



LECTURE 04

Introduction to Database Systems

Binary Representation and Boolean Logic

Xiaofan Li

School of Computer Science, Nanjing University



Binary Number System

BASE 2

Binary uses only the digits 0 and 1.

BASE 10

Decimal uses the digits 0 through 9.

Why computers use binary

01

Electronic devices naturally distinguish two stable states: ON and OFF.

02

Logical decisions likewise use TRUE and FALSE.

03

Two-state circuits are simple and reliable to implement.



Bits, Bytes, and Binary Counting

One **bit** is a binary digit; eight bits form one **byte**.

THREE-BIT COUNTING

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

Adding 1 uses carries, as in decimal:
 $011_2 + 1_2 = 100_2$.

CAPACITY

n bits $\longrightarrow 2^n$ patterns

unsigned range: 0 to $2^n - 1$

3 bits: 8 values (0–7). One byte: 10110110.



Decimal to Binary: Integers

Divide by 2; collect the remainders.

1. Divide the integer by 2.
2. Record the remainder: 0 or 1.
3. Repeat with the quotient until it is 0.

Read **bottom to top**: the last remainder is the most significant bit.

Dividend	Quotient	Remainder
1729	864	1
864	432	0
432	216	0
216	108	0
108	54	0
54	27	0
27	13	1
13	6	1
6	3	0
3	1	1
1	0	1

$$1729_{10} = 11011000001_2$$

Check by expanding powers of 2.



Decimal to Binary: Fractions

Multiply by 2; collect the integer bits.

1. Multiply the current fraction by 2.
2. Record the integer part as the next bit.
3. Continue with the fractional part.

Fraction	$\times 2$	Bit
0.718	1.436	1
0.436	0.872	0
0.872	1.744	1
0.744	1.488	1
0.488	0.976	0
0.976	1.952	1
0.952	1.904	1

$$2.718_{10} \approx 10.1011011_2$$

Read **top to bottom**. Stop at zero or the required precision.
The integer part is $2_{10} = 10_2$.

Seven fractional bits give 2.7109375. The approximation sign is essential.



Binary to Decimal: Integers

Multiply each bit by its positional weight, then add.

Decimal: $24 = 2 \cdot 10^1 + 4 \cdot 10^0$. Binary uses powers of **2** instead of 10.

$$b_k 2^k + \dots + b_1 2^1 + b_0 2^0$$

$$\begin{aligned} 11011000001_2 &= 2^{10} + 2^9 + 2^7 + 2^6 + 2^0 \\ &= 1024 + 512 + 128 + 64 + 1 = 1729_{10} \end{aligned}$$

The rightmost position has exponent 0; exponents increase to the left. Zero bits contribute nothing.



Binary to Decimal: Fractions

To the right of the binary point, weights are $2^{-1}, 2^{-2}, 2^{-3}, \dots$

$$.b_1b_2b_3\dots_2 = b_12^{-1} + b_22^{-2} + b_32^{-3} + \dots$$

Example: 10.10110111_2

Integer part: $10_2 = 2$.

Fraction: $2^{-1} + 2^{-3} + 2^{-4} + 2^{-6} + 2^{-7} + 2^{-8}$

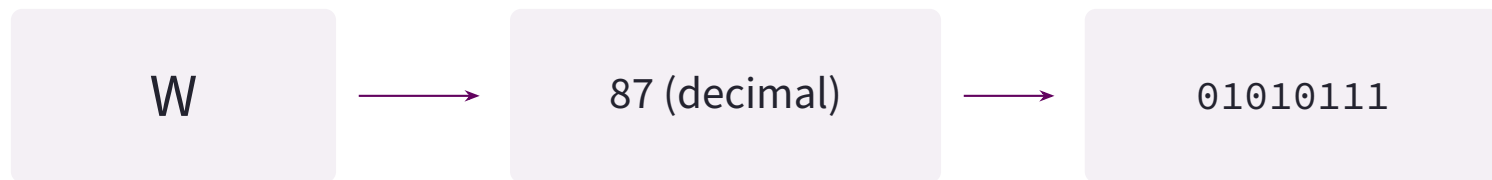
$= 0.5 + 0.125 + 0.0625 + 0.015625 + 0.0078125 + 0.00390625 = 0.71484375$.

Result: 2.71484375_{10} . This example has eight fractional bits; the earlier approximation had seven.



Character Encoding: From W to Bits

An encoding connects written characters to stored bit patterns. ASCII uses 7 bits: $2^7 = 128$ values.



ASCII characters	Codes
Digits 0–9	48–57
Uppercase A–Z	65–90
Lowercase a–z	97–122

ASCII: American Standard Code for Information Interchange.

$87_{10} = 1010111_2$; add a leading zero to store it in one byte.

“Extended ASCII” refers to different 8-bit code pages, not one universal standard. UTF encodings map code points to bytes or code units.



Unicode and Text Encodings

Unicode assigns code points to characters across many writing systems.

Code point

for example, U+0057

UTF-8

1–4 bytes

UTF-16

one or two
16-bit code units

ASCII code points U+0000–U+007F keep the same numeric values in Unicode.



Boolean Algebra

Boolean algebra formalizes operations on truth values. A Boolean variable holds either **0** (FALSE) or **1** (TRUE).

AND

$$A \wedge B$$

OR

$$A \vee B$$

NOT

$$\neg A$$

George Boole gave algebraic form to logic in *An Investigation of the Laws of Thought* (1854).

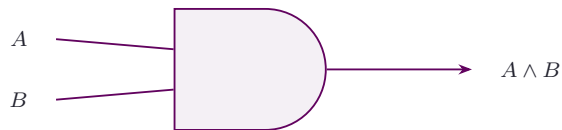
Applications include bit masks, error detection and correction, and memory management.



AND and OR

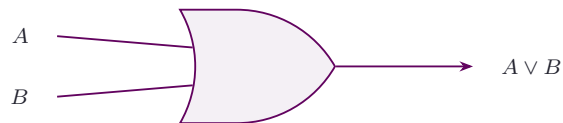
AND, written $A \wedge B$ or $A \cdot B$: true only when both inputs are true.

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1



OR, written $A \vee B$ or $A + B$: true when at least one input is true.

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

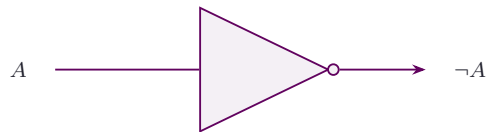




NOT and XOR

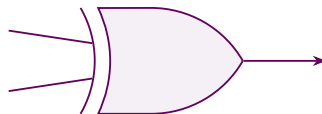
NOT: complements one input.

A	$\neg A$
0	1
1	0



XOR: true exactly when the inputs differ.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



$$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$$



Derived Operations and Boolean Laws

Operation	Definition
NAND	$\neg(A \wedge B)$
NOR	$\neg(A \vee B)$
XNOR	$\neg(A \oplus B)$

DE MORGAN'S LAWS

$$\neg(A \wedge B) = \neg A \vee \neg B$$

$$\neg(A \vee B) = \neg A \wedge \neg B$$

Boolean expressions also obey commutative, associative, distributive, identity, and complement laws. Complex gates are compositions of the basic operations.