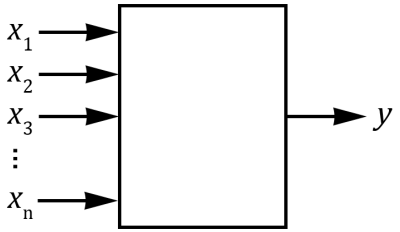
# Systemy cyfrowe i podstawy elektroniki

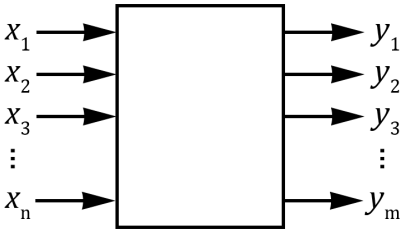
Adam Szmigielski
aszmigie@pjwstk.edu.pl
materiały: $ftp(public) : //aszmigie/SYC$

# Kombinacyjne bloki funkcjonalne - wykład 8

# Funkcja Boolowska a kombinacyjny blok funkcjonalny




funkcja boolowska




kombinacyjny blok funkcjonalny

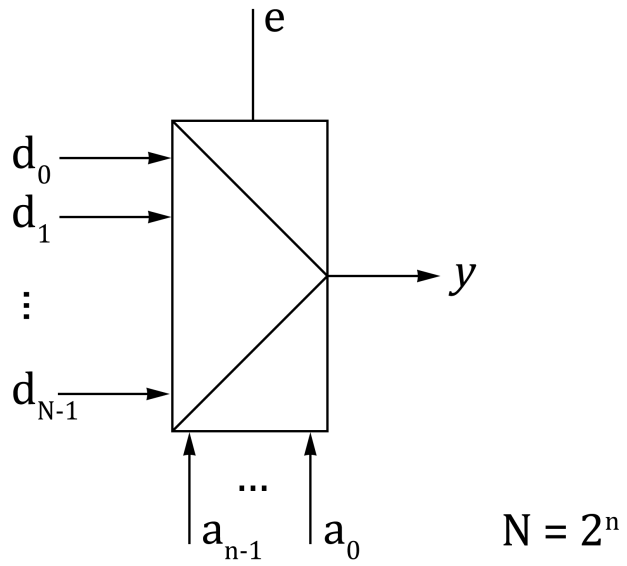
- *Kombinacyjny blok funkcjonalny* w technice cyfrowej jest układem kombinacyjnym złożonym z $n$ wejściach i $m$ wyjść, gdzie $m, n = 1, 2, \ldots$ są liczbami naturalnymi.
- *Funkcja Boolowska* jest szczególnym przypadkiem *kombinacyjnego bloku funkcjonalnego* - posiada tylko jedno wyjście $m = 1$.

## Kombinacyjne bloki funkcjonalne

Przykłady *kombinacyjnych bloków funkcjonalnych*

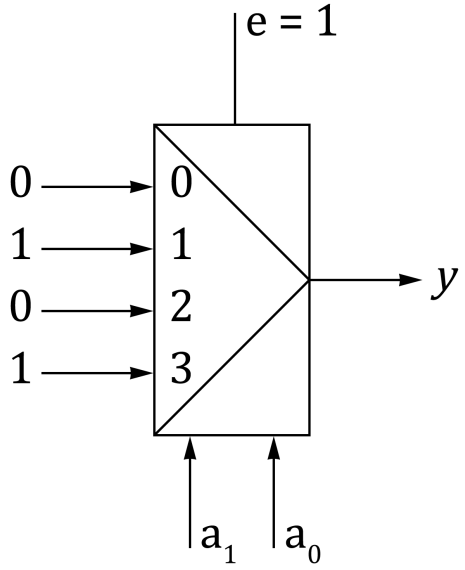
- układy komutacyjne:
  - multipleksery MUX,
  - demultipleksery DMUX,
  - konwertery kodów, dekodery DEC,
- układy arytmetyczne:
  - sumatory,
  - komparatory,
  - ...
- inne.

# Multiplekser (MUX)



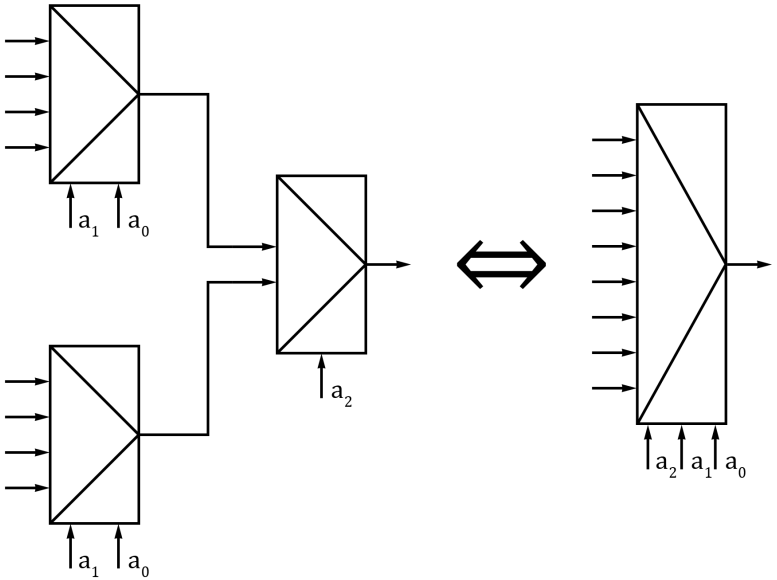

- W multiplekserze wyróżnia się dwa rodzaje wejść - *wejścia adresowe* i *wejścia informacyjne*,
- *Multiplekser* to funkcjonalny blok kombinacyjny, w którym jest $n$ wejść adresowych i $N = 2^n$ wejść informacyjnych, wyjście oraz wejście zezwolenia (enable).

# Multiplekser jako przełącznik



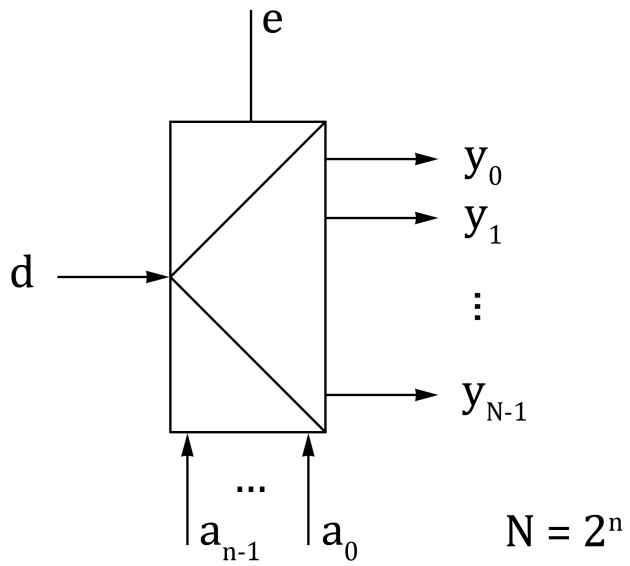

- Multiplekser pracuje jako przełącznik,
- Dany multiplekser realizuje funkcję $y = \overline{a_1} a_0 + a_1 a_0$
- Multiplekser wypisze na wyjściu taki sygnał jaki jest na wejściu informacyjnym wybranym przez wejścia adresowe.

# Kaskadowe łączenie multiplekserów



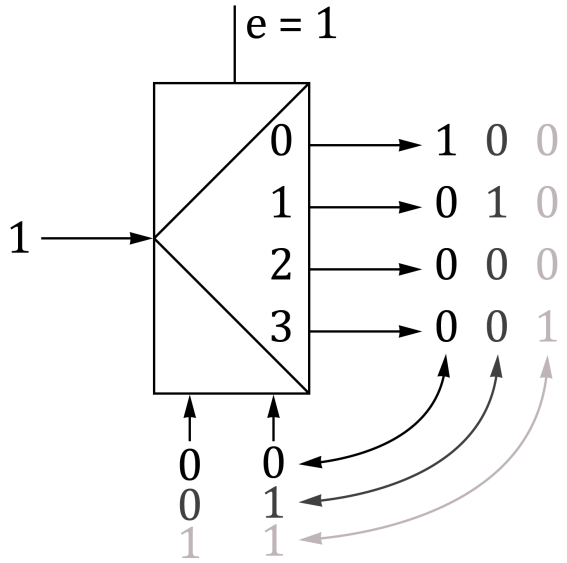

- Liczba wejść informacyjnych multipleksera rośnie wykładniczo dlatego nie realizuje się bezpośrednio multiplekserów o dużej liczbie wejść adresowych,
- Większe multipleksery można budować z mniejszych.

# Demultiplekser (DMUX)



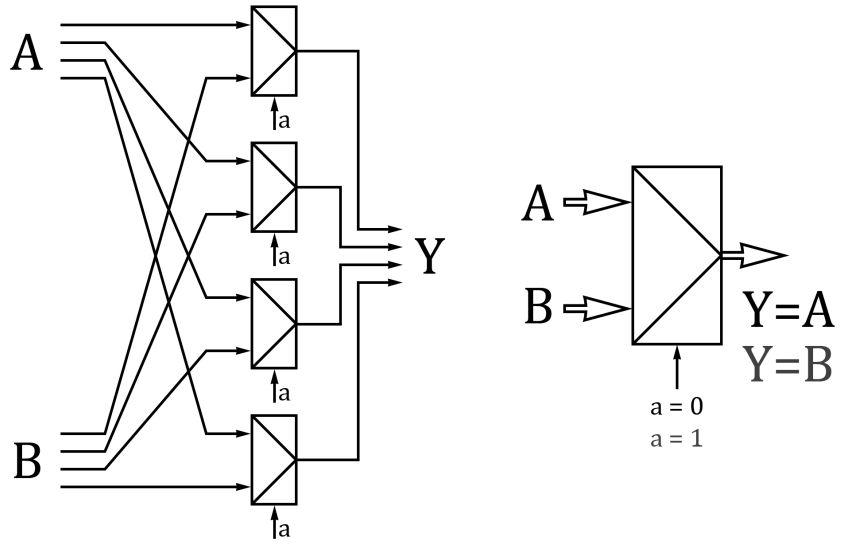

- *Demultiplekser* to układ kombinacyjny o jednym wejściu informacyjnym o $n$ wejść adresowych i $N = 2^n$ wyjściach oraz wejściu zezwalającym .

# Demultiplekser jako przełącznik



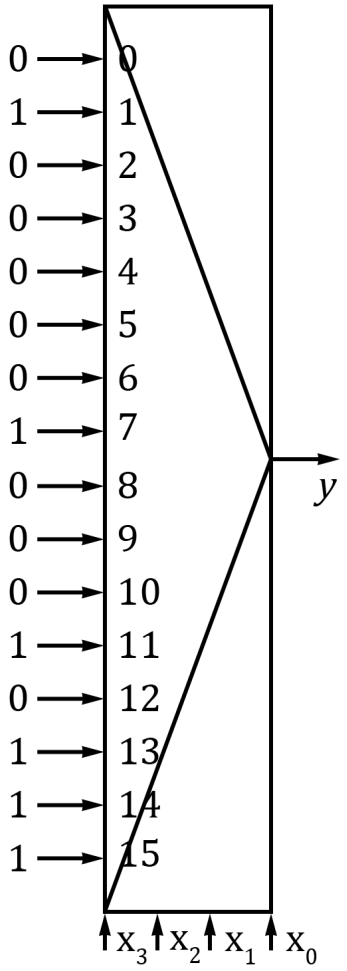

- Demultiplekser pracuje jako przełącznik,
- Demultiplekser wypisze sygnał z wejścia na wyjście wskazane przez stan wejść adresowych.

# Multipleksery i demultipleksery grupowe




- Realizacja bloków komutacyjnych, czyli elementów umożliwiających proste przełączanie sygnałów, jest najczęściej grupowa,
- Multiplekser grupowy (w tym przypadku 4-bitowy) może być dołączane do szyny w zależności od stanu wejścia adresowego .

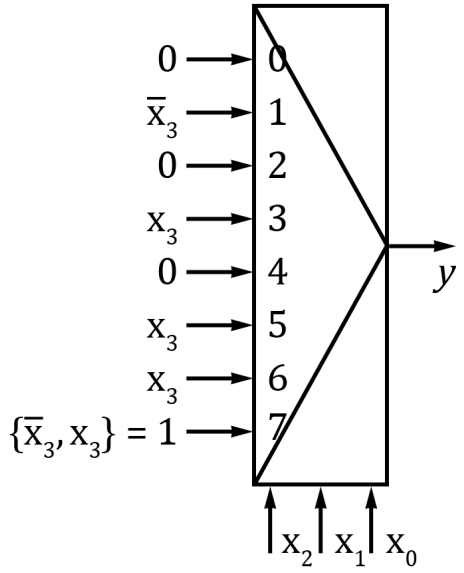
# Realizacja funkcji Boolowskiej za pomocą multipleksera




$y = \Sigma(1, 7, 11, 13, 14, 15)$

# Realizacja funkcji Boolowskiej za pomocą multipleksera o trzech wejściach adresowych - cd.

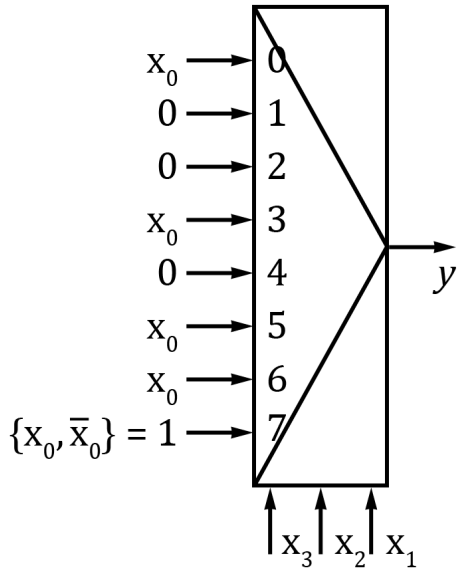
| $y$ | $x_3$ | $x_2x_1x_0$ | $x_2x_1x_0$ |
|---|---|---|---|
| 1 | 0 | 001 | 1 |
| 7 | 0 | 111 | 7 |
| 11 | 1 | 011 | 3 |
| 13 | 1 | 101 | 5 |
| 14 | 1 | 110 | 6 |
| 15 | 1 | 111 | 7 |




- $y = \sum(1, 7, 11, 13, 14, 15)$
- Na 1 wejściu MUX pojawia się $\overline{x_3}$ - potrzebna negacja.

**Realizacja funkcji Boolowskiej za pomocą multipleksera o trzech wejściach adresowych - wybór zmiennych sterujących**

| $y$ | $x_3x_2x_1$ | $x_0$ | $x_3x_2x_1$ |
|---|---|---|---|
| 1 | 000 | 1 | 0 |
| 7 | 011 | 1 | 3 |
| 11 | 101 | 1 | 5 |
| 13 | 110 | 1 | 6 |
| 14 | 111 | 0 | 7 |
| 15 | 111 | 1 | 7 |




- $y = \sum(1, 7, 11, 13, 14, 15)$
- Tym razem negacja jest niepotrzebna.

## Realizacja funkcji Boolowskiej za pomocą multipleksera o dwóch wejściach adresowych

| $x_3x_2\backslash x_1x_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 0 | 0 |
| 01 | 0 | 0 | 1 | 0 |
| 11 | 0 | 1 | 1 | 1 |
| 10 | 0 | 0 | 1 | 0 |

- $y = \sum(1, 7, 11, 13, 14, 15)$
- Jak wybrać wejścia adresowe ?

# cd. - Wybór zmiennych adresowych

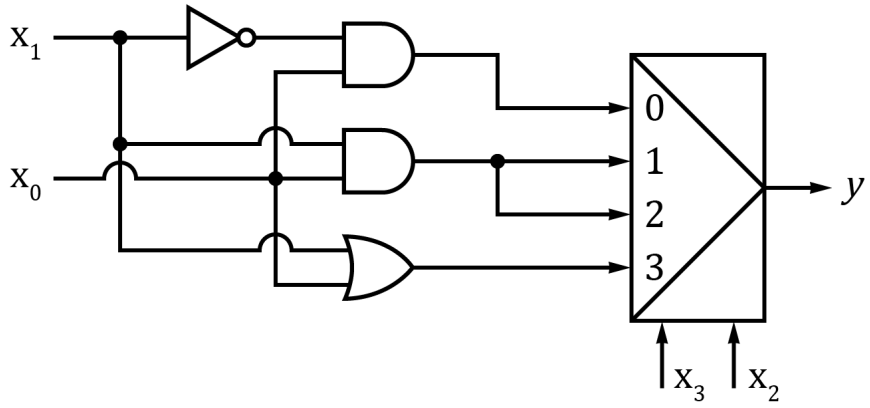
| $x_3x_2 \backslash x_1x_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 0 | 0 |
| 01 | 0 | 0 | 1 | 0 |
| 11 | 0 | 1 | 1 | 1 |
| 10 | 0 | 0 | 1 | 0 |

Na wejście adresowe wybraliśmy $x_3x_2$ wówczas na wejścia informacyjne podajemy wyjście funkcji $f(x_1, x_0)$ opisane poprzez odpowiednie wiersze mapy Karnough-a

- $x_3x_2 = 00 \Longrightarrow f(x_1, x_0) = \overline{x_1}x_0$
- $x_3x_2 = 01 \Longrightarrow f(x_1, x_0) = x_1x_0$
- $x_3x_2 = 11 \Longrightarrow f(x_1, x_0) = x_1 + x_0$
- $x_3x_2 = 10 \Longrightarrow f(x_1, x_0) = x_1x_0$

# cd. - Realizacja

- $x_3x_2 = 00 \Longrightarrow f(x_1, x_0) = \overline{x_1}x_0$
- $x_3x_2 = 01 \Longrightarrow f(x_1, x_0) = x_1x_0$
- $x_3x_2 = 11 \Longrightarrow f(x_1, x_0) = x_1 + x_0$
- $x_3x_2 = 10 \Longrightarrow f(x_1, x_0) = x_1x_0$



# Układy artytmetyczne i logiczne

**Popularne kody liczbowe używane w technice cyfrowej**

- **Kod dwójkowy** - to pozycyjny system liczbowy, w którym podstawą jest liczba 2.
- **Kod 1 z N** - sposób kodowania, w którym słowa binarne o długości $n$ bitów zawierają zawsze tylko jeden bit o wartości 1.
- **HEX** - kod szesnastkowy
- **Kod Graya** - dwójkowy kodem bezwagowy niepozycyjny.
- **kod BCD** (dziesiętny zakodowany dwójkowo) – sposób zapisu liczb polegający na zakodowaniu kolejnych cyfr dziesiętnych liczby dwójkowo przy użyciu czterech bitów.
- **kod ZM** - Znak Moduł
- **kod U1** - uzupełnień do 1
- **kod U2** - uzupełnień do 2

**Kod dwójkowy - Naturalny kod binarny (NKB)**

| pozycja: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| wartość: | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| wartość: | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| bity: | $b_7$ | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ |

- System pozycyjny o podstawie systemu 2
- Liczby określone są bez znaku
- Wartość liczby binarnej (N- długość słowa kodowego)
  $Wartosc = \sum_{i=0}^{N-1} 2^i \cdot b_i$
- Wartość cyfry zależy od pozycji $b_i = 2^i$ (numerowanie od zera)
- $2^N$ różnych wartości kodu (kod pełny)

# Kod 1 z N

| Wartość dziesiętna | Wartość binarna | Kod 1 z 10 |
|---|---|---|
| 0 | 0000 | 0000000001 |
| 1 | 0001 | 0000000010 |
| 2 | 0010 | 0000000100 |
| 3 | 0011 | 0000001000 |
| 4 | 0100 | 0000010000 |
| 5 | 0101 | 0000100000 |
| 6 | 0110 | 0001000000 |
| 7 | 0111 | 0010000000 |
| 8 | 1000 | 0100000000 |
| 9 | 1001 | 1000000000 |

# Kod BCD

| Cyfra dziesiętna | zapis binarny cyfry |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

- np. Liczba 123 składa się z trzech cyfr. Kodując każdą cyfrę binarnie otrzymujemy kod BCD: 0001 0010 0011.
- Kod niepełny,
- Używany ze względu na prostotę konwersji liczb zapisanych dziesiętnie.

# Reprezentacja "znak-moduł" ZM

Najstarszy bit słowa $b_{N-1}$ (MSB - ang. *Most Significant Bit*) pełni rolę znaku (tj. jeśli $b_{N-1} = 1$ to liczna jest ujemna, gdy $b_{N-1} = 0$ dodatnia) np.:

$$
\begin{array}{lcccccccc}
-24_{10} & = & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
118_{10} & = & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
-14_{10} & = & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{array}
$$

$wrtosc = (-1)^{b_{N-1}} \cdot \sum_{i=0}^{N-2} 2^i \cdot b_i$

- Ze względu na najstarszy bit kod nie jest wagowy,
- zakres kodu $< -(2^{N-1} - 1), 2^{N-1} - 1 >$,
- $2^N - 1$ kombinacji - zero posiadałoby dwie reprezentacje (kombinacja 10000000 (minus zero) jest zabroniona),

## kod uzupełnień do 1 (U1) (ang. *1's complement*)

- W zapisie tym najbardziej znaczący bit jest także bitem znaku (0 – liczba dodatnia, 1 – liczba ujemna), ale w zależności od jego wartości dalsze bity zapisu maja różne znaczenie,
  - Jeśli bit znaku jest 0 (liczba dodatnia), to dalsze bity reprezentują liczby dodatnie w ZM,
  - Natomiast gdy bit znaku jest 1 (liczba ujemna), to dalsze bity reprezentują moduł liczby ujemnej, w taki sposób, że zanegowane ich wartości odpowiadają modułowi tej liczby w kodzie ZM,
- Zapis U1 dla liczb dodatnich jest taki sam jak zapis ZM,
- Różnice w zapisie występują jedynie dla liczb ujemnych,
- Zakres liczb tego zapisu jest taki sam jak dla zapisu ZM.

## Kod uzupełnień do 1

- W zapisie U1 występują także dwie reprezentacje zera: 000000...00 i 111111...11,
- Sposób przeliczenia liczby ujemnej w zapisie ZM na zapis U1: Zanegować bity oznaczające moduł liczby (bit znaku pozostaje 1). Np. dla liczb 8-bitowych:

  zapis ZM: 11010110 (dziesiętnie -86)

  zapis U1: 10101001

## Kod uzupełnień do 2 (U2) (ang. *2's complement*)

Najstarszy bit MSB ma wartość ujemną pozostałe bity są dodatnie:

$$wartosc = -2^{N-1} \cdot b_{N-1} + \sum_{i=0}^{N-2} 2^i \cdot b_i$$

- Najstarszy bit identyfikuje czy liczba jest dodatnia czy ujemna,
- Zakres kodu: $< -2^{N-1}, 2^{N-1} - 1 >$,
- $2^N$ kombinacji (kod pełny), zero ma tylko jedną reprezentację,
- Liczby dodatnie z przedziału $< 0, 2^{N-1} - 1 >$ mają identyczną reprezentacje w U2 co w NKB, tj.:
  $(0, b_{N-2}, \ldots, b_1, b_0)_{U2} = \sum_{i=0}^{N-2} 2^i \cdot b_i$
- kod wagowy, najstarszy bit na wartość ujemną. Liczby ujemne można interpretować jako sumę:

$(1, b_{N-2}, \ldots, b_1, b_0)_{U2} = -2^{N-1} + \sum_{i=0}^{N-2} 2^i \cdot b_i$

- wada kodu $U2$: zakres kodu jest niesymetryczny, negacja liczby $-2^{N-1}$ prowadzi do błędu (np. dla $N = 128$ liczba $-128$ mieści się w zakresie, ale 128 już nie),
- Przekroczenie zakresu przy sumowaniu, np. dla $N = 8$:

  $(127)_{U2} + (4)_{U2} = (-125)_{U2}$ - błąd,
- Inkrementacja liczby 127 daje wynik $-128$.

## Negowanie liczb w kodzie U2

$$-(wartosc)_{U2} = \overline{(wartosc)_{U2}} + 1$$

Aby obliczyć liczbę przeciwną do danej w kodzie $U2$ należy zanegować wszystkie bity i do wyniku dodać jedynkę np.:

| | | |
|---|---|---|
| $7_{10}$ | | $(00000111)$ |
| negacja bitów | | $(11111000)$ |
| dodać bit | $+$ | $(00000001)$ |
| wynik $-7_{10}$ | $=$ | $(11111001)_{U2}$ |

## Dodawanie i odejmowanie w kodzie U2

- **Dodawanie** wykonywane jak w NKB, niezależnie od znaków argumentów
- **Wartość przeniesienia z najstarszego bitu jest ignorowana**,
- **Przekroczenie zakresu** może wystąpić w dwóch przypadkach:
  - gdy suma dwóch liczb dodatnich przekracza zakres,
  - gdy suma dwóch liczb ujemnych przekracza zakres,
- **Odejmowanie** w $U2$ **-** dodanie negacji odjemnika tj.:

$$a - b = a + (-b)$$

- wystarczą operacje negowania i dodawania.

## Odejmowanie w kodzie U2 - przykłady

- Sumowanie liczby dodatniej i ujemnej - wynik dodatni,

$25 + (-1)$ :

$$\begin{array}{rcl} 25: & & 00011001 \\ -1: & + & 11111111 \\ \hline (c_7 = 1): & = & 00011000_{U2} = 24_{10} \end{array}$$

- Sumowanie liczby dodatniej i ujemnej - wynik ujemny

$25 + (-56)$ :

$$\begin{array}{rcl} 25: & & 00011001 \\ -56: & + & 11001000 \\ \hline (c_7 = 0): & = & 11100001_{U2} = -31_{10} \end{array}$$

## Dodawanie w kodzie U2 - przykłady

- Sumowanie dwóch liczb dodatnich bez przekroczenia zakresu,

$$
\begin{array}{rcl}
25+1: & & \\
25: & & 00011001 \\
+1: & + & 00000001 \\
\hline
(c_7=0): & = & 00011010_{U2}=26_{10}
\end{array}
$$

- Sumowanie dwóch liczb ujemnych bez przekroczenia zakresu,

$$
\begin{array}{rcl}
(-25)+(-56): & & \\
-25: & & 11100111 \\
-56: & + & 11001000 \\
\hline
(c_7=1): & = & 10101111_{U2}=-81_{10}
\end{array}
$$

## Przekroczenie zakresu w kodzie U2 - przykłady

- Sumowanie dwóch liczb dodatnich z przekroczeniem zakresu,

$112 + 113$ :

| | | |
|---|---|---|
| $112$ : | | 01110000 |
| $113$ : | $+$ | 01110001 |
| $(c_7 = 0, c_6 = 1)$ : | $=$ | 11100001 - **przekroczeniem zakresu** |

- Sumowanie dwóch liczb ujemnych z przekroczeniem zakresu,

$(-75) + (-56)$ :

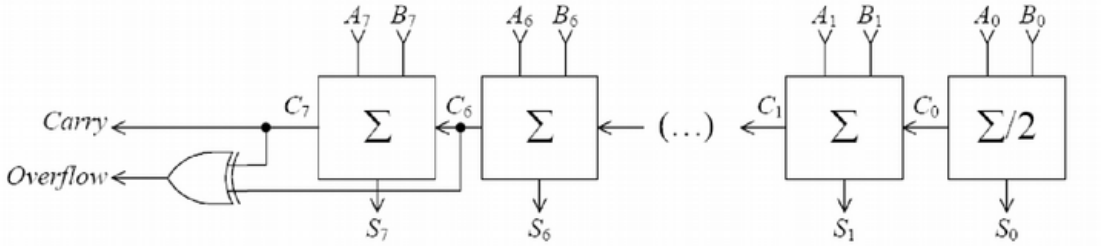
| | | |
|---|---|---|
| $-75$ : | | 10110101 |
| $-56$ : | $+$ | 11001000 |
| $(c_7 = 1, c_6 = 0)$ : | $=$ | 01111101 - **przekroczeniem zakresu** |

# Sprzętowe wykrywanie przekroczenia zakresu w $U2$

- Przekroczenie zakresu w U2 można zidentyfikować analizując przeniesienia:
  przychodzące $C_{IN}$ i generowane $C_{OUT}$ przez najstarszy bit,

| A B $C_{IN}$ | $C_{OUT}$ | S | OFL |
|---|---|---|---|
| 0 0 0 | 0 | 0 | 0 |
| 0 0 1 | 0 | 1 | 1 |
| 0 1 0 | 0 | 1 | 0 |
| 0 1 1 | 1 | 0 | 0 |
| 1 0 0 | 0 | 1 | 0 |
| 1 0 1 | 1 | 0 | 0 |
| 1 1 0 | 1 | 0 | 1 |
| 1 1 1 | 1 | 1 | 0 |

$\Rightarrow$ $OFL = C_{IN} \oplus C_{OUT}$




- Przekroczenie zakresu występuje wtedy i tylko wtedy, gdy oba przeniesienia $C_{IN}$ i $C_{OUT}$ są przeciwnego znaku.

## Reprezentacja liczb rzeczywistych

- **Reprezentacja stałoprzecinkowa** (ang. *fixed point*)
- **Reprezentacja zmiennoprzecinkowa** (ang. *floating point*)

## Reprezentacja stałoprzecinkowa

- W sposób arbitralny przyjmuje się, że część słowa jest *częścią całkowitą*, a pozostała część słowa *część ułamkową*,
- Przykładowo, dla słowa ośmiobitowego przyjmijmy część całkowitą jako 5 bitów a część ułamkową jako 3 bity:

| pozycja: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| wartość: | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ |
| wartość: | 16 | 8 | 4 | 2 | 1 | $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{8}$ |
| bity: | $b_7$ | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ |

## Reprezentacja stałoprzecinkowa

- Liczby stałoprzecinkowe można również interpretować w kodach $U1$, $U2$ czy $ZM$ - najstarszy bit będzie miał znaczenie jak w tych kodowaniach,
- Kodowanie stałoprzecinkowe może powodować błąd,
- Dokładność kodowania zależna jest od długości słowa,
- Niektóre liczby całkowite i wymierne nie mają swojej dokładnej reprezantacji w skończonym kodowaniu,
- Liczby niewymierne zawsze kodowane są z błędem.

# Reprezentacja stałoprzecinkowa - przykład

Podtrzymując założenie że, 5 najstarszych bitów przeznaczone jest na *część całkowitą* a pozostałe 3 bity na *część ułamkową*.
Dodatkowo przyjmujemy, że liczba jest zapisana w kodzie U2:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| wartość: | $-2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ |
| wartość: | $-16$ | $8$ | $4$ | $2$ | $1$ | $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{8}$ |

- Przykładowy ciąg bitów 11001110 jest wówczas równy:
  $-16 + (8 + 1 + \frac{1}{2} + \frac{1}{4}) = -6\frac{1}{4}$,
- Zakres reprezentowanych liczb mieści się w przedziale
  $< -16, 15\frac{7}{8} >$,
- Liczby rzeczywiste są reprezentowane z błędem nie większym od $\frac{1}{8}$.

## Reprezentacja zmiennoprzecinkowa

| eksponent | mantysa |
|---|---|

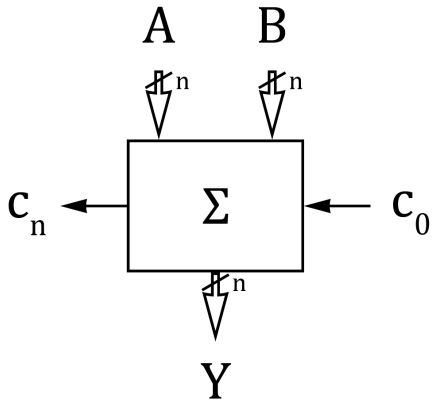
- Liczba zmiennoprzecinkowa jest reprezentowana jako *mantysa* i *eksponent* przy podstawie 2:

$$mantysa \cdot 2^{wykladnik}$$

- *Mantysa* może mieć różne interpretacje (co może być przyczyną nieporozumień),
- *Eksponent* jest liczbą całkowitą dodatnią albo ujemną.

# Dekoder

**Dekoder** zamienia kod *NKB* na *1 z N*.




- Szczególnym przypadkiem demultipleksera jest dekoder, w którym przyjmuje się, że do wejścia *d* zawsze jest dołączony sygnał o wartości logicznej 1. Wejście to nie jest dostępne na zewnątrz układu.

## Sumowanie

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $76_{10}$ | | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| $188_{10}$ | + | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| $194_{10}$ | = | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| przeniesienie | | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

- Sumowanie dwóch $a, b$ bitów:

$$a_i, b_i, c_i \Rightarrow s_i, c_{i+1}$$

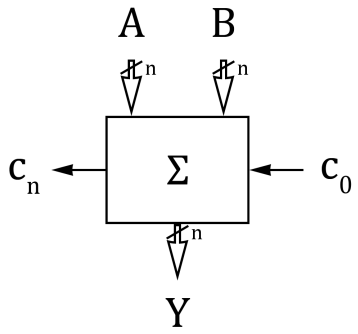
($c$ - przeniesienie, $s$ - wynik sumowania)

# Przekroczenie zakresu

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $152_{10}$ | | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| $118_{10}$ | + | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| $14_{10}$ ? | = | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| przeniesienie | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

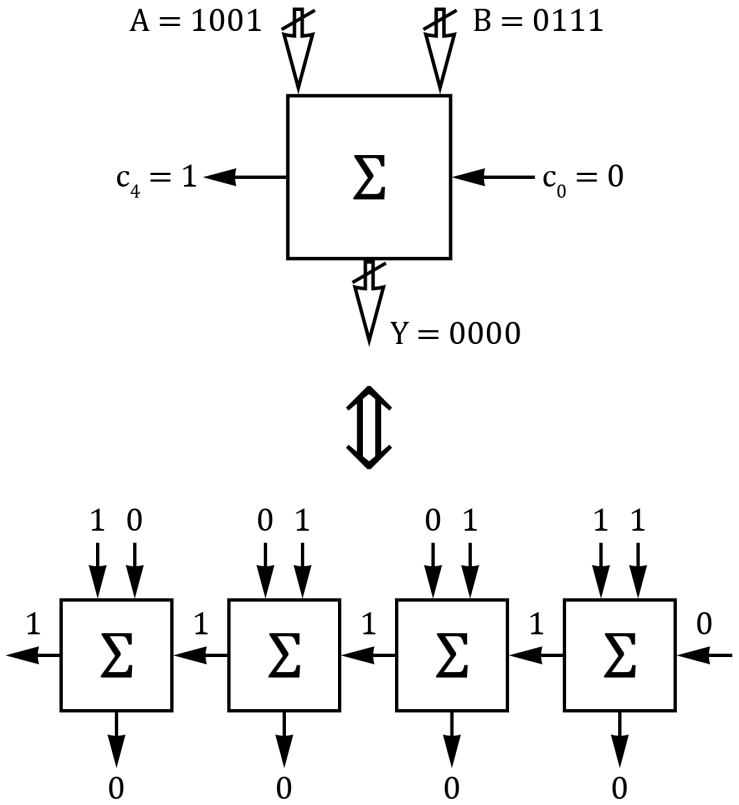
- Przeniesienie z najstarszego bitu ($c_{N-1} = 1$) oznacza przekroczenie zakresu dla słowa $N$-bitowego,
- Alternatywnie: Wystąpienie przeniesienia oznacza, że wynik jest liczbą bitową o długości $N + 1$. Przeniesienie bitu należy wówczas traktować jako $N + 1$ bit wyniku.

# Układy arytmetyczne - sumator




- Operację sumowania arytmetycznego $Y = A + B + c_0$ realizuje sumator. Na wyjściu sumatora powstaje suma $n$-bitowych liczb binarnych $A$ i $B$.
- Przypadek przekroczenia zakresu sygnalizowany jest sygnałem przeniesienia $c_n$.
- Bit przeniesienia można traktować jako najstarszy bit wyniku.

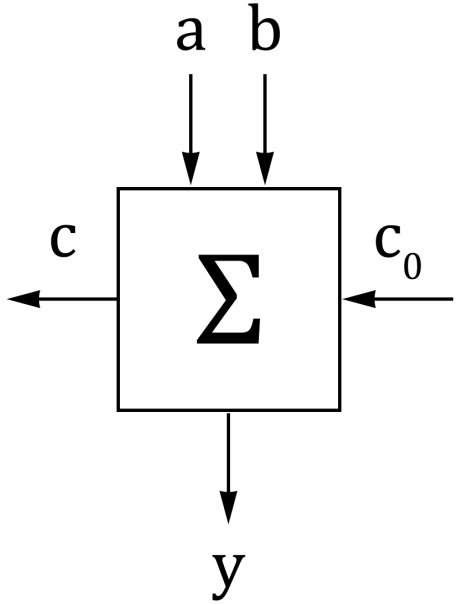
# Budowa kaskadowa sumatora




- W najprostszej realizacji sumator jest zbudowany z kaskadowo połączonych sumatorów jednobitowych, o wejściach $a_i$, $b_i$ i $c_i$, wyjściach $y_i$ i $c_{i+1}$ .

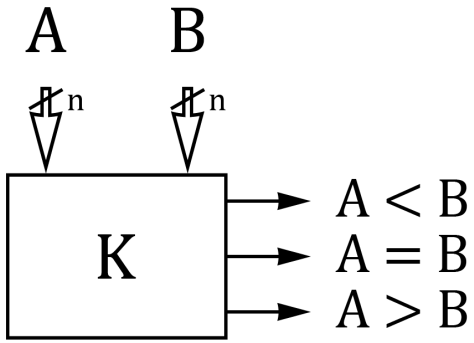
# Budowa sumatora jednobitowego

| a | b | $c_o$ | c | y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |




- $y = a \oplus b \oplus c_o$
- $c = ab + ac + bc_o$

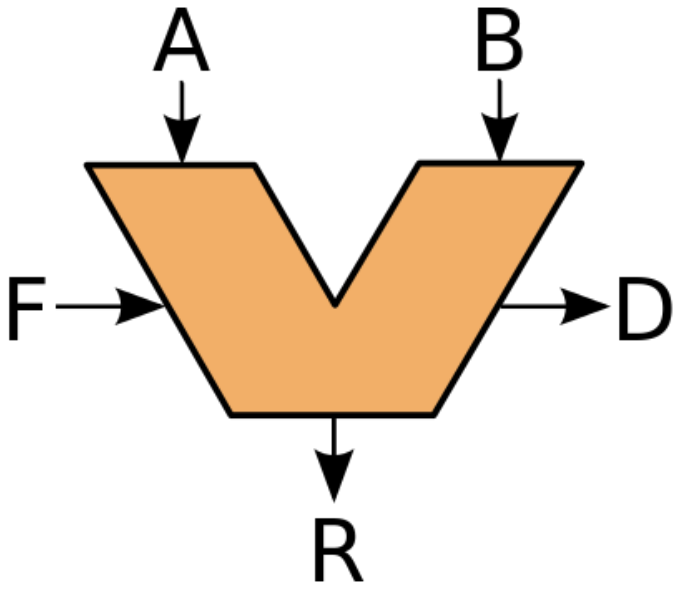
# Komparator




- Komparator umożliwia porównanie dwóch liczb $n$-bitowych i określenie czy są sobie równe, a także która z liczb jest większa, a która mniejsza.

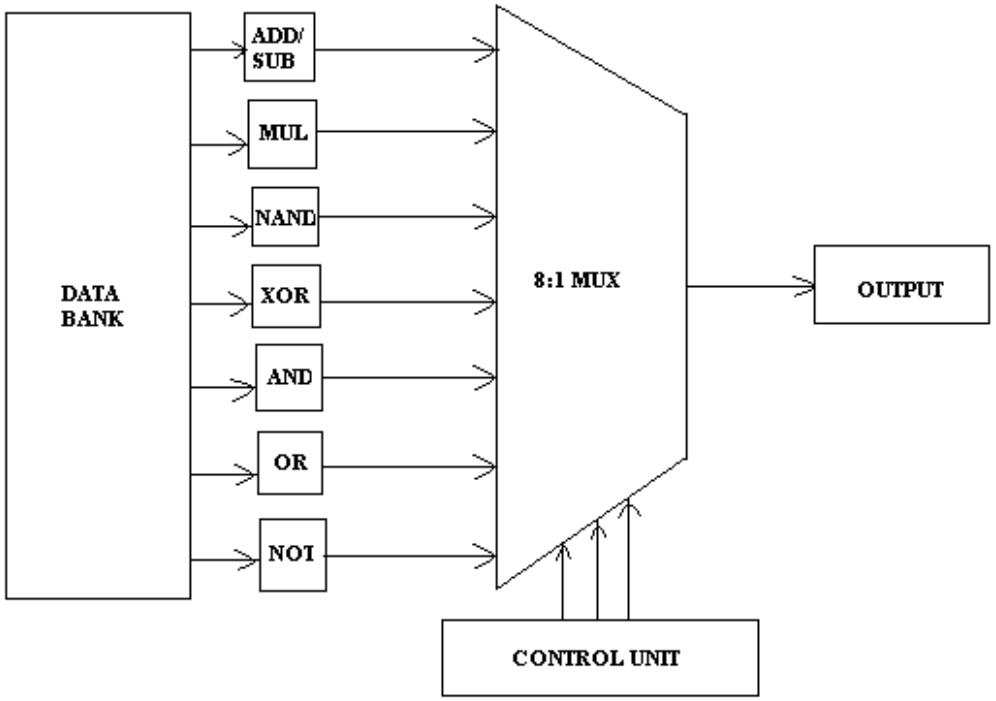
## Jednostka arytmetyczno - logiczna ALU

ALU jest układem cyfrowym kombinacyjnym, służącym do wykonywania operacji arytmetycznych, operacji logicznych pomiędzy dwiema liczbami lub operacji jednoargumentowych.




A i B - dane; R - wyjście; F - wybór operacji; D - status wyjścia

# Realizacja ALU na multiplekserze



# Zadania na ćwiczenia

1. Za pomocą multipleksera o czterech wejściach adresowych zrealizuj daną funkcję.
2. Za pomocą multipleksera o trzech wejściach adresowych i co najwyżej jednego negatora zrealizuj daną funkcję.
3. Zrealizuj układ identyfikujący przekroczenie zakresu dla liczb 4-bitowych w kodzie U2.
4. Zrealizuj 4-bitowe ALU wykonujący 4 działania. Kody operacji:
   00 - ADD - suma arytmetyczna,
   01 - NEG - negacja arytmetyczna,
   10 - XOR,
   11 - AND.
   ALU dodatkowo powinno ustawiać flagi:
   Z=1, gdy wynik operacji wynosi zero,
   C=1, gdy przekroczenia zakresu w kodzie U2.