Programming Practice for Data Science

Lecture 4: Bit Manipulation in Programming (10/04/24)

Graduate School of Data Science
Seoul National University

공지 사항

- Slack 가입
- Daily Code 제출을 위한 Git 사용 가이드 숙지
  - 매주 월요일 해당 주차 문제 출제, -금요일 일괄 체크

Background for Bit Manipulation

Graduate School of Data Science
Seoul National University

Background for Bit Manipulation

- 1 + 1 = ?

Background for Bit Manipulation

- 1 + 1 = 1 (Boolean Algebra)
- 1 + 1 = 10 (Binary Representation)

Background for Bit Manipulation

- **Boolean Algebra**
  - A branch of mathematics that deals with variables having two possible values and employs logical operations.

Background for Bit Manipulation

- **Boolean Algebra**
  - The values of the variables are the truth values true and false, usually denoted 1 and 0

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Background for Bit Manipulation

- **Boolean Algebra**
  - Boolean algebra uses logical operators such as conjunction (and) denoted as $\wedge$, disjunction (or) denoted as $\vee$, and negation (not) denoted as $\neg$

Logical Operator	Symbol
Not (unary)	~ or ¬
and	$\wedge$
or	$\vee$
if-then	$\rightarrow$
if-and-only-if	$\leftrightarrow$
exclusive or (xor)	$\oplus$

Background for Bit Manipulation

- **Boolean Algebra**
  - Ex. (Top 10% of grades) $\vee$ (A score of 60 or higher) $\rightarrow$ (pass)

Logical Operator	Symbol
Not (unary)	~ or ¬
and	$\wedge$
or	$\vee$
if-then	$\rightarrow$
if-and-only-if	$\leftrightarrow$
exclusive or (xor)	$\oplus$

Background for Bit Manipulation

- **Boolean Algebra**
  - Boolean algebra uses logical operators such as conjunction (and) denoted as $\wedge$, disjunction (or) denoted as $\vee$, and negation (not) denoted as $\neg$

Annulment
$A \wedge 0 = 0$
$A \vee 1 = 1$

Identity
$A \wedge 1 = A$
$A \vee 0 = A$

Idempotent
$A \vee A = A$
$A \wedge A = A$

Complement
$A \vee \neg A = 1$
$A \wedge \neg A = 0$

Double Negation
$\neg(\neg A) = A$

De Morgan's
$\neg(A \wedge B) = \neg A \vee \neg B$
$\neg(A \vee B) = \neg A \wedge \neg B$

Associative
$(A \vee B) \vee C = A \vee (B \vee C)$
$(A \wedge B) \wedge C = A \wedge (B \wedge C)$

Commutative
$A \vee B = B \vee A$
$A \wedge B = B \wedge A$

Distributive
$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$

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

Example

Show that $\neg(q \rightarrow p) \vee (p \wedge q) \equiv$ q

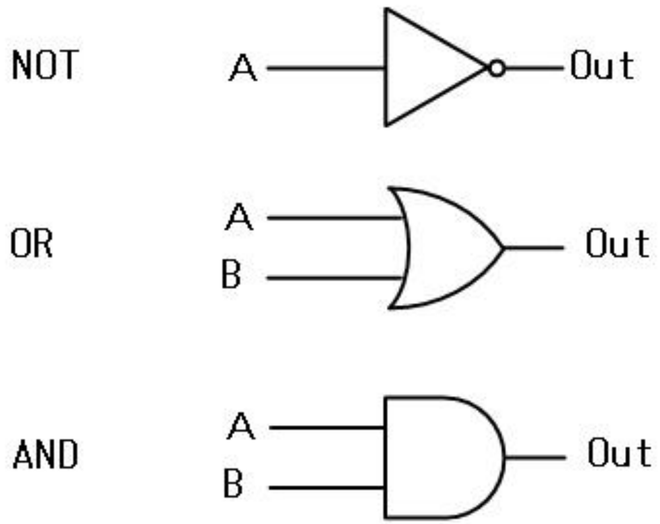
0.	$\neg(q \rightarrow p) \vee (p \wedge q)$	
1.	$\equiv \neg(\neg q \vee p) \vee (p \wedge q)$	Implication Law
2.	$\equiv (q \wedge \neg p) \vee (p \wedge q)$	De Morgan's & Double negation
3.	$\equiv (q \wedge \neg p) \vee (q \wedge p)$	Commutative Law
4.	$\equiv q \wedge (\neg p \vee p)$	Distributive Law
5.	$\equiv q \wedge 1$	Identity Law
	$\equiv q$	Identity Law

Background for Bit Manipulation

- **Boolean Algebra**
  - if ((x > 0 && y != 0) || (x > 0 && y == 0))
  - if (x > 0)

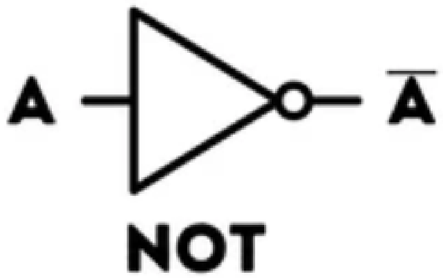
Background for Bit Manipulation

- **Boolean Algebra**
  - Bits operate on binary logic using logical operators



Background for Bit Manipulation

- **Boolean Algebra**
  - NOT



NOT Gate

A	$\overline{A}$
0	1
1	0

Background for Bit Manipulation

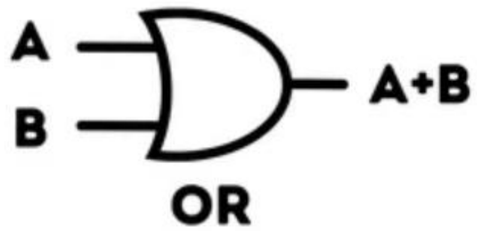
- Boolean Algebra
  - AND

2 input AND Gate

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

Background for Bit Manipulation

- Boolean Algebra
  - OR

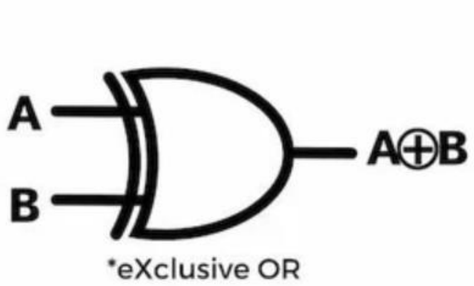


2 input OR Gate

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Background for Bit Manipulation

- Boolean Algebra
  - XOR



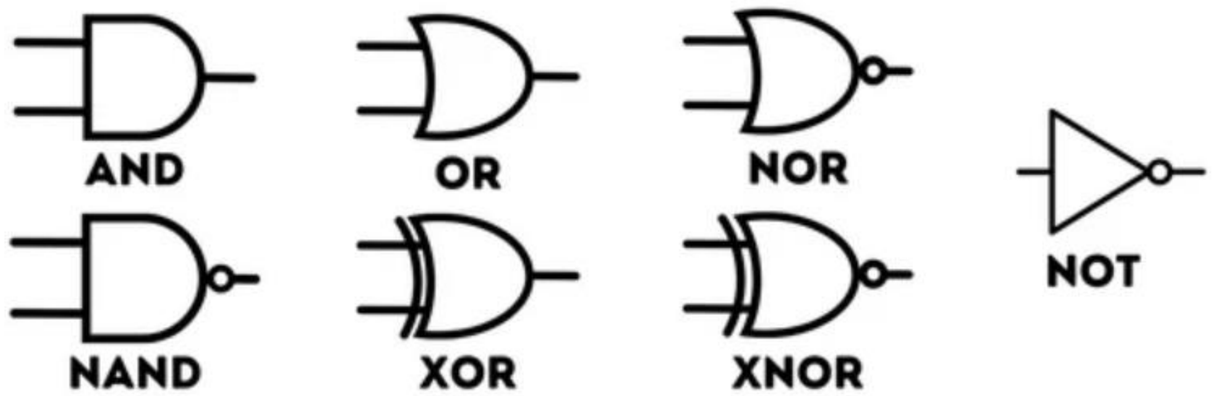
*eXclusive OR

2 input XOR gate

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Background for Bit Manipulation

- **Boolean Algebra**
  - Basic Logic Gate symbols

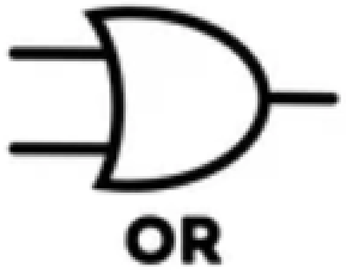


Background for Bit Manipulation

- Ex. (Top 10% of grades) ∨ (A score of 60 or higher)

(Top 10% of grades)

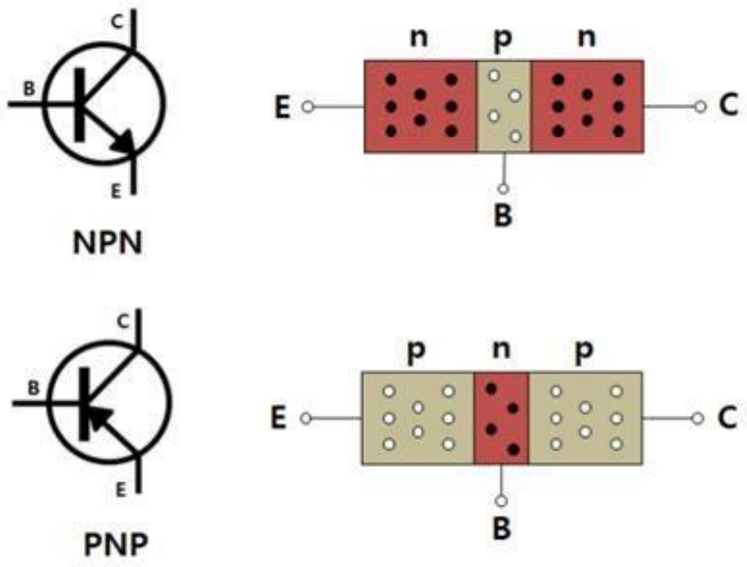
(A score of 60 or higher)



(pass)

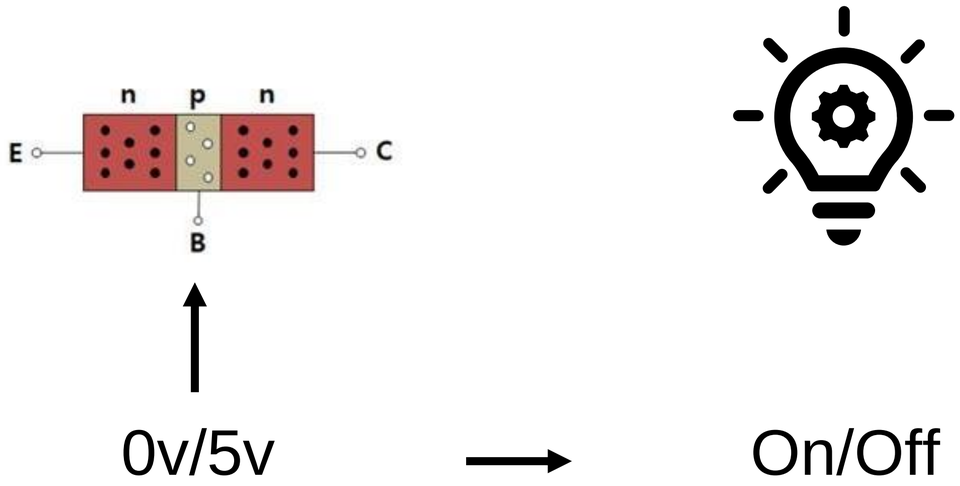
Background for Bit Manipulation

- **Binary Representation**
  - Basic circuit building block : transistor



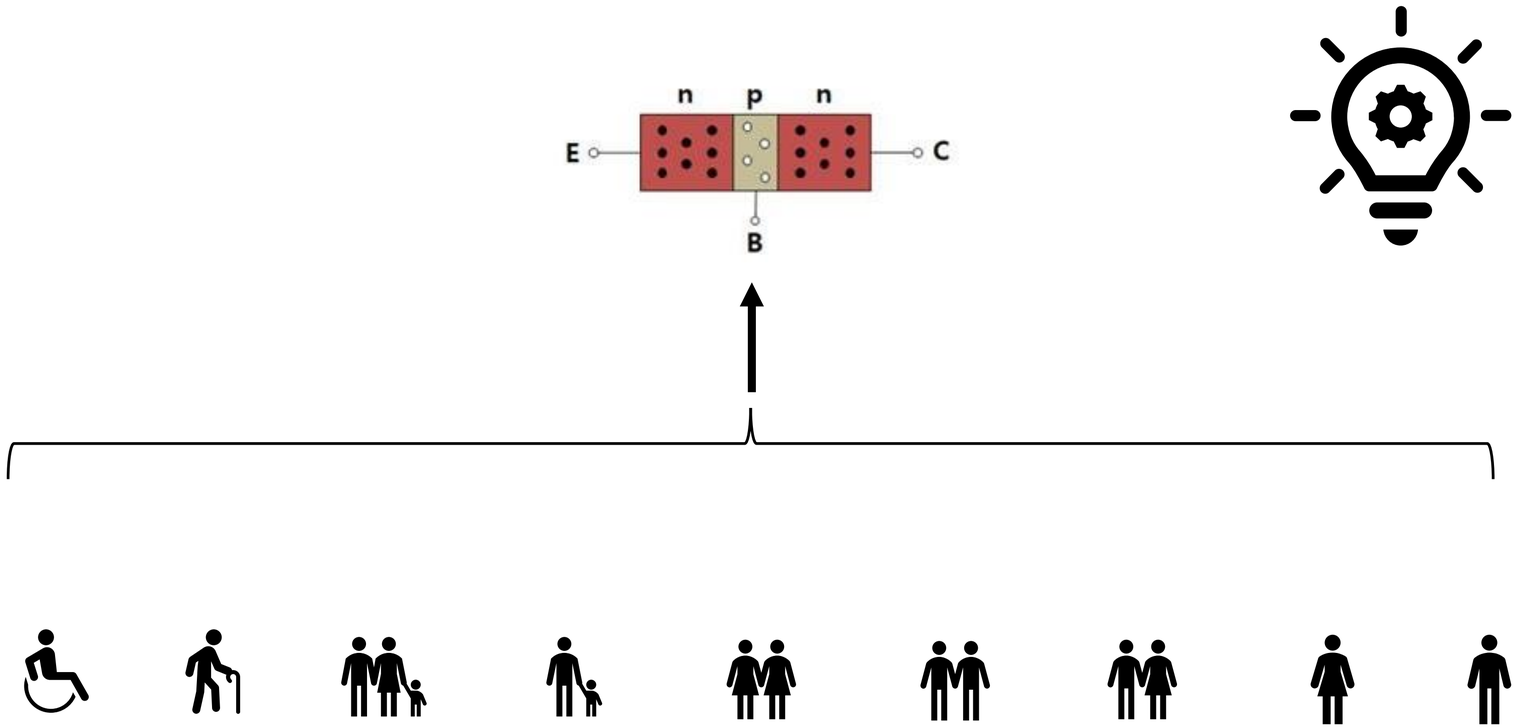
Background for Bit Manipulation

- Binary Representation



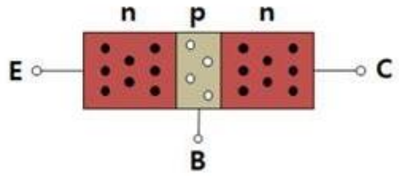
Background for Bit Manipulation

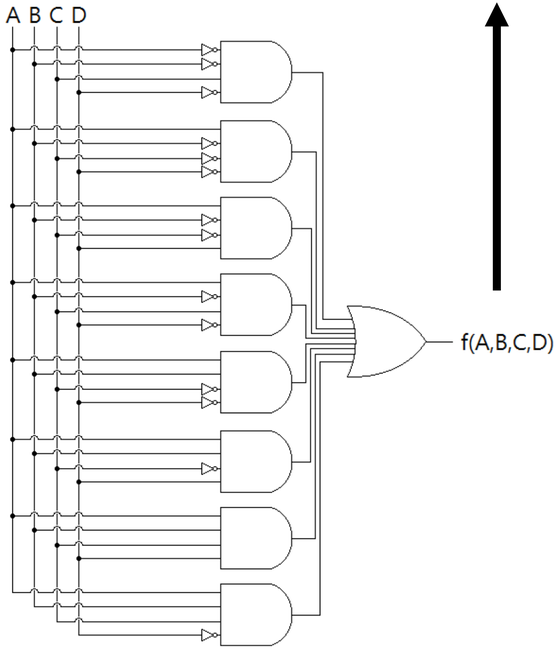
- Binary Representation



Background for Bit Manipulation

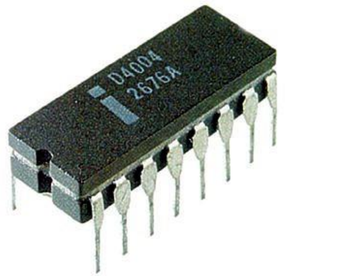
- Binary Representation

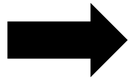


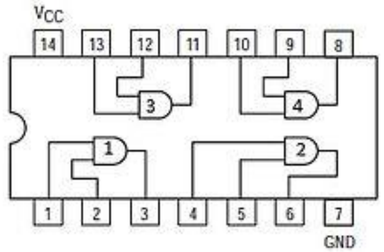


Background for Bit Manipulation

- Binary Representation

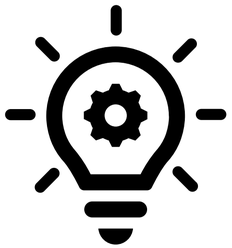


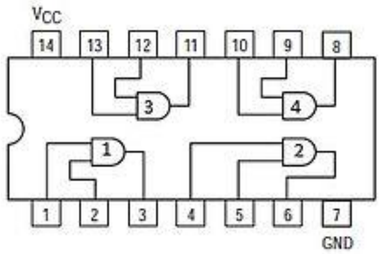




Background for Bit Manipulation

- Binary Representation






```c
#include "stm32f4xx.h"  // STM32F4 HAL 라이브러리 헤더

int main(void) {
    // 1. GPIO 레지스터 설정 (LED가 연결된 핀, 예: GPIOA의 PIN 5)

    // RCC (Reset and Clock Control) 레지스터를 사용해 GPIOA 클럭 활성화
    RCC->AHB1ENR |= (1 << 0);  // AHB1ENR 레지스터의 0번째 비트를 1로 설정 -> GPIOA 클럭 활성화

    // GPIOA의 모드 레지스터 설정: 핀 5를 출력 모드로 설정
    GPIOA->MODER &= ~(3 << (5 * 2));  // 5번 핀의 비트를 클리어
    GPIOA->MODER |= (1 << (5 * 2));   // 5번 핀을 출력 모드로 설정 (01)

    // 2. GPIO 레지스터를 사용해 LED 켜기
    GPIOA->ODR |= (1 << 5);  // ODR 레지스터의 5번째 비트를 1로 설정 -> LED ON

    while (1) {
        // 메인 루프 (LED 상태 유지)
    }
}
```

Background for Bit Manipulation

- **Binary Number**
  - Decimal number : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ⋯
  - Binary Number : 0, 1, 10, ⋯

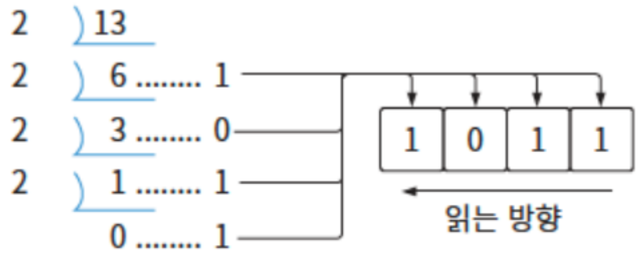
Background for Bit Manipulation

- Binary Number

10진수	2진수
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
⋮	⋮

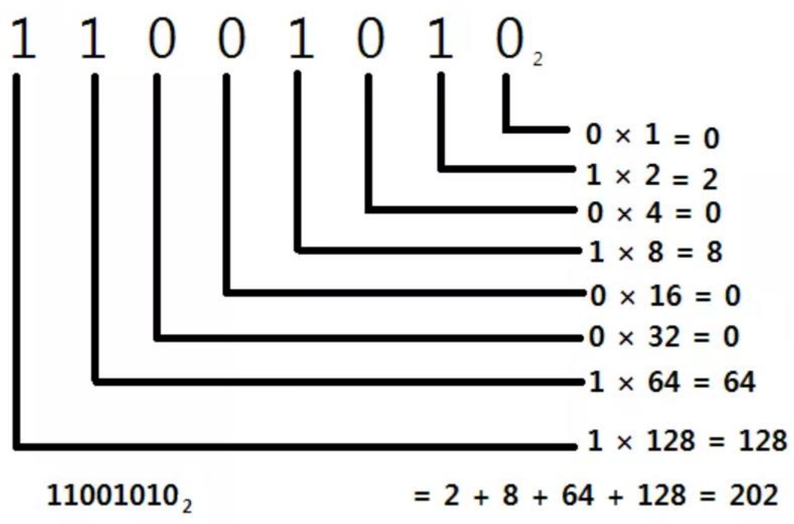
Background for Bit Manipulation

- **Binary Number**
  - Demical → Binary



Background for Bit Manipulation

- **Binary Number**
  - Binary → Demical



Background for Bit Manipulation

- **Binary Number**
  - 1 + 1 = 10 (Binary Representation)

Background for Bit Manipulation

- **Binary Number**
  - Binary (11111111)
  - Octal (377)
  - Hexadecimal (FF)

Bit Manipulation in Programming

- **Radix Prefix**

Python

```python
binary_num = 0b1010      # 이진수 (10진수 10)
octal_num  = 0o12        # 8진수  (10진수 10)
hex_num    = 0xA         # 16진수 (10진수 10)
decimal_num = 10         # 10진수

print(binary_num)   # 출력: 10
print(octal_num)    # 출력: 10
print(hex_num)      # 출력: 10
print(decimal_num)  # 출력: 10

```

C++14

```cpp
#include <iostream>

int main() {
    int binary_num = 0b1010;   // 이진수 (10진수 10)
    int octal_num  = 012;      // 8진수  (10진수 10)
    int hex_num    = 0xA;      // 16진수 (10진수 10)
    int decimal_num = 10;      // 10진수

    std::cout << binary_num << std::endl;  // 출력: 10
    std::cout << octal_num << std::endl;   // 출력: 10
    std::cout << hex_num << std::endl;     // 출력: 10
    std::cout << decimal_num << std::endl; // 출력: 10

    return 0;
}
```

Bit Manipulation in Programming

Graduate School of Data Science
Seoul National University

Bit Manipulation in Programming

- **Bitwise AND Operator (“&” in C, C++ / “&” in Python)**
- **Logical AND Operator (“&&” in C, C++ / “and” in Python)**
  - 0b10000101 && 0b10100000 = ?
  - 0b10000101 & 0b10000000 = ?

Bit Manipulation in Programming

- **Bitwise AND Operator (“&” in C, C++ / “&” in Python)**
- **Logical AND Operator (“&&” in C, C++ / “and” in Python)**
  - 0b10000101 && 0b10100000 = 1 (True, Boolean)
  - 0b10000101 & 0b10100000 = 0b10000000

Bit Manipulation in Programming

- **Bitwise OR Operator (“|” in C, C++ / “&” in Python)**
- **Logical OR Operator (“||” in C, C++ / “or” in Python)**
  - 0b10000101 || 0b10000000 = 1 (True, Boolean)
  - 0b10000101 | 0b10100101 = 0b10100101

Bit Manipulation in Programming

- **Bitwise NOT Operator (“~” in C, C++ / “~” in Python)**
- **Logical NOT Operator (“!” in C, C++ / “not” in Python)**
  - ~0b10000101 = 0b01111010
  - !0b10000101 = 0 (False, Boolean)

Bit Manipulation in Programming

- **Bitwise XOR Operator (“^” in C, C++ / “^” in Python)**
- **Logical XOR Operator (X)**
  - 0b1010 ^ 0b1100 = 0b0110

Bit Manipulation in Programming

- **Bit Shift operator <<, >>**
  - 0b11110000 >> 3 = 0b00011110
  - 0b00001111 << 2 = 0b00111100

Bit Manipulation in Programming

- **Bitwise Assignment**
  - Bitwise AND Assignment (&=)
  - Bitwise OR Assignment (|=)
  - Bitwise XOR Assignment (^=)

```
// Clear the 2nd bit of 'a'
a &= ~(1 << 2);

// Set the 3rd bit of 'a'
a |= (1 << 3);

// Toggle the 1st bit of 'a'
a ^= (1 << 1);
```

Bit Manipulation in Programming

- **Example 1)**

Python

```python
num = 5
if num & (num - 1) == 0:
    print("It is a power of 2.")
else:
    print("It is not a power of 2.")
```

C/C++

```c
int num = 5;
if (num & 1) {
    printf("This is a odd number.\n");
} else {
    printf("This is a even number.\n");
}
```

Bit Manipulation in Programming

- **Example 2)**

```
//Find lowest bit
int num = 12;      // 1100
int lowest_bit = num & -num; // 0100 (4)

//Remove lowest bit
int num = 12;      // 1100
int result = num & (num - 1); // 1000 (8)
```

Programming Practice

Graduate School of Data Science
Seoul National University

Programming Practice