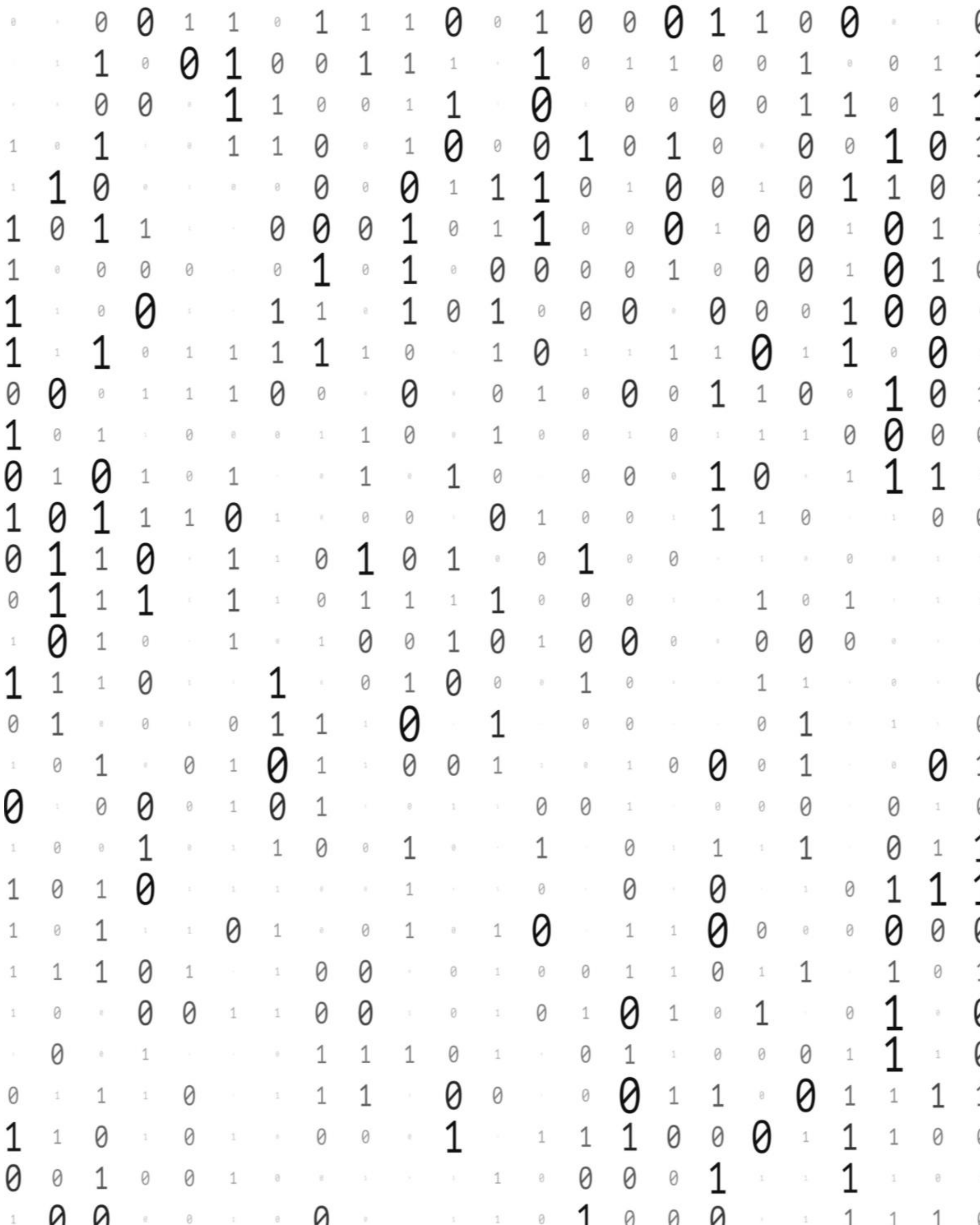


Representation of Signed Integers

Signed ~ Positive and Negative

- Sign and magnitude
- True and complement

Sign-and-Magnitude



Sign-and-Magnitude (SM)

- A signed integer x is represented by a pair

$$(x_s, x_m)$$

  where x_s is the **sign** and x_m is the **magnitude** (positive integer)
- Sign (positive, negative) is represented by a binary variable
  - 0→ positive; 1 → negative
- Magnitude can be represented as any positive integer
  - In a conventional radix-r system, the **range of n-digit magnitude** is

$$0 \leq x_m \leq r^n - 1$$

How is Zero Represented in SM?

- Two representations
  - Positive zero

$$x_s = 0, x_m = 0$$

  - Negative zero

$$x_s = 1, x_m = 0$$

- Is SM a redundant or nonredundant number system?

Sign-and-Magnitude

Examples

- Traditionally, the most-significant bit of a binary bit string is used as the sign bit
- Examples:

$$01010101_2 = +85_{10} \qquad 11010101_2 = -85_{10}$$
$$01111111_2 = +127_{10} \qquad 11111111_2 = -127_{10}$$
$$00000000_2 = +0_{10} \qquad 10000000_2 = -0_{10}$$

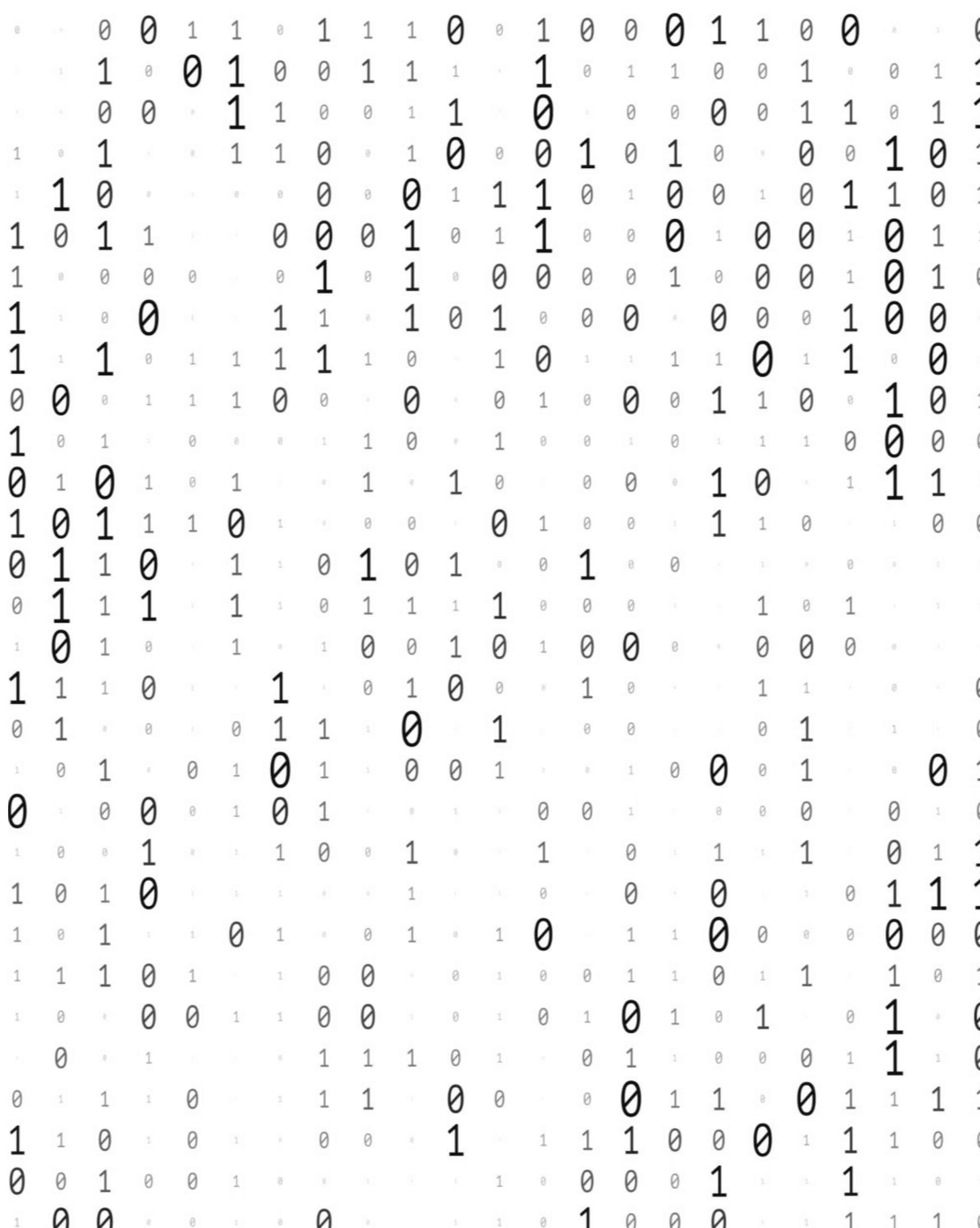
Sign-and-Magnitude
Range

- Symmetrical number system
  - Equal number of positive and negative integers
- An n-bit integer in sign-and-magnitude lies within the range

$$-(2^{n-1} - 1), +(2^{n-1} - 1)$$

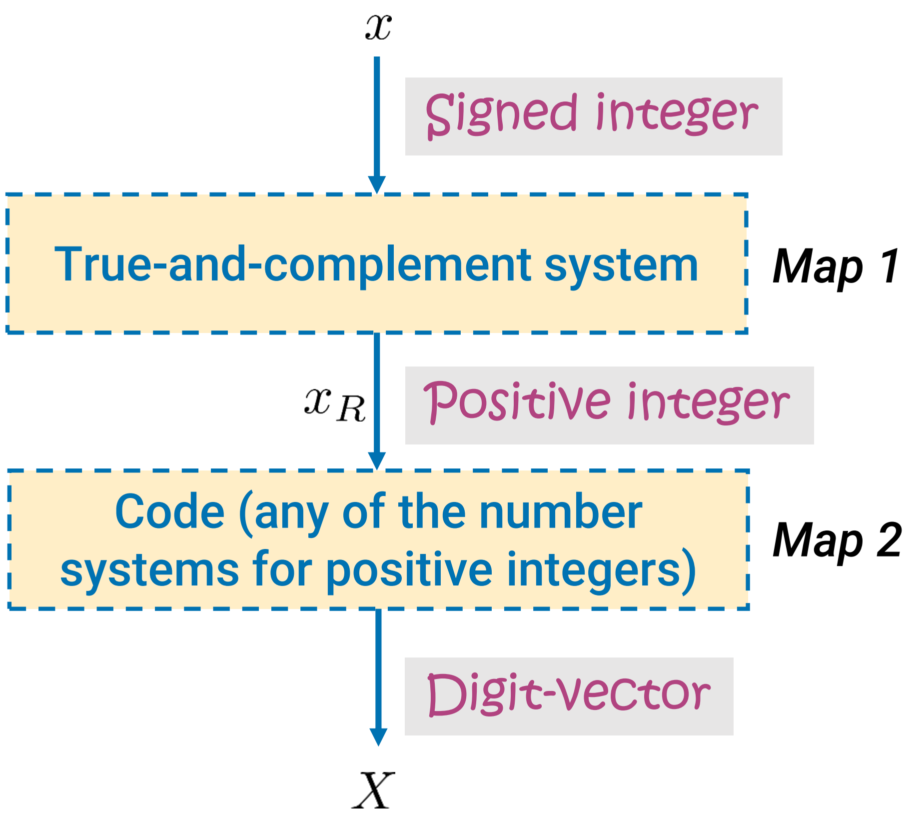
- Main disadvantage of SM: complex digital circuits for arithmetic operations (addition, subtraction, etc.)

True-and-Complement



True-and-Complement (TC)

- No separation between the representation of the sign and the representation of the magnitude
  - **Signed** integer is represented by a **positive** integer



True-and-Complement
Mapping

- A signed integer x is represented by a positive integer x_R :

$$x_R = x \mod C$$

C is a positive integer called the **complementation constant**

- For $|x| < C$, by the definition of the modulo function, we have

$$x_R = \begin{cases} x & \text{if } x \geq 0 \\ C - |x| = C + x & \text{if } x < 0 \end{cases}$$

True form

Complement form

True-and-Complement

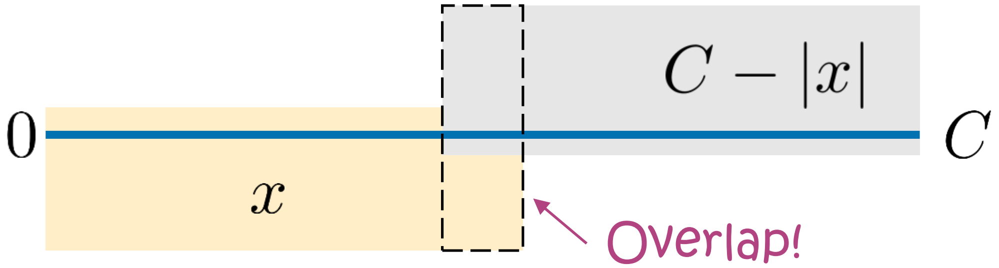
Unambiguous Representation

- Recall:

$$x_R = \begin{cases} x & \text{if } x \geq 0 \\ C - |x| = C + x & \text{if } x < 0 \end{cases}$$

True form

Complement form



- To have an unambiguous representation, the two regions should not overlap, translating to the following condition:

$$\max |x| < C/2$$

True-and-Complement

Converse Mapping

- Converse mapping:

$$x = \begin{cases} x_R & \text{if } x_R < C/2 \\ x_R - C & \text{if } x_R > C/2 \end{cases}$$

Positive values

Negative values

- When $x_R = C/2$, it is usually assigned to $x = -C/2$
  - **Asymmetrical representation**, but simplifies sign detection

- The choice of $C = 2^n$ defines a **two's complement** system

Two's Complement System

- Complementation constant

$$C = 2^n$$

- Range is **asymmetrical**:

$$-2^{n-1} \leq x \leq 2^{n-1} - 1$$

- The representation of zero is unique

x	x_R	
0	0	True forms (positive) $x_R = x$
1	1	
2	2	
…	…	
$2^{n-1} - 1$	$2^{n-1} - 1$	
-2^{n-1}	2^{n-1}	Complement forms (negative) $x_R = 2^n - \lvert x \rvert$
$-(2^{n-1} - 1)$	$2^{n-1} + 1$	
…	…	
-2	$2^n - 2$	
-1	$2^n - 1$	

Sign Detection
in Two's Complement System

- Since $|x| < C/2$ and assuming the sign is 0 for positive and 1 for negative numbers:

$$\mathrm{sign}(x) = \begin{cases} 0 & \text{if } x_R < C/2 \\ 1 & \text{if } x_R \geq C/2 \end{cases}$$

- Therefore, the sign is determined from the most-significant bit:

$$\mathrm{sign}(x) = \begin{cases} 0 & \text{if } X_{n-1} = 0 \\ 1 & \text{if } X_{n-1} = 1 \end{cases}$$; equivalent to $\mathrm{sign}(x) = X_{n-1}$

Mapping from Bit-Vectors to Values

in Two's Complement System

- **Positive** x :

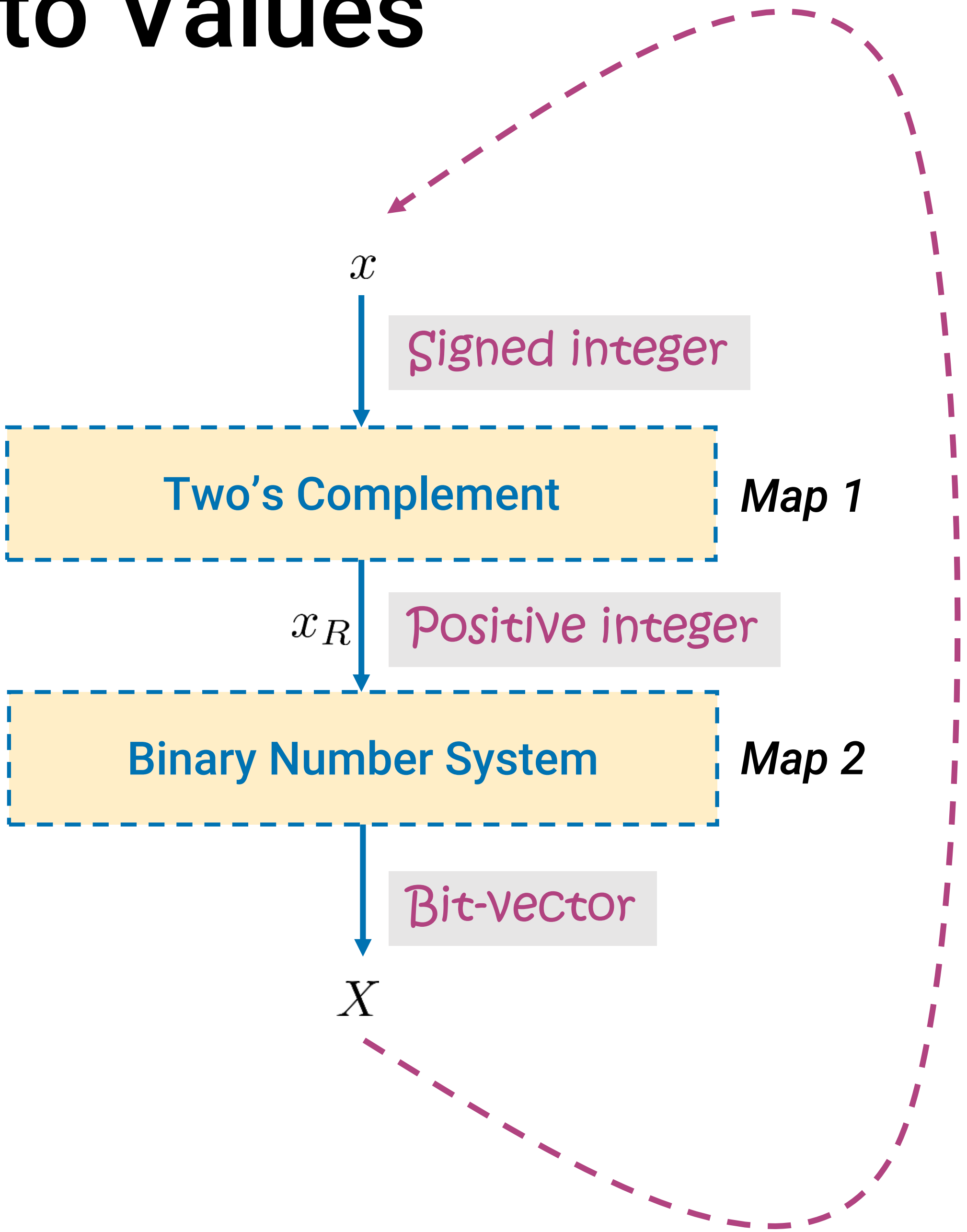
$$x = x_R$$

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

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

ZERO

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



Mapping from Bit-Vectors to Values
in Two's Complement System

- **Negative** x :

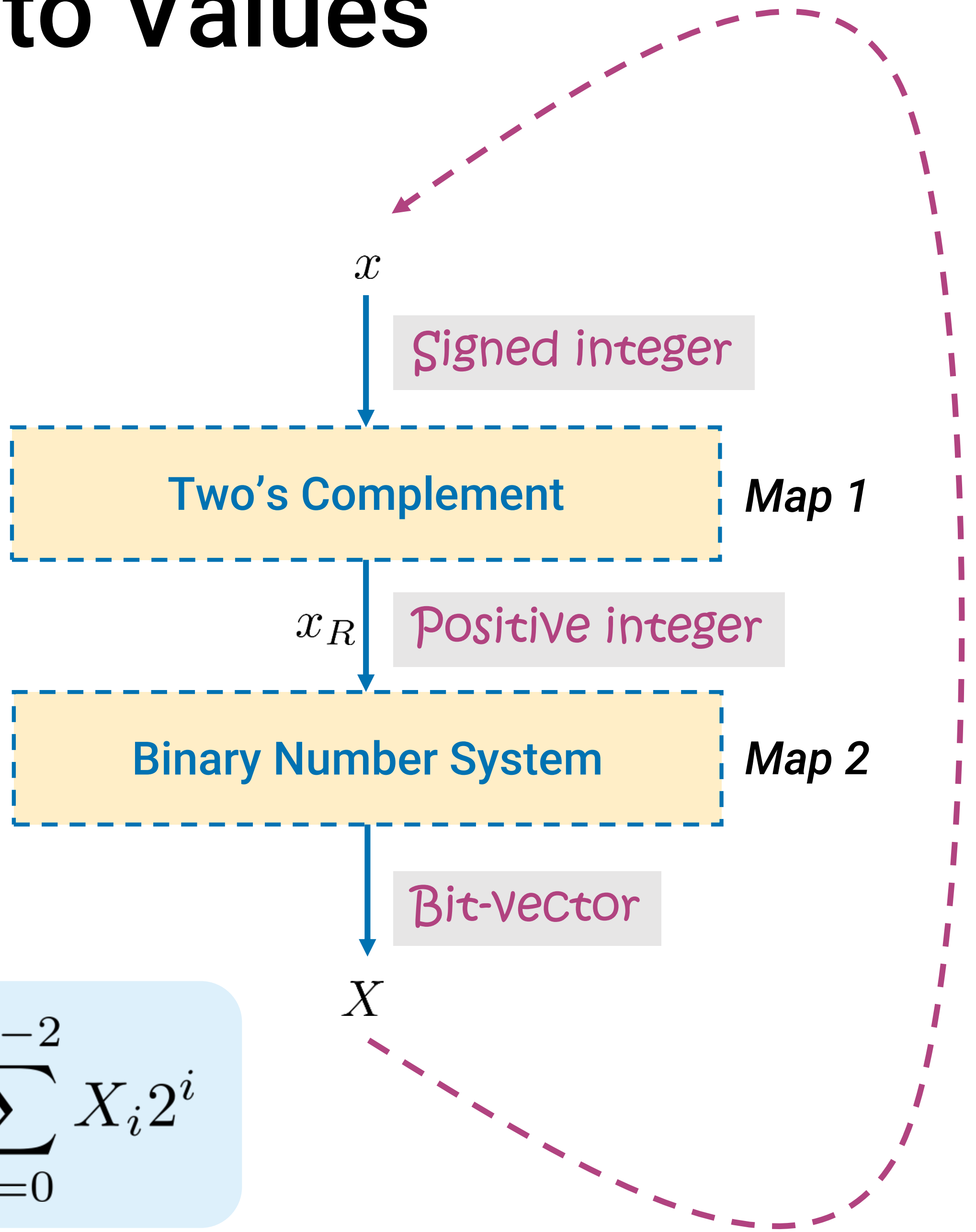
$$x = x_R - C = \sum_{i=0}^{n-1} X_i 2^i - 2^n$$

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

ONE

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

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



Mapping from Bit-Vectors to Values

Example: Two's Complement System

- Examples

$$X = 011011_2 = 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 16 + 8 + 2 + 1 = 27_{10}$$

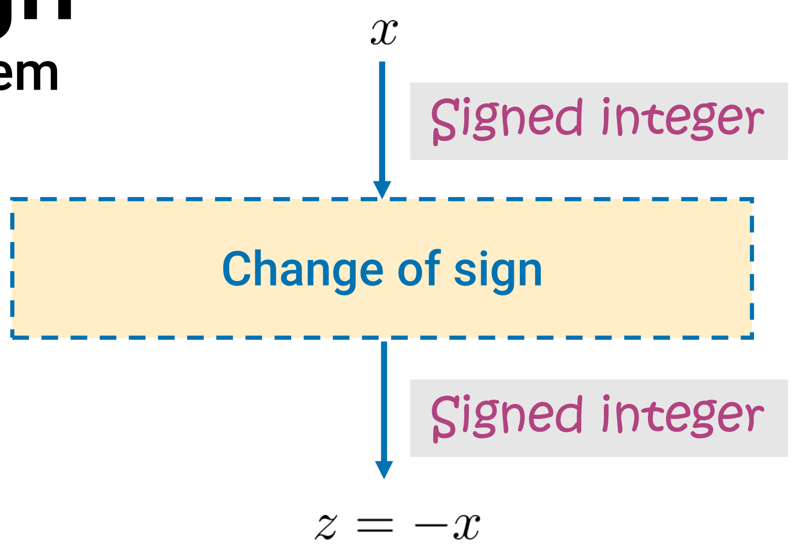
$$X = 11011_2 = -1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = -16 + 8 + 2 + 1 = -5_{10}$$

$$X = 10000000_2 = -1 \cdot 2^7 = -128_{10}$$

$$X = 10000011_2 = -1 \cdot 2^7 + 1 \cdot 2^1 + 1 \cdot 2^0 = -128 + 2 + 1 = -125_{10}$$

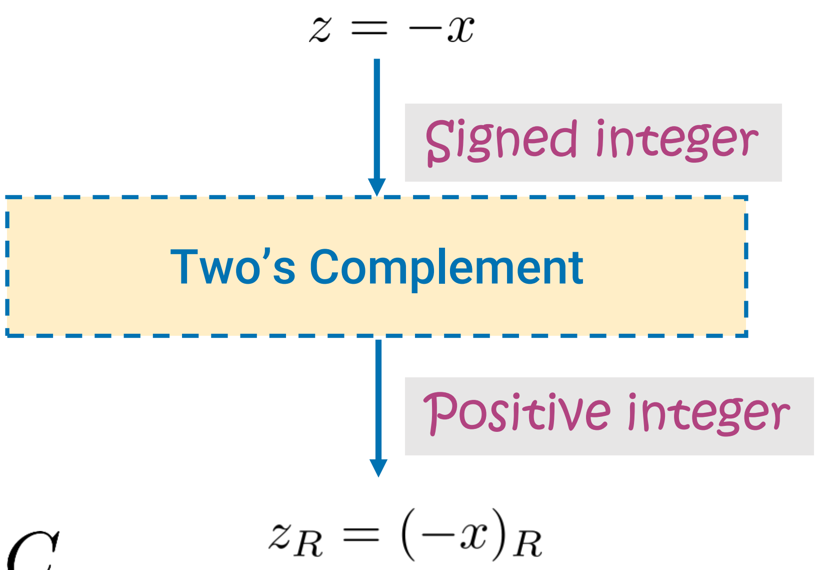
Change of Sign
in Two's Complement System

- Find $z = -x$



- As x and z are represented as x_R and z_R:

$$\begin{aligned} z_R = (-x)_R &= (-x) \mod C \\ &= C - (x \mod C) \\ &= C - x_R \end{aligned}$$



- Therefore, the change of sign operation consists of **subtracting x_R from the complementation constant C**

Change of Sign Algorithm

in Two's Complement System

- Recall: In a two's complement system, the complement of an n-bit number is obtained by subtracting it from 2^n
  - Equivalent to **complementing** each of the n bits and summing with **+1** (proof in literature)

$17_{10} = 00010001_2$

Complement (Change of polarity):

11101110

Add +1:

$+1$

$11101111_2 = -17_{10}$

$-99_{10} = 10011101_2$

Complement (Change of polarity):

01100010

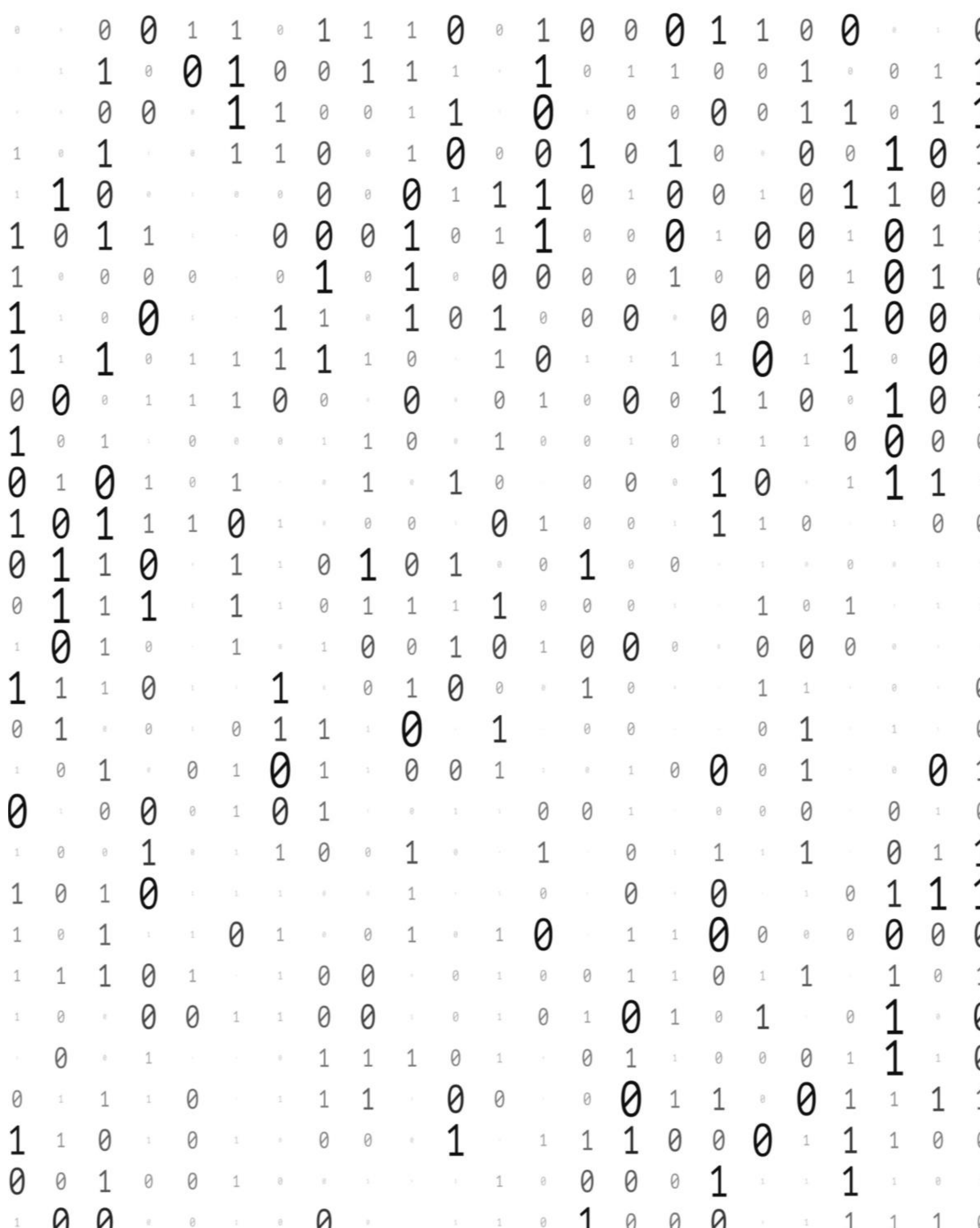
Add +1:

$+1$

$01100011_2 = +99_{10}$

EXAMPLES

Range Extension and Arithmetic Shifts



Range Extension

- Performed when a value x represented by a digit-vector of n bits needs to be represented by a digit-vector of m bits, $m > n$

$$x = z$$
$$X = (X_{n-1}, X_{n-2}, ..., X_1, X_0)$$
$$Z = (Z_{m-1}, Z_{m-2}, ..., Z_1, Z_0)$$
$$m > n$$

- Range extension is often performed in arithmetic operations

Range Extension Algorithm

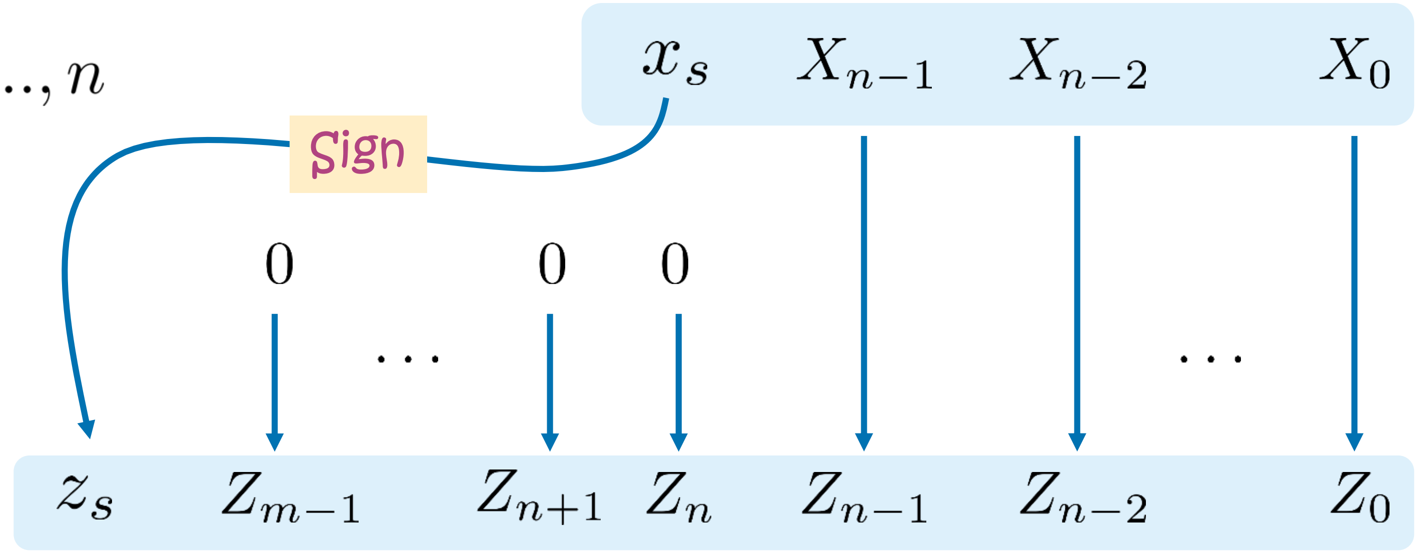
in Sign-and-Magnitude System

- In sign-and-magnitude system,
  the range-extension algorithm becomes

$$z_s = x_s$$ Sign

$$Z_i = 0, \quad i = m-1, m-2, ..., n$$

$$Z_i = X_i, \; i = n-1, ..., 0$$



- Example:

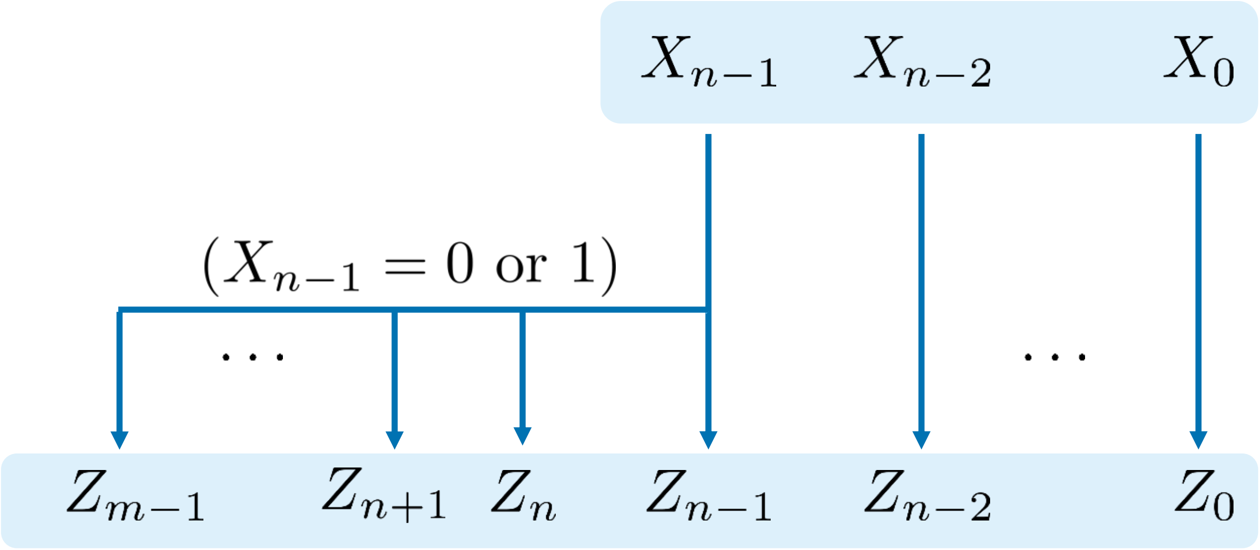
$$X = 1101101_2 = -45_{10}$$
$$X = 100101101_2 = -45_{10}$$

Range Extension Algorithm

in Two's Complement System

- In two's complement system,
  the range-extension algorithm becomes

$$Z_i = X_{n-1},\ \ i = m-1, m-2, ..., n$$
$$Z_i = X_i,\ \ \ i = n-1, ..., 0$$



- Example:

$$X = 10101_2 = -11_{10}$$
$$X = 11110101_2 = -11_{10}$$

Arithmetic Shifts

- Two elementary transformations often used in arithmetic operations are scaling (multiplying and dividing) by the radix
- Conventional radix-2 number system for integers:
  - **Left** arithmetic shift: **multiplication** by 2

$$z = 2x$$

  - **Right** arithmetic shift: **division** by 2

$$z = 2^{-1}x - \epsilon, \qquad |\epsilon| < 1$$

where the value of ϵ is such that it makes z an integer

Left Arithmetic Shift

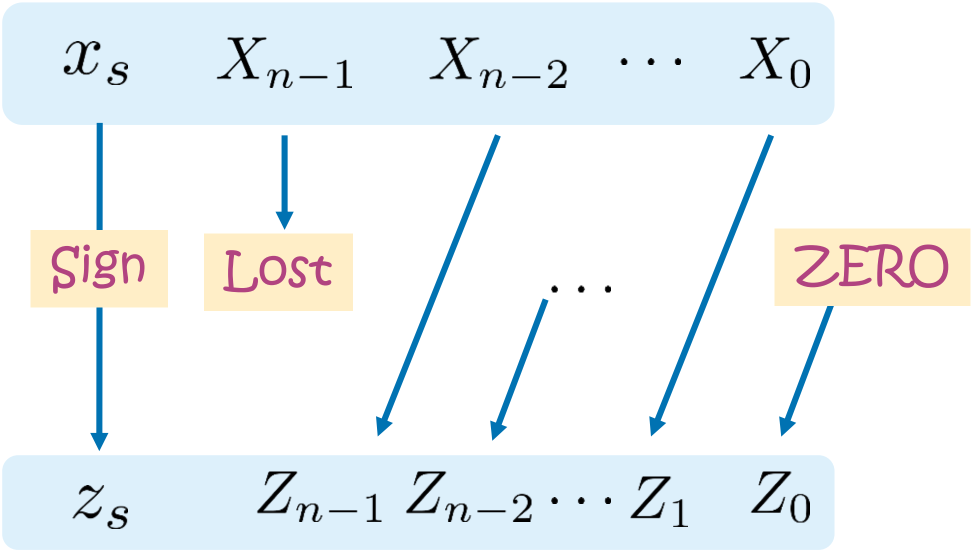
in Sign-and-Magnitude System

- Algorithm (assuming the overflow does not occur)

$$z_s = x_s$$ Sign

$$Z_{i+1} = X_i, \quad i = 0, \dots, n-2$$

$$Z_0 = 0$$



- Example:

$$X = 100101101_2 = -45_{10}$$

$$\mathrm{SL}(X) = 101011010_2 = -90_{10}$$

Right Arithmetic Shift

in Sign-and-Magnitude System

- Algorithm

$$z_s = x_s$$ Sign

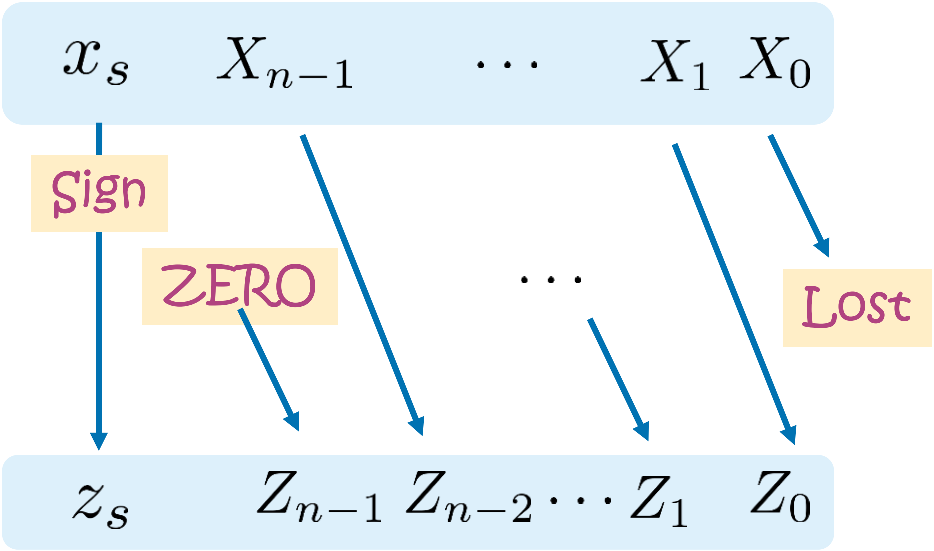
$$Z_{i-1} = X_i, \quad i = 1, ..., n-1$$

$$Z_{n-1} = 0$$

- Example

$$X = 100101101_2 = -45_{10}$$

$$\mathrm{SR}(X) = 100010110_2 = -22_{10}$$

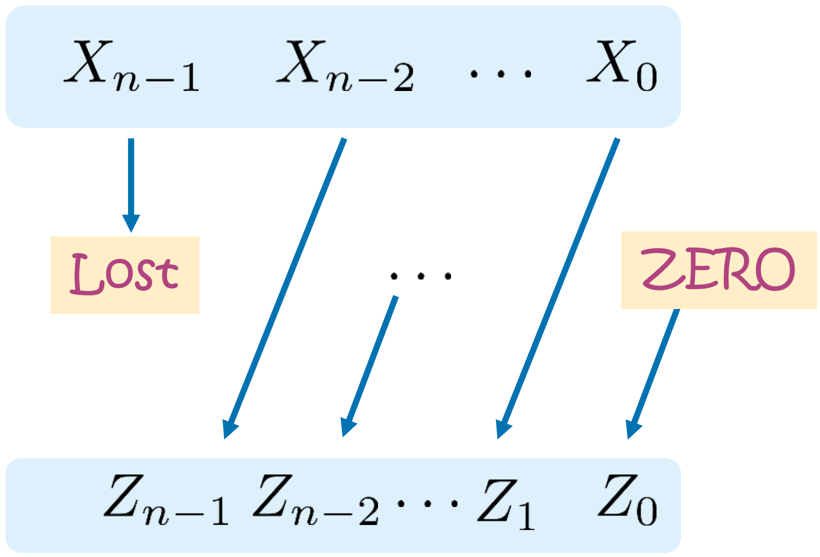


Left Arithmetic Shift

in Two's Complement System

- Algorithm (assuming the overflow does not occur)

$$Z_{i+1} = X_i, \quad i = 0, ..., n-2$$
$$Z_0 = 0$$



- Examples:

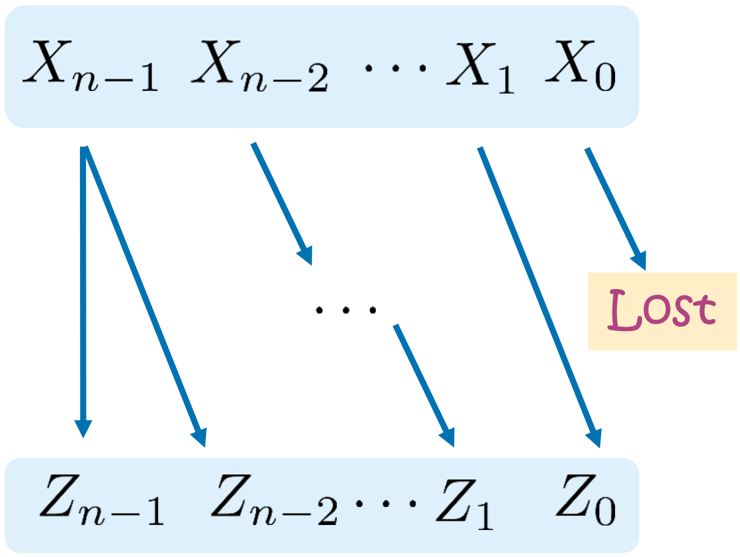
$$X = 001101_2 = 13_{10}$$
$$\text{SL}(X) = 011010_2 = 26_{10}$$
$$Y = 110101_2 = -11_{10}$$
$$\text{SL}(Y) = 101010_2 = -22_{10}$$

Right Arithmetic Shift

in Two's Complement System

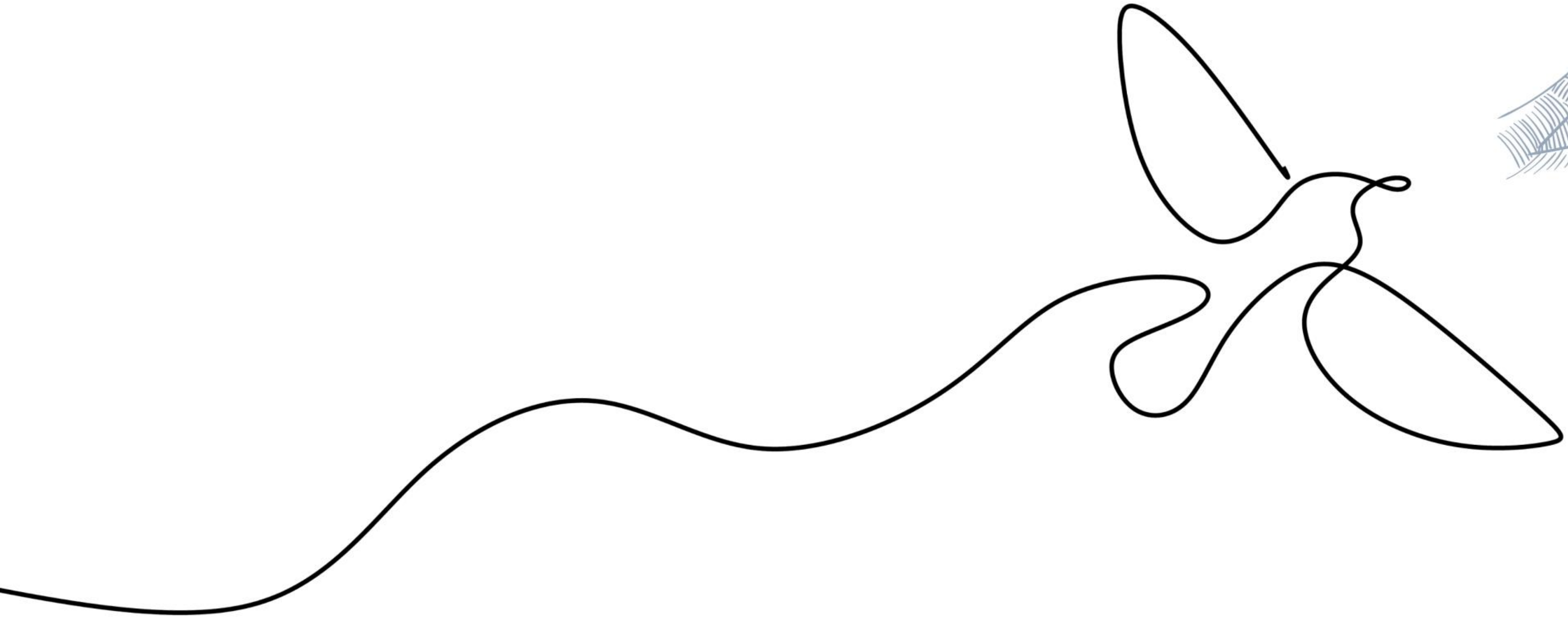
- Algorithm

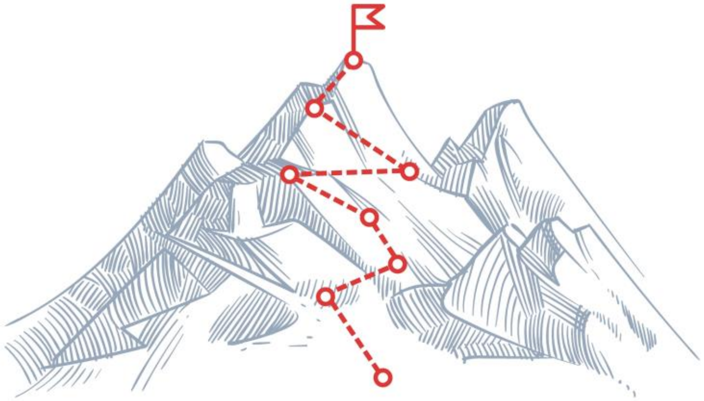
$$Z_{n-1} = X_{n-1}$$
$$Z_{i-1} = X_i, \quad i = 1, \ldots, n-1$$



- Examples:

$$X = 001101_2 = 13_{10}$$
$$\mathrm{SR}(X) = 000110_2 = 6_{10}$$
$$Y = 110101_2 = -11_{10}$$
$$\mathrm{SR}(Y) = 111010_2 = -6_{10}$$





Codes

Alternative Codes

Hamming Weight and Distance

- Named by Richard Hamming, *inventor of error-correcting codes which bear his name, and of the aphorism "The Purpose of computing is insight, not numbers," and many others.*
- Hamming weight
  - The number of binary ones (1) in a bit vector
  - E.g., $\mathrm{HW}(11010101) = 5$
- Hamming distance between two equal-length bit vectors
  - The number of positions in which they differ
  - E.g., $\mathrm{HD}(11010101, 01000111) = 3$

Binary Code for Decimal Numbers (BCD)

Conversion Algorithms

- BCD encodes decimal digits 0 through 9 by their 4-bit unsigned binary representations, 0000 through 1001;
  the code words 1010 through 1111 are not used
- Conversion algorithms:

Given n BCD digits di, compute the corresponding binary value D

```
1: i = n-1; D = 0
2: Multiply D by 10
3: add di to D
4: i = i – 1
5: Go back to line 2 if i>=0
```

Given a binary value D, convert it into the corresponding set of BCD digits

```
1: i = 0;
2: Divide D by 10; D = the quotient
3: di = the remainder
4: i = i + 1
5: Go back to line 2 if i<=n-1
```

Gray Code

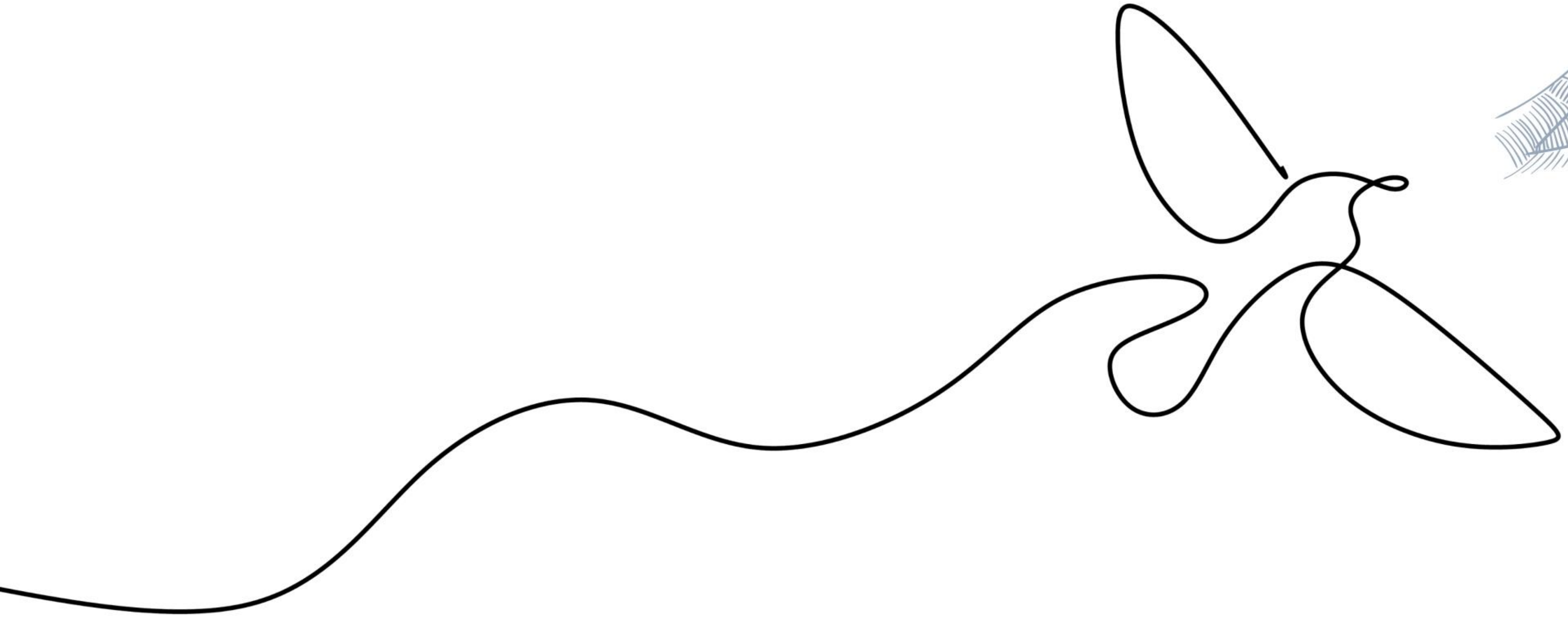
- Invented by Frank Gray, a physicist and researcher at Bell Labs who made numerous innovations in television, both mechanical and electronic, and is remembered for the Gray code.
- Gray code is an ordering of the binary numbers such that **two successive values differ in only one bit**
- Gray codes are widely used to prevent spurious output from electromechanical switches and to facilitate error correction in digital communications

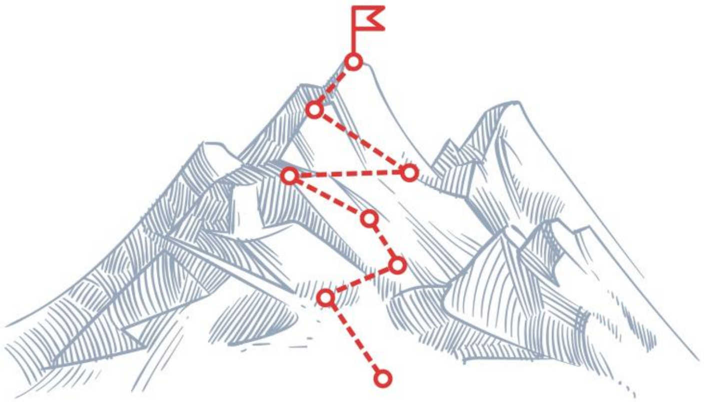
Gray Code
Conversion Algorithm

- Deriving a code word in an n-bit Gray-code from the corresponding n-bit binary code
  - The bits of an n-bit binary or Gray code are numbered from right to left, from 0 to n-1
  - Bit i of a Gray-code vector is 0 if bits i and i+1 of the binary vector are the same; else, bit i is 1;
    - when i+1 = n, bit n of the binary vector is considered to be zero

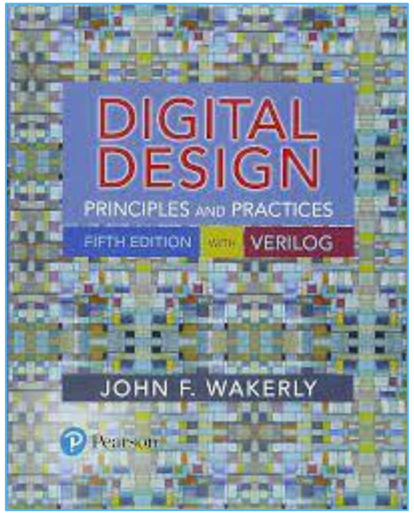
- Comparison of 3-bit binary and Gray codes

Decimal	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

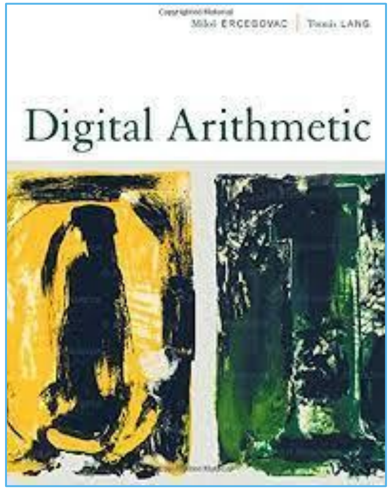




Literature



- Chapter 2: Number Systems and Codes
  - 2.1–2.3
  - 2.5
  - 2.10
  - 2.11



- Chapter 1: Preview of Basic Number Representations and Arithmetic Algorithms
  - 1.1
  - 1.2
  - 1.4

Glossary

- Precision
- Digit-vector
- Least-significant/most-significant bit
- (Non)Redundant
- Weighted
- Radix
- Canonical
- Conventional
- Sign-and-magnitude
- True-and-complement
- Two's complement
- Range extension
- Arithmetic shifts
- Hamming weight
- Hamming distance
- Binary Code for Decimal (BCD)
- Gray code