



# ORGANIZACJA I ARCHITEKTURY KOMPUTERÓW

NOTATKI DO EGZAMINU 2025

Vurtzel Moran

# Wykład 1

## 1. Pojęcia Organizacji i Architektury Komputera

- **Architektura Komputera (Computer Architecture):**
  - **Definicja:** Określa funkcjonalne właściwości systemu komputerowego z perspektywy użytkownika.
  - **Przykład:** Zestaw instrukcji, formaty danych, sposób komunikacji między komponentami.
  - **Tłumaczenie:** Architektura to jak plan budynku – określa, jakie pomieszczenia są, jak są rozmieszczone i jakie mają funkcje.
- **Organizacja Komputera (Computer Organization):**
  - **Definicja:** Dotyczy sposobu implementacji architektury, czyli jak komponenty są fizycznie zbudowane i współpracują.
  - **Przykład:** Czy rozkaz mnożenia jest wykonywany przez specjalną jednostkę mnożącą czy przez wielokrotne użycie jednostki sumującej.
  - **Tłumaczenie:** Organizacja to sposób, w jaki plan budynku jest realizowany – jak zbudowane są ściany, instalacje, jakie materiały są użyte.
- **Zależność między Architekturą a Organizacją:**
  - Architektura definiuje, **co** komputer potrafi, a organizacja **jak** to jest realizowane.
  - **Tłumaczenie:** Architektura to projekt domu, a organizacja to rzeczywista budowa zgodnie z tym projektem.

## 2. Schemat Blokowy Komputera

- **CPU (Central Processing Unit):** Centralna jednostka przetwarzająca, odpowiedzialna za wykonywanie instrukcji.
- **Data Path:** Ścieżka danych między komponentami CPU, umożliwiająca przepływ informacji.
- **Bloki Funkcjonalne:**
  - **Rejestry:** Szybka pamięć wewnętrzna CPU.
  - **ALU (Arithmetic Logic Unit):** Jednostka arytmetyczno-logiczna, wykonuje operacje matematyczne i logiczne.
  - **Kontroler:** Zarządza przepływem danych i operacjami CPU.
- **Tłumaczenie:** Schemat blokowy to jak mapa drogową pokazująca główne części komputera i jak się ze sobą komunikują, aby wszystko działało sprawnie.

## 3. Systemy Liczbowe i Konwersje

- **Rodzaje Systemów Liczbowych:**

- **Dziesiętny (Decimal):** Podstawa 10, cyfry 0-9.
- **Dwójkowy (Binary):** Podstawa 2, cyfry 0-1.
- **Ósemkowy (Octal):** Podstawa 8, cyfry 0-7.
- **Szesnastkowy (Hexadecimal):** Podstawa 16, cyfry 0-9 oraz A-F.
- **Konwersje Między Systemami:**
  - **Dziesiętny ↔ Dwójkowy:**
    - **Całkowita część:** Dzielenie przez 2 i zbieranie reszt.
    - **Część ułamkowa:** Mnożenie przez 2 i zbieranie części całkowitych.
  - **Dwójkowy ↔ Ósemkowy/Szesnastkowy:**
    - Grupowanie bitów (3 dla ósemkowego, 4 dla szesnastkowego) i zamiana na odpowiadające cyfry.
- **Tłumaczenie:** Komputery używają różnych systemów liczbowych do przechowywania i przetwarzania danych. Konwersje między nimi są jak tłumaczenie między różnymi językami, aby komputer mógł efektywnie operować na danych.

#### 4. Reprezentacja Liczb Całkowitych

- **Kod Znak-Moduł:**
  - **Dodatnie:** Pierwszy bit (bit znaku) to 0.
  - **Ujemne:** Pierwszy bit to 1.
  - **Przykład:** 3 w kodzie znak-moduł to 0 011, a -3 to 1 011.
  - **Tłumaczenie:** Prosty sposób, gdzie pierwszy bit mówi, czy liczba jest dodatnia czy ujemna.
- **Kod Znak-Uzupełnienie do 1 (U1):**
  - **Definicja:** Negacja wszystkich bitów liczby dodatniej.
  - **Przykład:** Uzupełnienie do 1 liczby 1001 to 0110.
  - **Tłumaczenie:** Aby zapisać liczbę ujemną, odwracamy wszystkie bity liczby dodatniej.
- **Kod Znak-Uzupełnienie do 2 (U2):**
  - **Definicja:** Negacja bitów w kodzie U1 i dodanie 1.
  - **Przykład:** -3 w U2: 0011 → 1100 (negacja) + 0001 = 1101.
  - **Zalety:** Jedyna reprezentacja zera, co upraszcza operacje arytmetyczne.

- **Tłumaczenie:** Bardziej zaawansowana metoda, która pozwala na łatwiejsze wykonywanie działań matematycznych.

## 5. Kod ASCII

- **Definicja:** Standardowy system kodowania znaków alfanumerycznych.
- **Rodzaje:**
  - **7-bitowy:** 128 znaków.
  - **8-bitowy:** 256 znaków (możliwość dodania znaków narodowych).
- **Przykład:** Litera A ma kod binarny 0100 0001 i szesnastkowy 41.
- **Tłumaczenie:** ASCII to sposób, w jaki komputery rozumieją i reprezentują znaki tekstowe, takie jak litery i cyfry.

## 6. Operacje Arytmetyczne na Liczbach Binarnych

- **Dodawanie:**
  - Podobne do dodawania w systemie dziesiętnym, uwzględniając przeniesienia.
  - **Przykład:**  $1011 + 1101 = 11000$ .
- **Odejmowanie:**
  - Wykonywane poprzez użycie uzupełnień do 1 lub 2.
  - **Przykład:**  $205 - 18$  w systemie binarnym prowadzi do poprawnego wyniku 187.
- **Tłumaczenie:** Komputery potrafią dodawać i odejmować liczby binarne, używając specjalnych technik uwzględniających przeniesienia i pożyczki, aby zapewnić poprawność operacji.

---

## Podsumowanie

Pierwszy wykład wprowadza podstawowe pojęcia związane z organizacją i architekturą komputerów, wyjaśniając różnice między nimi oraz ich wzajemne zależności. Omówiono również schemat blokowy komputera, który pokazuje główne komponenty i ich interakcje. Kluczowym elementem były systemy liczbowe oraz metody konwersji między nimi, co jest fundamentem działania komputerów. Dodatkowo, przedstawiono różne sposoby reprezentacji liczb całkowitych oraz standard kodowania ASCII, który umożliwia komputerom pracę z tekstem. Na koniec omówiono podstawowe operacje arytmetyczne na liczbach binarnych, które są niezbędne do przetwarzania danych przez komputer.

Zrozumienie tych podstaw jest kluczowe dla dalszego studiowania bardziej zaawansowanych tematów w dziedzinie informatyki i inżynierii komputerowej.

## Wykład 2

### 1. Mnożenie Dwoch Liczb Binarnych

- **Proces Mnożenia:**

- **Krok 1:** Pisanie obu liczb binarnych jedna pod drugą.
- **Krok 2:** Mnożenie każdej cyfry dolnej liczby przez całą górną liczbę, przesuając wynik w lewo w zależności od pozycji cyfry.
- **Krok 3:** Sumowanie wszystkich przesuniętych wyników.

- **Przykład 1:**

**Mnożenie:**

```
1101 (13)10
x 1011 (11)10
-----
1101 (1101 × 1)
0000 (1101 × 0, przesunięte o 1 pozycję w lewo)
1101 (1101 × 1, przesunięte o 2 pozycje w lewo)
1101 (1101 × 1, przesunięte o 3 pozycje w lewo)
-----
10001111 (143)10
```

- **Obliczenia:**

- $(1101)_2 = 13_{10}$
- $(1011)_2 = 11_{10}$
- $13 \times 11 = 143_{10} = (10001111)_2$

- **Przykład 2:**

**Mnożenie:**

```
1111 (15)10
x 1101 (13)10
-----
1111 (1111 × 1)
1111 (1111 × 0, przesunięte o 1 pozycję w lewo)
```

1111 (1111  $\times$  1, przesunięte o 2 pozycje w lewo)

1111 (1111  $\times$  1, przesunięte o 3 pozycje w lewo)

-----

11000011 (195)<sub>10</sub>

- **Obliczenia:**

- $(1111)_2 = 15_{10}$
- $(1101)_2 = 13_{10}$
- $15 \times 13 = 195_{10} = (11000011)_2$

## 2. Dzielenie Liczb Binarnych

- **Proces Dzielenia:**

- **Krok 1:** Ustawienie dzielnej i dzielnika.
- **Krok 2:** Porównanie dzielnika z odpowiednią częścią dzielnej.
- **Krok 3:** Jeśli dzielnik pasuje, zapisanie 1 w części wyniku i odjęcie dzielnika od tej części dzielnej.
- **Krok 4:** Przesunięcie się do następnej cyfry dzielnej i powtórzenie procesu.
- **Krok 5:** Kontynuowanie aż do przetworzenia wszystkich cyfr dzielnej.

- **Przykład 1:**

**Dzielenie:**

1101  $\div$  1011

\_\_\_\_\_

- **Obliczenia:**

- $(1101)_2 = 13_{10}$
- $(1011)_2 = 11_{10}$
- $13 \div 11 = 1$  z resztą 2
- Wynik: 1 (reszta: 0010)

- **Przykład 2:**

**Dzielenie:**

10010001  $\div$  1011

\_\_\_\_\_

$$10010001 \div 1011 = 13 \text{ z resztą } 2$$

$$(10010001)_2 = 145_{10}$$

$$(1011)_2 = 11_{10}$$

$$145 \div 11 = 13 + 2$$

- **Obliczenia:**

- $(10010001)_2 = 145_{10}$
- $(1011)_2 = 11_{10}$
- $145 \div 11 = 13 \text{ z resztą } 2$
- Wynik:  $(13)_{10} + (2)_{10}$

### 3. Odejmowanie Dwóch Liczb n-bitowych

- **Metoda Uzupełnień:**

- **Krok 1:** Do odjemnej M dodajemy uzupełnienie do p odjemnika N.
  - **Formuła:**  $M + (p^n - N) = M - N + p^n$
- **Krok 2:** Jeśli  $M \geq N$ , od wyniku odejmujemy  $p^n$ , aby uzyskać poprawny wynik  $M - N$ .
- **Przypadki:**
  - **M > N:** Otrzymujemy dodatnią różnicę.
  - **M = N:** Różnica wynosi 0.
  - **M < N:** Różnica jest ujemna, reprezentowana jako uzupełnienie do p różnicy (N - M).

- **Przykład 1: M > N**

**Dane:**

$$M = (1987)_{10}$$

$$N = (1958)_{10}$$

$$p = 10, n = 4$$

- **Obliczenia:**

- Uzupełnienie do p:  $10^4 = 10000$
- $p^n - N = 10000 - 1958 = 8042$
- $M + (p^n - N) = 1987 + 8042 = 10029$
- Ponieważ  $M > N$ , odejmujemy  $p^n$ :  $10029 - 10000 = 29$

- **Wynik:**  $29_{10}$

- **Przykład 2:  $M = N$**

**Dane:**

$$M = (1958)_{10}$$

$$N = (1958)_{10}$$

$$p = 10, n = 4$$

- **Obliczenia:**

- $M + (p^n - N) = 1958 + (10000 - 1958) = 1958 + 8042 = 10000$
- Ponieważ  $M = N$ , odejmujemy  $p^n$ :  $10000 - 10000 = 0$
- **Wynik:**  $0_{10}$

- **Przykład 3:  $M < N$**

**Dane:**

$$M = (1958)_{10}$$

$$N = (1987)_{10}$$

$$p = 10, n = 4$$

- **Obliczenia:**

- Uzupełnienie do  $p$ :  $10^4 = 10000$
- $p^n - N = 10000 - 1987 = 8013$
- $M + (p^n - N) = 1958 + 8013 = 9971$
- Ponieważ  $M < N$ , wynik jest ujemny:  $-(N - M) = -(1987 - 1958) = -29$
- **Wynik:**  $-29_{10}$

- **Przykład 4: Odejmowanie w Systemie Dwójkowym ( $p = 2$ )**

**Dane:**

$$M = (11001)_2 = 25_{10}$$

$$N = (1010)_2 = 10_{10}$$

$$p = 2, n = 5$$

- **Obliczenia:**

- Uzupełnienie do  $p$ :  $2^5 = 32$
- $p^n - N = 32 - 10 = 22$  (w binarnym:  $10110$ )



- $M + (p^n - N) = 25 + 22 = 47$  (w binarnym: 101111)
- Ponieważ  $M > N$ , odejmujemy  $p^n$ :  $47 - 32 = 15$  (w binarnym: 01111)
- **Wynik:**  $15_{10} = (01111)_2$

- **Przykład 5: Odejmowanie Liczb Równości**

**Dane:**

$$M = (1010)_2 = 10_{10}$$

$$N = (1010)_2 = 10_{10}$$

$$p = 2, n = 5$$

- **Obliczenia:**

- $M + (p^n - N) = 10 + (32 - 10) = 10 + 22 = 32$
- Ponieważ  $M = N$ , odejmujemy  $p^n$ :  $32 - 32 = 0$
- **Wynik:**  $0_{10}$

#### 4. Instalacja Multisim

- **Instrukcje:**

- Zainstaluj studencką wersję Multisim-a, narzędzia do symulacji obwodów elektronicznych.
- **Link do pobrania:** [Multisim \(National Instruments\)](#)

#### Mnożenie Dwoch Liczb Binarnych

Mnożenie liczb binarnych działa podobnie do mnożenia liczb dziesiętnych. Każda cyfra dolnej liczby (mnożna) jest mnożona przez całą górną liczbę, a wynik jest przesuwany w lewo w zależności od pozycji tej cyfry. Następnie wszystkie przesunięte wyniki są sumowane, aby uzyskać końcowy rezultat.

**Przykład:** Mnożenie  $1101 (13)_2$  przez  $1011 (11)_2$  daje wynik  $10001111 (143)_2$ . Proces obejmuje mnożenie każdej cyfry dolnej liczby przez górną i przesuwanie wyników w lewo przed ich dodaniem.

#### Dzielenie Liczb Binarnych

Dzielenie binarne jest procesem podobnym do dzielenia w systemie dziesiętnym. Polega na porównaniu dzielnika z odpowiednią częścią dzielnej, zapisaniu 1 lub 0 w wyniku w zależności od tego, czy dzielnik "mieści się" w tej części, a następnie odjęciu dzielnika i przesunięciu się do następnej cyfry dzielnej.

**Przykład:** Dzielenie  $1101 (13)_2$  przez  $1011 (11)_2$  daje wynik 1 z resztą 2. Oznacza to, że 11 mieści się raz w 13, pozostawiając resztę 2.

#### Odejmowanie Dwoch Liczb n-bitowych

Odejmowanie w systemie binarnym można przeprowadzić za pomocą metod uzupełnień, które upraszczają proces odjęcia przez zamianę na dodawanie. Metoda ta wykorzystuje uzupełnienie do  $p$  (gdzie  $p$  to podstawa systemu liczbowego) od liczby, którą odejmujemy, a następnie dodaje ją do liczby odjemnej.

- **Jeżeli  $M \geq N$ :** Wynik odejmowania jest dodatni i można go bezpośrednio odczytać po wykonaniu operacji.
- **Jeżeli  $M < N$ :** Wynik jest ujemny i reprezentowany jako uzupełnienie do  $p$  różnicy  $(N - M)$ .

**Przykład:** Odejmowanie 1958 od 1987 w systemie dziesiętnym daje różnicę 29. Jeśli  $M$  jest mniejsze od  $N$ , wynik będzie ujemny, na przykład  $1958 - 1987 = -29$ .

---

## Podsumowanie

Drugi wykład kontynuował zagadnienia związane z operacjami arytmetycznymi w systemie binarnym, wprowadzając procesy mnożenia, dzielenia i odejmowania liczb binarnych. Przedstawiono szczegółowe przykłady, które ilustrują, jak te operacje są wykonywane krok po kroku, zarówno w systemie dziesiętnym, jak i binarnym. Metoda uzupełnień do  $p$  została wyjaśniona jako efektywny sposób na przeprowadzanie odejmowania, co jest kluczowe dla implementacji operacji matematycznych w systemach komputerowych. Dodatkowo, wspomniano o narzędziu Multisim, które jest użyteczne w praktycznym projektowaniu obwodów elektronicznych.

Zrozumienie tych operacji jest fundamentem dla dalszego zgłębiania tematów związanych z przetwarzaniem danych, algorytmami oraz projektowaniem sprzętu komputerowego.

## Wykład 3

### 1. Nadmiar (Przepełnienie) w Dodawaniu Liczb Binarnych

- **Definicja Nadmiaru:**
  - **Nadmiar (Overflow):** Występuje, gdy wynik dodawania dwóch  $n$ -bitowych liczb przekracza zakres reprezentacji tych liczb, co powoduje konieczność użycia dodatkowego bitu.
- **Nadmiar w Liczbach Bez Znaków:**
  - **Przykład:**
    - **Format danych:** 8-bitowe liczby bez znaku (0 do 255).
    - **Dodawanie:**  $150 + 190 = 340$ .
      - **W systemie binarnym:**
      - $150_{(10)} = 10010110$
      - $190_{(10)} = 10111110$
      - -----

- Suma = 1 01010100 (9 bitów)
- **Interpretacja:** Suma 340 przekracza maksymalny zakres 8-bitowych liczb bez znaku (255). Bit przeniesienia (najbardziej znaczący bit) jest nadmiarowy i zostaje pominięty, co daje błędny wynik 84.
- **Nadmiar w Liczbach Ze Znakiem:**
  - **Format danych:** 8-bitowe liczby ze znakiem (zakres: -128 do +127), gdzie najbardziej znaczący bit jest bitem znaku (0 dla dodatnich, 1 dla ujemnych).
  - **Przykład 1: Dodawanie Liczb Dodatnich:**
    - **Dane:**
    - 50 (10) = 00110010
    - 90 (10) = 01011010
    - -----
    - Suma = 10001100 (-116 w zapisie znak-uzupełnienie do 2)
    - **Interpretacja:** 50 + 90 = 140, co przekracza maksymalny zakres +127. Wynik jest błędnie interpretowany jako liczba ujemna (-116).
  - **Przykład 2: Dodawanie Liczb Ujemnych:**
    - **Dane:**
    - -50 (10) = 11001110
    - -90 (10) = 10100110
    - -----
    - Suma = 10110100 (-76 w zapisie znak-uzupełnienie do 2)
    - **Interpretacja:** -50 + (-90) = -140, co przekracza minimalny zakres -128. Wynik jest błędnie interpretowany jako liczba dodatnia (-76).

## 2. Algebra Boole'a

- **Definicja:**
  - **Algebra Boole'a:** Gałąź matematyki zajmująca się operacjami logicznymi, wykorzystywana w projektowaniu i analizie układów cyfrowych. Zmienne boolowskie mogą przyjmować tylko dwie wartości: 0 lub 1.
- **Podstawowe Operacje:**
  - **AND (Iloczyn logiczny):**  $X \cdot Y = XY$
  - **OR (Suma logiczna):**  $X + Y$

- **NOT (Negacja):**  $\neg X$  lub  $X'$

- **Tablice Prawdy:**

- **AND:**

X	Y	$X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1

- **OR:**

X	Y	$X + Y$
0	0	0
0	1	1
1	0	1
1	1	1

- **NOT:**

X	$\neg X$
0	1
1	0

- **Zasada Dualności:**

- **Definicja:** Zmiana wszystkich operacji logicznych na ich dualne oraz zamiana 0 z 1 i odwrotnie.
- **Przykład:** Dualność między AND a OR: Jeśli mamy wyrażenie  $X + Y$ , jego dualne to  $X \cdot Y$ .

- **Prawa de Morgana:**

- **Pierwsze Prawo de Morgana:**  $\neg(X \cdot Y) = \neg X + \neg Y$
- **Drugie Prawo de Morgana:**  $\neg(X + Y) = \neg X \cdot \neg Y$

- **Podstawowe Tożsamości Algebry Boole'a:**

- **Przemienność:**

- $X + Y = Y + X$
- $X \cdot Y = Y \cdot X$

- **Łączność:**
  - $(X + Y) + Z = X + (Y + Z)$
  - $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$
- **Rozdzielność:**
  - $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$
  - $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$
- **Przykład Upraszczania Wyrażenia:**
  - **Wyrażenie:**  $(A + B)(A + CD)$
  - **Kroki Upraszczania:**
  - $(A + B)(A + CD) = AA + ACD + BA + BCD = A + ACD + BA + BCD$
  - $= A(1 + CD + B) + BCD = A + BCD$
  - **Ostateczny Wynik:**  $A + BCD$

### 3. Bramki Logiczne

- **Podstawowe Bramki:**
  - **AND:** Wydaje 1 tylko wtedy, gdy oba wejścia są 1.
  - **OR:** Wydaje 1, gdy przynajmniej jedno wejście jest 1.
  - **NOT:** Inwertuje wartość wejścia ( $0 \rightarrow 1, 1 \rightarrow 0$ ).
- **Zaawansowane Bramki:**
  - **NAND (NOT AND):** Wynik negacji operacji AND. Wydaje 1 zawsze, chyba że oba wejścia są 1.
  - **NOR (NOT OR):** Wynik negacji operacji OR. Wydaje 1 tylko wtedy, gdy oba wejścia są 0.
  - **XOR (Exclusive OR):** Wydaje 1, gdy liczba wejść 1 jest nieparzysta.
- **Zbiór Funkcjonalnie Pełny:**
  - **Definicja:** Zbiór bramek, za pomocą których można skonstruować dowolną funkcję logiczną.
  - **Przykłady:**
    - $\{AND, OR, NOT\}$  jest zbiór funkcjonalnie pełny.
    - $\{NAND\}$  jest zbiór funkcjonalnie pełny.
    - $\{NOR\}$  jest zbiór funkcjonalnie pełny.

- **Przykłady Implementacji Bramki:**

- **Bramka NAND:**

- Można skonstruować bramki AND, OR, i NOT używając tylko bramek NAND.

- **Bramka NOR:**

- Podobnie, bramki AND, OR, i NOT można zbudować z bramek NOR.

- **Przykład Symulacji Bramki Logiczej w Multisim:**

- **Krok 1:** Umieszczenie odpowiednich bramek (np. NOT, AND) na schemacie.
  - **Krok 2:** Podłączenie wejść i zasilania (np. 5V).
  - **Krok 3:** Obserwacja wyjścia na oscyloskopie lub mierniku.

#### 4. Operacje Arytmetyczne na Liczbach Binarnych - Odejmowanie

- **Metoda Uzupełnień:**

- **Krok 1:** Do odjemnej M dodajemy uzupełnienie do p odjemnika N.
    - **Formuła:**  $M + (p^n - N) = M - N + p^n$
  - **Krok 2:** Jeśli  $M \geq N$ , odejmujemy  $p^n$  od wyniku, aby uzyskać poprawny wynik  $M - N$ .

- **Przypadki:**

- **M > N:** Wynik jest dodatni.
  - **M = N:** Wynik to 0.
  - **M < N:** Wynik jest ujemny, reprezentowany jako uzupełnienie do p różnicy (N - M).

- **Przykłady:**

- **Przykład 1: M > N**

- **Dane:**

- $M = 1987_{10}$

- $N = 1958_{10}$

- $p = 10, n = 4$

- **Obliczenia:**

- $p^n = 10^4 = 10000$

- $p^n - N = 10000 - 1958 = 8042$

- $M + (p^n - N) = 1987 + 8042 = 10029$

- Ponieważ  $M > N$ , odejmujemy  $p^n$ :  $10029 - 10000 = 29$

- **Wynik:**  $29_{10}$

○ **Przykład 2:  $M = N$**

- **Dane:**

- $M = 1958_{10}$

- $N = 1958_{10}$

- $p = 10, n = 4$

- **Obliczenia:**

- $M + (p^n - N) = 1958 + (10000 - 1958) = 1958 + 8042 = 10000$

- Ponieważ  $M = N$ , odejmujemy  $p^n$ :  $10000 - 10000 = 0$

- **Wynik:**  $0_{10}$

○ **Przykład 3:  $M < N$**

- **Dane:**

- $M = 1958_{10}$

- $N = 1987_{10}$

- $p = 10, n = 4$

- **Obliczenia:**

- $p^n = 10^4 = 10000$

- $p^n - N = 10000 - 1987 = 8013$

- $M + (p^n - N) = 1958 + 8013 = 9971$

- Ponieważ  $M < N$ , wynik jest ujemny:  $-(N - M) = -(1987 - 1958) = -29$

- **Wynik:**  $-29_{10}$

○ **Przykład 4: Odejmowanie w Systemie Dwójkowym ( $p = 2$ )**

- **Dane:**

- $M = 11001_2 = 25_{10}$

- $N = 1010_2 = 10_{10}$

- $p = 2, n = 5$

- **Obliczenia:**

- $p^n = 2^5 = 32$

- $p^n - N = 32 - 10 = 22$  (w binarnym:  $10110_2$ )

- $M + (p^n - N) = 25 + 22 = 47$  (w binarnym:  $101111_2$ )
- Ponieważ  $M > N$ , odejmujemy  $p^n$ :  $47 - 32 = 15$  (w binarnym:  $01111_2$ )
- **Wynik:**  $15_{10} = 01111_2$

○ **Przykład 5: Odejmowanie Liczb Równości**

- **Dane:**
- $M = 1010_2 = 10_{10}$
- $N = 1010_2 = 10_{10}$
- $p = 2, n = 5$
- **Obliczenia:**
  - $M + (p^n - N) = 10 + (32 - 10) = 10 + 22 = 32$
  - Ponieważ  $M = N$ , odejmujemy  $p^n$ :  $32 - 32 = 0$
  - **Wynik:**  $0_{10}$



## Dwuelementowa algebra Boole'a.

Dwuelementowa algebra Boole'a jest działem matematyki wykorzystywanym do projektowania i analizy układów cyfrowych. Zmienne boolowskie (logiczne) mogą przyjmować wartości tylko ze zbioru  $\{0, 1\}$ . W algebrze Boole'a do przedstawienia iloczynu logicznego (AND) zmiennych  $X$  i  $Y$  będziemy stosować zapis  $X \cdot Y = XY$ . Do przedstawienia sumy logicznej (OR) zmiennych  $X$  i  $Y$  będziemy stosować zapis  $X + Y$ . Negację zmiennej  $X$  będziemy zapisywać jako  $\bar{X}$  lub  $X'$ .

Poniżej są przedstawione (za pomocą tablicy prawdy) definicje następujących funkcji logicznych: AND, OR i NOT.

AND			OR			NOT	
$X$	$Y$	$X \cdot Y$	$X$	$Y$	$X + Y$	$X$	$\bar{X}$
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

09.03.2021

12

1. $x + 0 = x$	2. $x \cdot 1 = x$
3. $x + 1 = 1$	4. $x \cdot 0 = 0$
5. $x + x = x$	6. $x \cdot x = x$
7. $x + \bar{x} = 1$	8. $x \cdot \bar{x} = 0$
9. $\bar{\bar{x}} = x$	
10. $x + y = y + x$	11. $x \cdot y = y \cdot x$
12. $x + (y + z) = (x + y) + z$	13. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
14. $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	15. $x + (y \cdot z) = (x + y) \cdot (x + z)$
16. $\overline{x + y} = \bar{x} \cdot \bar{y}$	17. $\overline{x \cdot y} = \bar{x} + \bar{y}$

*The duality principle.*

1. $x + 0 = x$	2. $x \cdot 1 = x$
3. $x + 1 = 1$	4. $x \cdot 0 = 0$
5. $x + x = x$	6. $x \cdot x = x$
7. $x + \bar{x} = 1$	8. $x \cdot \bar{x} = 0$
9. $\bar{\bar{x}} = x$	
10. $x + y = y + x$	11. $x \cdot y = y \cdot x$
12. $x + (y + z) = (x + y) + z$	13. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
14. $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	15. $x + (y \cdot z) = (x + y) \cdot (x + z)$
16. $\overline{x + y} = \bar{x} \cdot \bar{y}$	17. $\overline{x \cdot y} = \bar{x} + \bar{y}$

**Prawa de Morgana** (pozycje:16 i 17 w tablicy powyżej).

$$\overline{x_1 + x_2 + \dots + x_n} = \bar{x}_1 \cdot \bar{x}_2 \cdot \dots \cdot \bar{x}_n$$

$$\overline{x_1 \cdot x_2 \cdot \dots \cdot x_n} = \bar{x}_1 + \bar{x}_2 + \dots + \bar{x}_n$$

TABLE 2-6 Basic Identities of Boolean Algebra		Podstawowe tożsamości algebry boolowskiej	
1. $X + 0 = X$	2. $X \cdot 1 = X$		
3. $X + 1 = 1$	4. $X \cdot 0 = 0$		
5. $X + X = X$	6. $X \cdot X = X$		
7. $X + \bar{X} = 1$	8. $X \cdot \bar{X} = 0$		
9. $\bar{\bar{X}} = X$			
10. $X + Y = Y + X$	11. $XY = YX$	Commutative	Przemienność
12. $X + (Y + Z) = (X + Y) + Z$	13. $X(YZ) = (XY)Z$	Associative	Łączność
14. $X(Y + Z) = XY + XZ$	15. $X + YZ = (X + Y)(X + Z)$	Distributive	Rozdzielność
16. $\overline{X + Y} = \bar{X} \cdot \bar{Y}$	17. $\overline{X \cdot Y} = \bar{X} + \bar{Y}$	DeMorgan's	Prawa de Morgana

### 5. Podsumowanie

W trzecim wykładzie omówiono zagadnienia związane z nadmiarem (przepełnieniem) w operacjach arytmetycznych na liczbach binarnych zarówno bez znaku, jak i ze znakiem. Przedstawiono, jak nadmiar wpływa na wyniki dodawania, a także jak jest wykrywany w różnych systemach liczbowych.

Dodatkowo, wprowadzono podstawy algebry Boole’a, kluczowego narzędzia w projektowaniu układów cyfrowych. Przedstawiono podstawowe operacje logiczne, tablice prawdy, zasady dualności oraz prawa de Morgana, które są fundamentem dla upraszczania wyrażeń logicznych.

Omówiono również bramki logiczne, zarówno podstawowe (AND, OR, NOT), jak i zaawansowane (NAND, NOR, XOR), oraz ich rolę w budowaniu funkcjonalnie pełnych systemów logicznych. Zwrócono uwagę na możliwość tworzenia dowolnych funkcji logicznych za pomocą zbiorów bramek funkcjonalnie pełnych.

### Tłumaczenie po Ludzku

#### Nadmiar (Przepełnienie) w Dodawaniu Liczb Binarnych

**Co to jest nadmiar?** Nadmiar, czyli przepełnienie, występuje, gdy dodajemy dwie liczby binarne i wynik jest większy niż to, co możemy zapisać za pomocą dostępnych bitów. To tak, jakbyśmy próbowali zapisać większą liczbę w małym pudełku – część liczby "wypada" poza pudełko.

**Przykład bez znaku:** Wyobraź sobie, że masz pudełko na liczby od 0 do 255 (8 bitów). Dodajesz do siebie 150 i 190, co daje 340. Niestety, 340 nie mieści się w tym pudełku, więc część liczby jest utracona, a wynik jest błędny.

**Przykład ze znakiem:** Kiedy dodajesz dwie liczby dodatnie, jak 50 i 90, wynik to 140. Jednak w 8-bitowym systemie ze znakiem możemy zapisać tylko liczby od -128 do +127. 140 jest poza tym zakresem, więc komputer błędnie interpretuje wynik jako liczbę ujemną. Podobnie, dodając dwie liczby ujemne, np. -50 i -90, otrzymujemy -140, co również jest poza zakresem i komputer pokazuje błędny wynik jako liczbę dodatnią.

## Algebra Boole'a

**Czym jest algebra Boole'a?** To dział matematyki, który pozwala nam pracować z wartościami logicznymi – prawda (1) i fałsz (0). Jest fundamentem dla projektowania obwodów cyfrowych, takich jak komputery.

### Podstawowe operacje:

- **AND (Iloczyn):** Wynik jest prawdziwy tylko wtedy, gdy oba wejścia są prawdziwe.
- **OR (Suma):** Wynik jest prawdziwy, jeśli przynajmniej jedno z wejść jest prawdziwe.
- **NOT (Negacja):** Odwraca wartość – prawda staje się fałszem i odwrotnie.

### Prawa i zasady:

- **Przemienność:** Kolejność wejść nie ma znaczenia ( $X + Y = Y + X$ ).
- **Łączność:** Grupowanie wejść nie wpływa na wynik ( $(X + Y) + Z = X + (Y + Z)$ ).
- **Rozdzielność:** Pozwala rozdzielać operacje AND i OR ( $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$ ).

**Przykład upraszczania wyrażenia:** Wyrażenie  $(A + B)(A + CD)$  można uprościć do  $A + BCD$ . Dzięki temu wyrażenie jest prostsze i łatwiejsze do realizacji w obwodzie logicznym.

## Bramki Logiczne

**Co to są bramki logiczne?** To podstawowe elementy budulcowe układów cyfrowych. Działają na zasadzie algebry Boole'a, przetwarzając sygnały wejściowe (0 lub 1) i generując sygnał wyjściowy.

### Rodzaje bramek:

- **AND, OR, NOT:** Podstawowe bramki wykonujące podstawowe operacje logiczne.
- **NAND, NOR, XOR:** Zaawansowane bramki, które mogą zastąpić podstawowe bramki w różnych zastosowaniach.

**Zbiór funkcjonalnie pełny:** Oznacza, że za pomocą tych bramek można zbudować dowolny układ logiczny. Na przykład, bramka NAND sama w sobie jest funkcjonalnie pełna, co oznacza, że można zbudować za jej pomocą wszystkie inne bramki logiczne.

**Przykład użycia w Multisim:** Korzystając z Multisim, możemy symulować działanie bramek logicznych, tworzyć różne kombinacje i testować ich zachowanie bez konieczności fizycznego budowania układów.

### Operacje Arytmetyczne na Liczbach Binarnych - Odejmowanie

**Jak odejmować liczby binarne?** Odejmowanie można przeprowadzić poprzez dodanie uzupełnienia do liczby, którą odejmujemy. To pozwala na wykorzystanie mechanizmów dodawania do wykonania operacji odejmowania.

**Przykład:** Chcemy odjąć 10 od 25 w systemie binarnym:

- 25 (10) to  $11001_2$
- 10 (10) to  $1010_2$
- $2^5 = 32$
- $32 - 10 = 22$  ( $10110_2$ )
- Dodajemy  $25 + 22 = 47$  ( $101111_2$ )
- Ponieważ  $25 > 10$ , odejmujemy 32:  $47 - 32 = 15$  ( $01111_2$ )
- **Wynik:**  $15$  (10) =  $01111_2$

---

## Wykład 4

### 1. Podstawowe Bramki Logiczne

#### NAND (Negative-OR)

- **Równanie:**  $\overline{X+Y}$
- **Opis:** Bramki NAND realizują operację logicznego OR, a następnie negację wyniku. Innymi słowy, wyjście bramki NAND jest **falszywe** tylko wtedy, gdy **wszystkie** jej wejścia są **prawdziwe**.
- **Zastosowanie:** Bramki NAND są uniwersalne, co oznacza, że można je wykorzystać do stworzenia dowolnej innej bramki logicznej.
- **Tabela Prawdy:**

X	Y	$\overline{X+Y}$
0	0	1
0	1	0
1	0	0
1	1	0

#### NOR (Negative-AND)

- **Równanie:**  $\overline{X \cdot Y}$
- **Opis:** Bramki NOR realizują operację logicznego AND, a następnie negację wyniku. Wyjście bramki NOR jest **prawdziwe** tylko wtedy, gdy **wszystkie** jej wejścia są **falszywe**.
- **Zastosowanie:** Podobnie jak bramki NAND, bramki NOR są uniwersalne i mogą być używane do tworzenia innych bramek logicznych.
- **Tabela Prawdy:**

X	Y	$\overline{X \cdot Y}$
0	0	1
0	1	1
1	0	1
1	1	0

$$\overline{XY} = \overline{X} + \overline{Y}$$

INPUTS		OUTPUT	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

$$\overline{X + Y} = \overline{X} \cdot \overline{Y}$$

INPUTS		OUTPUT	
X	Y	$\overline{X + Y}$	$\overline{X} \cdot \overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

## 2. Optymalizacja Układów Logicznych

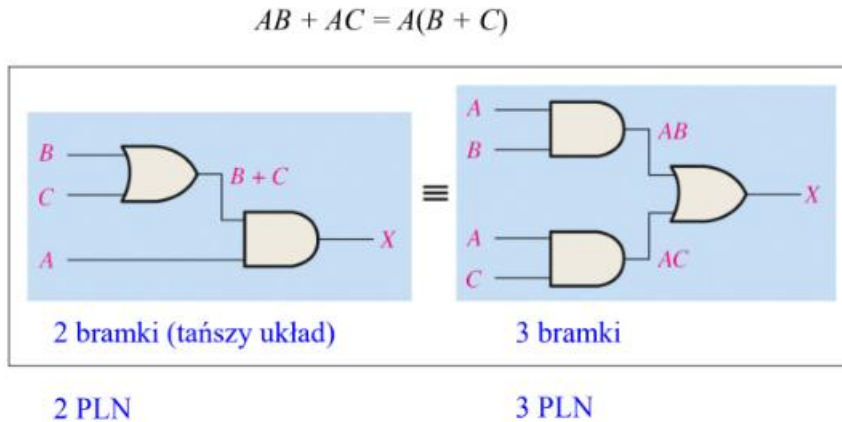
### Koszt i Oszczędności

- **Założenie:** Koszt jednej bramki logicznej to **1 PLN**.
- **Przykład Oszczędności:**
  - **Bez Optymalizacji:** Użycie **3 bramek** → Koszt: **3 PLN**
  - **Z Optymalizacją:** Użycie **2 bramek** → Koszt: **2 PLN**
  - **Oszczędność:** **1 PLN** na układ  $\times$  liczba produkowanych układów
    - **Przykład:**  $1 \text{ PLN} \times 10^6 = 10^6 \text{ PLN}$  oszczędności przy produkcji miliona układów.

### Reguły Redukcji

- **Prawo Dystrybucji:**

- Przykład:  $AB + AC = A(B + C)$
- **Opis:** Zastosowanie prawa dystrybucji pozwala na zmniejszenie liczby bramek potrzebnych do implementacji wyrażenia logicznego, co przekłada się na niższe koszty produkcji.



### 3. Rodzaje Bram i Implementacje

#### Bramki 3-input NOR i NAND

- **Opis:** Bramki z trzema wejściami mogą realizować bardziej złożone funkcje logiczne w porównaniu do bramek dwuwejściowych.

#### Typy Implementacji:

- **(a) Użycie bramek 2-input:**
  - **Opis:** Realizacja funkcji 3-input przy użyciu bramek 2-input wymaga kaskadowania kilku bramek, co może prowadzić do zwiększonego opóźnienia sygnału i wyższego zużycia energii.
  - **Zalety:** Większa elastyczność w projektowaniu.
  - **Wady:** Większa liczba bramek → Wyższy koszt.
- **(b) Użycie bramek 3-input:**
  - **Opis:** Bezpośrednia implementacja funkcji 3-input za pomocą bramek 3-input jest bardziej efektywna pod względem liczby bramek i wydajności.
  - **Zalety:** Mniejsza liczba bramek → Niższy koszt, mniejsze opóźnienia.
  - **Wady:** Ograniczona elastyczność w porównaniu do bramek 2-input.
- **(c) Bramki Kaskadowe NAND:**
  - **Opis:** Łączenie kilku bramek NAND w szereg (kaskadowo) pozwala na realizację bardziej złożonych funkcji logicznych przy zachowaniu optymalizacji kosztów.
  - **Zalety:** Możliwość tworzenia złożonych funkcji przy użyciu minimalnej liczby bramek.

- **Wady:** Może wymagać dokładniejszego zarządzania opóźnieniami sygnałów.

#### **Bramki Kaskadowe**

- **Opis:** Kaskadowe połączenie bramek oznacza, że wyjście jednej bramki jest podłączone jako wejście do kolejnej bramki. Pozwala to na tworzenie bardziej złożonych funkcji logicznych z mniejszą liczbą bramek podstawowych.

#### **Porównanie Implementacji z użyciem bramek 2-input i 3-input**

- **2-input:**
    - **Zalety:** Elastyczność w projektowaniu i możliwość tworzenia różnych kombinacji logicznych.
    - **Wady:** Większa liczba bramek potrzebnych do realizacji funkcji 3-input, co zwiększa koszt i opóźnienia.
  - **3-input:**
    - **Zalety:** Mniejsza liczba bramek potrzebnych do realizacji funkcji 3-input, co obniża koszt i zmniejsza opóźnienia.
    - **Wady:** Może być mniej elastyczne w pewnych projektach, zwłaszcza tam, gdzie potrzebna jest większa kombinatoryka logiczna.
- 

## **4. Języki Opisujące Sprzęt (HDL)**

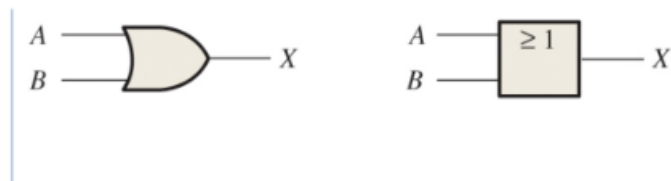
### **VHDL (VHSIC Hardware Description Language)**

- **Opis:** VHDL to język wysokiego poziomu służący do modelowania, symulacji oraz syntezy układów cyfrowych. Umożliwia opis struktury i zachowania układów cyfrowych na różnych poziomach abstrakcji.
- **Zastosowanie:** Definiowanie bramek logicznych, struktur układów, opis architektury.

### **Verilog**

- **Opis:** Verilog to alternatywny język HDL, często używany w przemyśle do projektowania układów cyfrowych. Jest mniej złożony składniowo w porównaniu do VHDL.
- **Porównanie z VHDL:**
  - **Verilog:**
    - Mniej złożona składnia.
    - Popularny w Stanach Zjednoczonych.
    - Często używany w projektach wymagających szybkiego prototypowania.
  - **VHDL:**
    - Bardziej rygorystyczny i silnie typowany.
    - Popularny w Europie.
    - Lepszy do tworzenia dokumentacji i bardziej złożonych projektów.

- VHDL
- Verilog



## 5. Iloczyn Pełny (Minterm)

TABLE 2-9  
Minterms for Three Variables

X	Y	Z	Product Term	Symbol	$m_0$	$m_1$	$m_2$	$m_3$	$m_4$	$m_5$	$m_6$	$m_7$
0	0	0	$\bar{X}\bar{Y}\bar{Z}$	$m_0$	1	0	0	0	0	0	0	0
0	0	1	$\bar{X}\bar{Y}Z$	$m_1$	0	1	0	0	0	0	0	0
0	1	0	$\bar{X}Y\bar{Z}$	$m_2$	0	0	1	0	0	0	0	0
0	1	1	$\bar{X}YZ$	$m_3$	0	0	0	1	0	0	0	0
1	0	0	$X\bar{Y}\bar{Z}$	$m_4$	0	0	0	0	1	0	0	0
1	0	1	$X\bar{Y}Z$	$m_5$	0	0	0	0	0	1	0	0
1	1	0	$XY\bar{Z}$	$m_6$	0	0	0	0	0	0	1	0
1	1	1	$XYZ$	$m_7$	0	0	0	0	0	0	0	1

Definicja:

- Iloczyn pełny n zmiennych to taki produkt logiczny zmiennych lub ich negacji, w którym każda zmienna (lub jej negacja) występuje dokładnie raz. Każdy minterm odpowiada jednej unikalnej kombinacji wartości wejściowych.

Przykład:

- Dla trzech zmiennych x, y, z, mintermy to:

- $m_0 = \bar{x} \cdot \bar{y} \cdot \bar{z}$

- $m_1 = \bar{x} \cdot \bar{y} \cdot z$

- $m_2 = \bar{x} \cdot y \cdot \bar{z}$

- $m_3 = \bar{x} \cdot y \cdot z$

- $m_4 = x \cdot \bar{y} \cdot \bar{z}$

- $m_5 = x \cdot \bar{y} \cdot z$

- $m_6 = x \cdot y \cdot \bar{z}$

- $m_7 = x \cdot y \cdot z$



### Znaczenie:

- Mintermy są kluczowe w projektowaniu i analizie funkcji logicznych, pozwalają na reprezentację każdej możliwej kombinacji wejść. Umożliwiają tworzenie wyrażeń logicznych w formie kanonicznej, co jest podstawą dla metod takich jak Karnaugh Maps czy Quine-McCluskey.
- 

## 6. Przykłady Kodów VHDL

### Przykład 1: Prosta Bramka OR

```
entity E1 is
  port(
    A, B : in bit;  -- Wejścia
    X     : out bit  -- Wyjście
  );
end E1;

architecture A1 of E1 is
begin
  X <= A or B;  -- Implementacja bramki OR
end A1;
```

- **Opis:**
  - **entity E1:** Definiuje nazwę modułu oraz jego porty (wejścia i wyjścia).
  - **architecture A1:** Opisuje zachowanie bramki OR, gdzie wyjście X jest wynikiem operacji OR na wejściach A i B.

### Przykład 2: Bramki Logiczne

- **OR\_2 (2-input OR Gate):**
- 

```
entity OR_2 is
  port(
    A, B : in bit;  -- Wejścia
    X     : out bit  -- Wyjście
  );
end OR_2;

architecture A1 of OR_2 is
begin
  X <= A or B;  -- Implementacja bramki OR
end A1;
```

- **AND\_2 (2-input AND Gate):**

```

entity AND_2 is
  port(
    A, B : in bit;  -- Wejścia
    X    : out bit  -- Wyjście
  );
end AND_2;

architecture A1 of AND_2 is
begin
  X <= A and B;  -- Implementacja bramki AND
end A1;

```

- **Inverter:**

```

entity inverter is
  port(
    A : in bit;  -- Wejście
    X : out bit  -- Wyjście
  );
end entity inverter;

architecture NOTfunction of inverter is
begin
  X <= not A;  -- Implementacja inwertera (NOT)
end architecture NOTfunction;

```

- **NAND Gate:**

```

entity NANDgate is
  port(
    A, B, C : in bit;  -- Wejścia
    X       : out bit  -- Wyjście
  );
end entity NANDgate;

architecture NANDfunction of NANDgate is
begin
  X <= A nand B nand C;  -- Implementacja bramki NAND
end architecture NANDfunction;

```

- **XNOR Gate:**

```

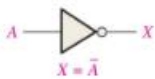
entity XNORgate is
  port(
    A, B : in bit;  -- Wejścia
    X    : out bit  -- Wyjście
  );
end entity XNORgate;

architecture XNORfunction of XNORgate is
begin
  X <= A xnor B;  -- Implementacja bramki XNOR
end architecture XNORfunction;

```

## Wyjaśnienie Kodów:

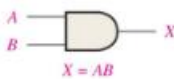
- **entity:** Definiuje interfejs modułu (nazwy i kierunki portów).
- **port:** Określa wejścia i wyjścia modułu.
- **architecture:** Opisuje wewnętrzne działanie modułu.
- **X <= A or B;** – Przypisuje wyjściu X wynik operacji OR na wejściach A i B.
- **not A;** – Operacja negacji na wejściu A.
- **nand, xnor:** Operacje logiczne odpowiadające bramkom NAND i XNOR.



```

entity Inverter is
  port (A: in bit; X: out bit);
end entity Inverter;
architecture NOTfunction of Inverter is
begin
  X <= not A;
end architecture NOTfunction;
  
```

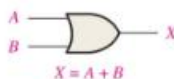
(a) Inverter



```

entity ANDgate is
  port (A, B: in bit; X: out bit);
end entity ANDgate;
architecture ANDfunction of ANDgate is
begin
  X <= A and B;
end architecture ANDfunction;
  
```

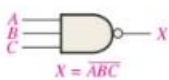
(b) AND gate



```

entity ORgate is
  port (A, B: in bit; X: out bit);
end entity ORgate;
architecture ORfunction of ORgate is
begin
  X <= A or B;
end architecture ORfunction;
  
```


(c) OR gate



```

entity NANDgate is
  port (A, B, C: in bit; X: out bit);
end entity NANDgate;
architecture NANDfunction of NANDgate is
begin
  X <= A nand B nand C;
end architecture NANDfunction;
  
```

(d) NAND gate



```

entity XNORgate is
  port (A, B: in bit; X: out bit);
end entity XNORgate;
architecture XNORfunction of XNORgate is
begin
  X <= A xnor B;
end architecture XNORfunction;
  
```

(e) XNOR gate

VHDL

## 7. Tabela Prawdy

### Tabela Prawdy dla Różnych Funkcji Logicznych:

X	Y	$X \cdot Y$	$\overline{X + Y}$	$\overline{X \cdot Y}$	$X + Y$	$X \oplus Y$	$X \odot Y$	F
0	0	0	1	1	0	0	1	0
0	1	0	0	1	1	1	0	1
1	0	0	0	1	1	1	0	1
1	1	1	0	0	1	0	1	1

### Legenda:

- $X \cdot Y$ : Operacja AND
- $\overline{X + Y}$ : Operacja NOR
- $\overline{X \cdot Y}$ : Operacja NAND

- $X + Y$ : Operacja OR
- $X \oplus Y$ : Operacja XOR (Exclusive OR)
- $X \odot Y$ : Operacja XNOR (Exclusive NOR)
- **F**: Wynik końcowy funkcji logicznej

#### Uwagi:

- Tabela powinna zawierać wszystkie omawiane funkcje logiczne oraz ich kombinacje.
- Można rozszerzyć tabelę o dodatkowe kolumny dla bardziej złożonych funkcji lub większej liczby zmiennych.

## 8. Podsumowanie

Wykład 4 obejmował kluczowe zagadnienia związane z podstawowymi bramkami logicznymi (NAND, NOR), ich implementacją oraz optymalizacją kosztów w projektowaniu układów logicznych. Przedstawiono także wprowadzenie do języków HDL, takich jak VHDL i Verilog, które są niezbędne w opisie i symulacji układów cyfrowych. Omówiono pojęcie iloczynu pełnego (minterm), które jest fundamentalne dla zrozumienia funkcji logicznych w układach cyfrowych. Przykłady kodów VHDL ilustrują praktyczne zastosowanie teorii w tworzeniu rzeczywistych bramek logicznych.

**Czym są bramki logiczne?** Bramki logiczne to podstawowe elementy cyfrowych układów elektronicznych, które działają jak małe przełączniki decydujące, czy sygnał (reprezentowany przez 0 lub 1) przechodzi dalej, czy zostaje zatrzymany. Najważniejsze bramki to:

- **NAND**: Działa jak odwrócony OR. Przechodzi sygnał (1) tylko wtedy, gdy oba wejścia są 0.
- **NOR**: Działa jak odwrócony AND. Przechodzi sygnał (1) tylko wtedy, gdy oba wejścia są 0.

**Dlaczego optymalizacja jest ważna?** Podczas projektowania układów logicznych, każda bramka ma swój koszt (np. 1 PLN). Im mniej bramek użyjemy, tym taniej będzie produkcja układu, zwłaszcza gdy produkujemy ich miliony. Optymalizacja pozwala na redukcję liczby bramek poprzez zastosowanie praw logiki, takich jak dystrybucja.

**Jak działają języki HDL?** Języki opisu sprzętu, takie jak VHDL i Verilog, pozwalają inżynierom opisywać, jak małe elementy układu (np. bramki logiczne) powinny się zachowywać i współpracować ze sobą. Dzięki temu można symulować i testować układ przed jego fizycznym stworzeniem, co oszczędza czas i koszty.

**Czym jest minterm?** Minterm to sposób reprezentacji wszystkich możliwych kombinacji wejść dla danej funkcji logicznej. Dla każdej kombinacji wejść istnieje dokładnie jeden minterm, co

pozwała na pełne opisanie funkcji logicznej w sposób kanoniczny. To jest podstawą dla metod projektowania i analizy bardziej złożonych układów.

**Podsumowanie:** Wykład wyjaśnił, jak podstawowe bramki logiczne są używane do budowy bardziej złożonych systemów komputerowych, jak optymalizować ich liczbę, aby zmniejszyć koszty, oraz jak używać języków HDL do opisu i symulacji tych systemów. Zrozumienie mintermów pomaga w pełnym opisie funkcji logicznych, co jest kluczowe dla efektywnego projektowania układów cyfrowych.

---