

Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji
Katedra Informatyki i Inżynierii Systemów

Organizacja Systemów Komputerowych

Materiały do ćwiczeń

Dr inż. Krzysztof Billewicz

Wrocław, 2023

Spis treści

Systemy liczbowe	8
Pozycyjne systemy liczbowe	8
Wprowadzenie	8
Wartości rzeczywiste – ułamki	8
Zera wiodące	9
Wady pozycyjnych systemów liczbowych	9
Dziesiętny system liczbowy	9
Dwójkowy system liczbowy	10
Ułamki w kodzie dwójkowym	12
Podstawy arytmetyki liczb dwójkowych	13
System ósemkowy	13
System szesnastkowy	15
Półbajt, tetrada	17
Mnożenie ułamków szesnastkowych	17
Niepozycyjny system liczbowy	18
Metody konwersji liczb dla różnych zapisów stałopozycyjnych systemów liczbowych..	19
Konwersje liczb z jednego systemu pozycyjnego na inny – metody ogólne	19
Ogólne ujęcie	19
Metoda bezpośrednia I (ang. <i>1. direct methods</i>)	19
Metoda bezpośrednia II (ang. <i>2. direct methods</i>)	21
Metoda różnicowa (ang. <i>differential method</i>)	22
Metoda ilorazowo-iloczynowa (ang. <i>quotient-product method</i>)	23
Metoda podziału (ang. <i>split method</i>)	27
Sposoby kodowania liczb	29
Uzupełnienia p -te i $(p-1)$ -sze	30
Opis uzupełnień	30
Własności uzupełnień	31
Uzupełnienia w różnych systemach pozycyjnych	31
Odejmowanie liczb za pomocą uzupełnień	32
Kodowanie	33
Wprowadzenie	33
Słowo	34
Kody binarne	34

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Naturalny kod binarny (NKB).....	34
Kod Graya.....	35
Kody zrównoważone	39
Kody „ m z n “	39
Kod „1 z n “	39
Kod „1 z 10 “	39
Konwersja BCD do 1 z 10.....	40
Kod znak-moduł	41
Kod uzupełnień do jedności U1.....	43
Problem z arytmetyką w kodzie U1	43
Kod uzupełnień do dwóch U2	44
Kod dwójkowo-dziesiętny BCD	44
Porównanie kodu BCD z dwójkowym.....	45
Kod BCD 84-2-1	46
Zasady.....	46
BCD 8421 a BCD 84-2-1.....	46
Kod Aikena	46
Kod EXCESS-3.....	47
Inne kody BCD.....	48
Kod spolaryzowany	48
Porównanie liczb w różnych kodach	49
Własność samouzupełnienia i sekwencyjność kodów	50
Arytmetyka	50
Przeniesienie	50
Dopisywanie nieznaczących cyfr	51
Działania na liczbach binarnych	51
Negowanie liczb stałopozycyjnych	52
Negowanie w kodzie znak-moduł	52
Negowanie w kodzie U1	52
Negowanie liczb w kodzie U2.....	53
Negowanie liczb w kodzie spolaryzowanym	53
Dodawanie i odejmowanie liczb stałopozycyjnych.....	53
Odejmowanie liczb stałopozycyjnych.....	55
Pożyczka w odejmowaniu liczb binarnych	56
Najprostsze odejmowanie w kodzie ZM	56
Odejmowanie w kodzie U1.....	57

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład wykonywania działań arytmetycznych na liczbach binarnych	59
Odejmowanie w kodzie U2.....	59
Inne podejście do wykorzystania uzupełnień	60
Mnożenie liczb stałopozycyjnych – całkowitych.....	62
Informacje o mnożeniu liczb:	62
Podstawy mnożenia binarnego	62
Mnożenie binarne – metoda bezpośrednia I	64
Mnożenie binarne z wykorzystaniem liczb ujemnych – metoda bezpośrednia I	65
Mnożenie binarne – metoda bezpośrednia II.....	66
Mnożenie liczb rzeczywistych.....	66
Mnożenie binarne – Metoda Bootha.....	68
Mnożenie binarne – Metoda Bootha II.....	68
Mechanizm Bootha – jak działa?	71
Dzielenie liczb binarnych.....	72
Arytmetyka w kodzie BCD.....	73
Negowanie liczb w kodzie BCD.....	73
Dodawanie w kodzie BCD	73
Uzupełnienia w kodzie BCD	74
Liczyby ujemne w kodzie BCD	74
Uzupełnienia w kodzie BCD i EXCESS-3.....	75
Zalety kodu EXCESS-3	76
Korekty przy dodawaniu liczb w kodzie BCD	77
Korekty przy dodawaniu liczb w kodzie EXCESS-3.....	77
Dodawanie liczb ujemnych	77
Odejmowanie w kodzie BCD.....	78
Odejmowanie w kodzie BCD z użyciem uzupełnień.....	78
Podstawy naukowe wykorzystywania uzupełnień w odejmowaniu.....	78
Zasady ogólne	79
Przykład odejmowania - kod BCD z U9 – wynik <0.....	80
Przykład odejmowania - kod BCD z U9 – wynik >0.....	81
Przykład działania - kod BCD z U9 $\rightarrow (-A)+(-B)$	82
Przykład odejmowania - kod BCD z U10 – wynik <0	83
Przykład odejmowania - kod BCD z U10 – wynik >0	84
Przykład działania - kod BCD z U10 $\rightarrow (-A)+(-B)$	85
Przykład odejmowania - kod EXCESS-3 z U9 – wynik <0 EXCESS-3.....	86
Przykład odejmowania - kod EXCESS-3 z U9 – wynik >0.....	87

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład działania - kod EXCESS-3 z U9 $\rightarrow (-A)+(-B)$	88
Przykład odejmowania - kod EXCESS-3 z U10 – wynik <0 EXCESS-3	89
Przykład odejmowania - kod EXCESS-3 z U10 – wynik >0	90
Przykład działania - kod EXCESS-3 z U10 $\rightarrow (-A)+(-B)$	91
Mnożenie BCD	92
Dzielenie w kodzie BCD	92
Liczby zmiennopozycyjne	94
Wprowadzenie do reprezentacji zmiennopozycyjnej liczb	94
Reprezentacja liczby zmiennopozycyjnej	95
Dziesiętna liczba zmiennoprzecinkowa	95
Binarna liczba zmiennoprzecinkowa	96
Mantysa	96
Bit ukryty	97
Wykładnik	97
Skończony zbiór kombinacji mantysy i wykładnika	98
Niedomiar, nadmiar	98
Liczby szczególne, „ponadnormalne” (IEEE 754)	98
Nie-liczba NaN	99
Niedomiar zmiennoprzecinkowy – liczby zbyt małe	99
Różne reprezentacje liczb a standard	99
Standard IEEE 754	100
Wprowadzenie	100
Standard IEEE 754 – pojedyncza precyzja	100
Formaty podstawowe standardu IEEE	100
Standard IEEE 754 – podwójna precyzja	101
Arytmetyka zmiennopozycyjna	102
Dodawanie liczb zmiennopozycyjnych	102
Przykłady dodawania dwóch liczb	103
Przykłady odejmowania dwóch liczb z użyciem uzupełnienia U9	106
Mnożenie i dzielenie liczb zmiennopozycyjnych	108
Mnożenie liczb rzeczywistych	108
Właściwości arytmetyki zmiennoprzecinkowej	109
Algebra Boole’a	110
Wprowadzenie	110
Tablica prawdy	110
Skrócona tablica prawdy	111

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Diagram Venna	111
Pusta prawda.....	113
Podstawowe działania	114
Negacja – Funkcja logiczna NOT	114
Suma logiczna OR.....	114
Iloczyn logiczny AND	116
Różnica symetryczna XOR.....	117
Równoważność \Leftrightarrow	118
Implikacja materialna \Rightarrow	118
Odwrotna implikacja \Leftarrow	121
Zaprzeczona implikacja	121
Zaprzeczona implikacja odwrotna	122
Podstawowe zestawy używanych symboli	123
Podstawowe prawa algebry Boole’a	123
Podstawowe tożsamości algebry Boole’a.....	124
Zebrałe tożsamości algebry Boole’a.....	126
Kilka przykładów	127
Specyfika algebry Boole’a	127
Kolejność wykonywania działań	128
Bramki cyfrowe	128
Definicja.....	128
Bufor cyfrowy	128
Negacja – bramka NOT.....	129
Suma logiczna OR.....	130
Bramka AND.....	133
Prawa de Morgana.....	137
Bramki podstawowe.....	137
Bramka XOR	138
Bramka NOR.....	140
Bramka NAND	142
Brama równoważności XNOR	145
Brama implikacji	145
Dane wejściowe a wszystkie możliwe odpowiedzi	146
Systemy funkcjonalnie pełne.....	147
Funkcje boolowskie	148
Definicja funkcji boolowskiej	148

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Zapisy funkcji boolowskich.....	149
Typy zapisów	149
Funkcje boolowskie zupełne zwracają określone, zawsze takie same wartości dla każdej z możliwych kombinacji wartości zmiennych.....	150
Dysjunkcyjna/sumacyjna postać normalna	150
Koniunkcyjna/iloczynowa postać normalna	152
Tablice Karnaugh'a.....	155
Wykorzystanie tablic Karnaugh'a do upraszczania funkcji.....	159
Metoda implikantów prostych.....	162

Systemy liczbowe

Pozycyjne systemy liczbowe

Wprowadzenie

Liczba jest sposobem reprezentowania wartości arytmetycznej, liczebności lub miary określonej wielkości. System liczbowy można uznać za matematyczną notację liczb wykorzystujących w określony sposób zestaw cyfr lub symboli. Mówiąc prościej, system liczbowy jest metodą przedstawiania liczb. Każdy system liczbowy jest identyfikowany za pomocą jego podstawy. Przykładowo podstawa równa się 10 oznacza, że w tym systemie liczbowym występuje 10 różnych symboli. Uogólniając, jeżeli podstawa równa się "x" oznacza to, że w tym systemie liczbowym istnieje "x" różnych symboli.

Pozycyjny system liczbowy (ang. *Positional Number Systems* lub ang. *positional notation*) jest to taki sposób zapisywania liczb, w którym pozycja cyfry w ciągu oznacza wielokrotność potęgi liczby uważanej za bazę/podstawę danego systemu liczbowego. Każda pozycja ma niezmienną oraz jednoznacznie określoną wagę liczbową. Jeżeli dana potęga jest nieużywana lub jest niepotrzebna do zapisu danej liczby, to zapisuje się tam cyfrę zero (pomiędzy innymi cyframi) lub pozostawia to miejsce puste (przed pierwszą cyfrą różną od zera i po ostatniej).

W pozycyjnych systemach liczbowych wykonywanie działań arytmetycznych jest bardziej skomplikowane niż w systemach niepozycyjnych.

W celu przedstawiania liczb w danym systemie pozycyjnym wykorzystuje się cyfry z zakresu: 0, 1, 2, ... $p-1$, gdzie p – podstawa systemu liczbowego.

Pozycyjny system liczbowy	Podstawa	Wykorzystuje cyfry z zakresu $0 \div (p-1)$	Wykorzystywane cyfry
Dwójkowy	2	$0 \div 1$	0,1
Ósemkowy	8	$0 \div 7$	0,1,2,3,4,5,6,7
Dziesiętny	10	$0 \div 9$	0,1,2,3,4,5,6,7,8,9
Szesnastkowy	16	$0 \div F$	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Poszczególne cyfry są ustawiane na kolejnych pozycjach w liczbie. Wartość liczby wyznaczamy sumując iloczyny kolejnych cyfr pomnożonych przez wagi ich pozycji.

Wartości rzeczywiste – ułamki

Wartości ułamkowe (ang. *fractions*) przedstawione są za pomocą separatora oddzielającego część całkowitą liczby od części ułamkowej. Znak wykorzystywany jako separator może się różnić w różnych krajach. Najczęściej stosuje się kropkę albo przecinek.

Cyfry po lewej stronie separatora mnożone są przez kolejne, rosnące, całkowite potęgi podstawy systemu liczbowego, poczynając od 0, natomiast cyfry po prawej stronie separatora mnożone są przez kolejne, malejące, całkowite, ujemne potęgi podstawy systemu liczbowego, poczynając od (-1).

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Liczby wymierne przedstawia się albo za pomocą skończonej liczby znaków (czyli niedokładnie) albo pokazując, że od pewnego miejsca są okresowe – biorąc w nawias cyfry dalej cyklicznie powtarzające się. Wartości ułamkowe reprezentowane są zgodnie z tą samą zasadą, co liczby całkowite, jednak stosowane są ujemne potęgi liczby stanowiącej podstawę systemu liczbowego.

Zera wiodące

Zero wiodące (ang. *leading zero*) jest to dowolna cyfra 0, która występuje przed pierwszą niezerową cyfrą w ciągu liczbowym w notacji pozycyjnej. W systemie pozycyjnym zera dodane przed najbardziej znaczącą cyfrą nie mają znaczenia, nie zmieniają wartości liczby. Zera wiodące są używane do tego, aby rosnąca długość liczb (coraz więcej cyfr) odpowiadała kolejności alfabetycznej: np. 22 pojawia się alfabetycznie przed 3, ale po 03.

Zera dodane przed najbardziej znaczącą cyfrą, czyli zera wiodące są bez znaczenia. Podobnie zera w części ułamkowej na końcu liczby (dodane po ostatniej niezerowej cyfrze), matematycznie są bez znaczenia, jednak w inżynierii czasami bywają istotne, przykładowo w metrologii jest różnica pomiędzy wynikami pomiarów 0,5 a 0,50000, ponieważ wiadomo, że drugi pomiar został dokonany precyzyjniej – z większą dokładnością, a ewentualne zaokrąglenia miały miejsce na mniej istotnych pozycjach liczb.

Zero wiodące (zero początkowe, zero nieznaczące) jest to znak zera dopisywany po lewej stronie liczby, niemające wpływu na jej wartość. Zazwyczaj wprowadzone są one ze względu na przyjęty format prezentacji danych np. 01.01.2019 r.

Zera wiodące zostały zaznaczone kolorem czerwonym. Inaczej można powiedzieć, że są to zera rozpoczynające ciągi cyfr określające np. liczby całkowite.

Niektóre urządzenia używają zer wiodących; są to: liczniki, stopery, liczniki kilometrów i zegary cyfrowe.

Wady pozycyjnych systemów liczbowych

Pozycyjne systemy liczbowe mają różne wady. Najpoważniejsze z nich to: Podatność na fałszerstwa, które polegają na dołożeniu/dopisaniu najbardziej znaczącej cyfry np. dodanie do liczby 100 z przodu dodatkowej cyfry 4 spowoduje, że w efekcie otrzyma się wartość 4100, czyli dużo wyższą, niż była ona na początku.

Aby uniknąć tego typu przekłamań dodatkowo wykorzystuje się redundancję np. w postaci słownego zapisu tej samej wartości. Takie rozwiązanie też ma wady, ponieważ przed liczbą 50 można dopisać dodatkową jedynkę, wtedy będzie to liczba 150. W opisie słownym wystarczyłoby na początku dodać trzy litery „sto”, które zajmują stosunkowo niewiele miejsca.

Niekiedy, aby uniemożliwić dopisanie dodatkowych cyfr, przed pierwszą cyfrą stawia się znak równości „=”, natomiast po ostatniej znaczącej cyfrze dodaje się np. znaki „-”.

Dziesiętny system liczbowy

W codziennym życiu do reprezentowania liczb używa się systemu opartego na liczbach dziesiętnych.

Dziesiętny system liczbowy (ang. *decimal number system*) jest to system liczbowy stosowany w jako podstawowy w prawie wszystkich krajach świata. Liczby zapisywane są w ciąg cyfr, przy czym każda kolejna cyfra położona bardziej po lewej stronie powinna zostać pomnożona z kolejną potęgą liczby 10.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Jego podstawą jest liczba 10. Najprawdopodobniej, pochodzi ona od dawnego liczenia na palcach obydwu rąk. Podstawa oznacza, że każda cyfra w liczbie jest mnożona przez liczbę 10 podniesioną do potęgi odpowiadającej pozycji tej cyfry w liczbie.

W tym systemie wykorzystuje się 10 cyfr:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Liczba

3	2	1	0	-1	-2	-3	indeks
2	2	3	4	1	2	3	
10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}	waga pozycji
1000	100	10	1	0,1	0,01	0,001	Mnożnik liczby na danej pozycji

Przykładowa liczba w systemie dziesiętnym jest rozumiana w następujący sposób:

$$0,234 = (2 \cdot 10^{-1}) + (3 \cdot 10^{-2}) + (4 \cdot 10^{-3})$$

Liczba

2235

Jest rozumiana następująco:

$$2 \text{ (dwa) tysiące} = 2 \cdot 1000 = 2 \cdot 10^3$$

$$2 \text{ (dwie) setki} = 2 \cdot 100 = 2 \cdot 10^2$$

$$3 \text{ (trzy) dziesiątki} = 3 \cdot 10 = 3 \cdot 10^1$$

$$5 \text{ (pięć) jedności} = 5 \cdot 1 = 5 \cdot 10^0$$

$$2235 = 2 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

$$2235 = 2 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

W ciągu cyfr poszczególne jedynki są rozumiane w następujący sposób:

- 0001 wartość 10^0
- 0010 wartość 10^1
- 0100 wartość 10^2
- 1000 wartość 10^3

Wzór ogólny, przedstawiający dziesiętną reprezentację liczby wygląda następująco:

$$X = \{ \dots d_2 d_1 d_0, d_{-1} d_{-2} d_{-3} \dots \},$$

a wartość X można zapisać wzorem:

$$X = \sum_i d_i 10^i$$

$$d_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Dwójkowy system liczbowy

Dwójkowy system liczbowy (ang. *binary number system*) jest inaczej nazywany systemem binarnym. Podstawą tego systemu liczbowego jest liczba 2. Do zapisu liczb w tym systemie wykorzystywane są tylko dwie cyfry 1 i 0. Jest on powszechnie

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

stosowany w elektronice i informatyce, gdzie minimalizacja liczby stanów (ograniczenie ich do dwóch), pozwala na prostą implementację sprzętową, odpowiadającą zwykłym stanom: włączony i wyłączony.

Liczby zapisywane są jako ciąg cyfr, podobnie jak w systemie dziesiętnym, jednak każda z kolejnych cyfr (położonych coraz bardziej po lewej stronie) jest kolejną potęgą podstawy, czyli liczby 2.

Liczby przedstawiane w systemie niedziesiętnym czasami oznacza się indeksem dolnym zapisanym w systemie dziesiętnym, a oznaczającym podstawę danego systemu.

Ogólnie, binarna reprezentacja liczby wygląda następująco:

$$Y = \{ \dots b_2 b_1 b_0, b_{-1} b_{-2} b_{-3} \dots \},$$

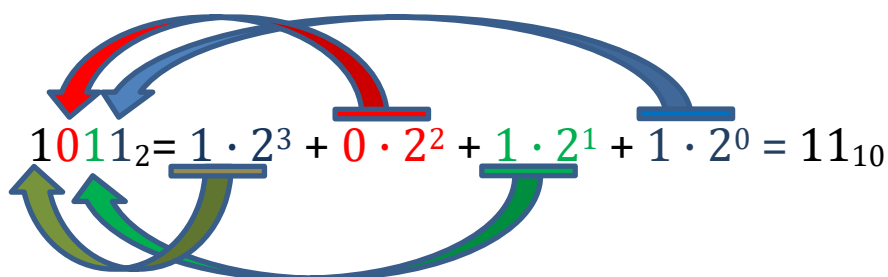
wartość Y można zapisać wzorem:

$$Y = \sum_i b_i 2^i$$

$b_i \in \{0, 1\}$

Liczba

7	6	5	4	3	2	1	0	-1	-2	-3	indeks
1	1	1	1	1	0	1	1	,	1	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	Waga pozycji
10.000.	1.000.	100.	10.	1.	100	10	1	0,1	0,01	0,001	Mnożnik cyfry na danej pozycji
000	000	000	000	000							
128	64	32	16	8	4	2	1	0,5	0,25	0,125	Mnożnik dziesiętny cyfry na danej pozycji



$$1011_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11_{10}$$

1011_2 – liczba w systemie binarnym
 11 lub 11_{10} – liczba w systemie dziesiętnym

$$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$$

$$= 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 11_{10}$$

$$0,1101_2 = 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} =$$

$$= 0 + 1 \cdot 0,5 + 1 \cdot 0,25 + 0 \cdot 0,125 + 1 \cdot 0,0625 = 0,8125_{10}$$

Problem z reprezentacją liczb rzeczywistych w systemie binarnym

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykładowo liczba dziesiętna 0,1, czyli wynik prostego dzielenia $1/10$, nie ma skończonej reprezentacji w systemie binarnym. Nie można znaleźć dokładnej wartości za pomocą sumy skończonej liczby iloczynów wartości, zer i jedynek w binarnej reprezentacji, czyli z kolejnymi ujemnymi potęgami dwójki. W konsekwencji wykonanie prostego działania pomnożenia dwóch liczb $10 \cdot 0,1$ w tym systemie liczbowym jako wynik końcowy niekoniecznie musi zwrócić wartość 1.

Generalnie, jeżeli liczby są zapisywane w kodzie binarnym, istnieje problem z dokładnym przedstawieniem nawet prostych liczb, które w systemie dziesiętnym przedstawione są w postaci ułamków dziesiętnych. Dlatego też poszukuje się innych sposobów zapisywania w systemie binarnym części ułamkowych liczb dziesiętnych.

W społeczeństwie pewne powszechnie obowiązujące wielkości liczb dziesiętnych przedstawia się standardowo z jednym lub dwoma miejscami po przecinku (kwoty w walucie, temperatura ciała, poziom spalania paliwa w samochodzie, waga wielu artykułów spożywczych).

Należy jednak przyznać, że również w systemie dziesiętnym występuje problem z przedstawieniem niektórych ułamków: $1/3$, $1/7$, $1/11$.

Ułamki w kodzie dwójkowym

Ułamek (ang. *fraction*) w dziesiętnym systemie

$$9 \frac{25}{100} = 9,25$$

$$9 \frac{25}{10000} = 9,0025$$

$$\frac{37}{100} = 0,37$$

$$\frac{37}{10000} = 0,0037$$

Ułamek w systemie dwójkowym

$$101,1010_2 = 101_2 + \frac{1010}{10000_2} \rightarrow \text{dziesiętnie} \rightarrow 5 \frac{10}{16_{10}} = 5 + \frac{1}{2} + \frac{0}{4} + \frac{1}{8} + \frac{0}{16} = 5 \frac{5}{8}$$

$$5_{10} = 101_2$$

$$\frac{5}{8_{10}} = \frac{1}{8_{10}} + \frac{1}{2_{10}} = \frac{1}{8_{10}} + \frac{4}{8_{10}} = \frac{101}{1000_2} = 0,101_2$$

$$\frac{5}{32_{10}} = \frac{1}{32_{10}} + \frac{1}{8_{10}} = \frac{1}{32_{10}} + \frac{4}{32_{10}} = \frac{101}{100000_2} = 0,00101_2$$

$$\frac{5}{64_{10}} = \frac{101}{1000000_2} = 0,000101_2$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Podstawy arytmetyki liczb dwójkowych

Dodawanie

Podstawowe działania dodawania (ang. *addition*) w systemie binarnym wykonuje się w analogiczny sposób jak w systemie dziesiętnym.

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10$$

$$10 + 10 = 100$$

Odejmowanie

Odejmowanie (ang. *subtraction*) w systemie binarnym wykonuje się w analogiczny sposób, jak dodawanie.

$$0 - 0 = 0$$

$$0 - 1 = (-1)$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

$$11 - 10 = 1$$

$$\begin{array}{r} 100101_2 = 37_{10} \\ - 11110_2 = 30_{10} \\ \hline 111_2 = 7_{10} \end{array}$$

Niektóre własności liczb binarnych

$2^2_{10} = 100_2$ – jedynka i tyle zer, jaką wartość liczbową ma potęga, w tym przypadku 2;

$$2^5_{10} = 100000_2$$

$7_{10} = 111_2$ – w systemie dwójkowym mnożenie razy 2 wykonuje się przez przesunięcie bitów w lewo na bardziej znaczące pozycje

$$14_{10} = 1110_2$$

$5_{10} = 101_2$ mnożenie razy 4 – przesunięcie w lewo o dwie pozycje

$$30_{10} = 10100_2$$

$$62_{10} = 111110_2 \quad \backslash \cdot 2$$

$$124_{10} = 1111100_2$$

System ósemkowy

System ósemkowy (ang. *octal number system*) jest to system liczbowy, którego podstawą jest liczba 8. Czasem jest on nazywany oktalnym od wyrazu *octal*. W systemie tym wykorzystuje cyfry od 0 do 7. Stosowany np. w Meksyku, gdzie rdzenni Amerykanie liczą wykorzystując przestrzenie między palcami. Każdą cyfrę systemu oktalnego można przedstawić na 3 bitach. Był wykorzystywany np. w komputerach 12-bitowych, 24-bitowych, 36-bitowych. Obecnie wszystkie współczesne platformy komputerowe używają 16, 32- lub 64-bitowych słów maszynowych, dalej dzielonych na 8-bitowe bajty.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Liczba:

4	3	2	1	0	-1	-2	-3	indeks
1	1	0	1	1	1	0	1	
8^4	8^3	8^2	8^1	8^0	8^{-1}	8^{-2}	8^{-3}	Waga pozycji
4096	512	64	8	1	0,125	0,015625	0,001953125	Mnożnik liczby na danej pozycji

Ogólnie, oktalna reprezentacja liczby wygląda w sposób następujący:

$$Y = \{ \dots 020100, 0-10-20-3 \dots \},$$

wartość Y można zapisać wzorem:

$$Y = \sum_i o_i 8^i$$

$$o_i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$$

$$1234_8 = 1 \cdot 8^3 + 2 \cdot 8^2 + 3 \cdot 8^1 + 4 \cdot 8^0 =$$

$$= 1 \cdot 512 + 3 \cdot 64 + 3 \cdot 8 + 4 \cdot 1 = 668_{10}$$

$$0,325_8 = 0 \cdot 8^0 + 3 \cdot 8^{-1} + 2 \cdot 8^{-2} + 5 \cdot 8^{-3} =$$

$$= 0 + 3 \cdot 0,125 + 2 \cdot 0,015625 + 5 \cdot 0,001953125 = 0,142578125_{10}$$

Tabela. Tabliczka mnożenia w systemie ósemkowym

$\times y$	1	2	3	4	5	6	7	10
1	1	2	3	4	5	6	7	10
2	2	4	6	10	12	14	16	20
3	3	6	11	14	17	22	25	30
4	4	10	14	20	24	30	34	40
5	5	12	17	24	31	36	43	50
6	6	14	22	30	36	44	52	60
7	7	16	25	34	43	52	61	70
10	10	20	30	40	50	60	70	100

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

System szesnastkowy

System szesnastkowy (ang. *hexadecimal system*) czasem nazywany jest jako heksadecymalny jest to system liczbowy, w którym podstawą jest liczba 16.

Wykorzystuje on cyfr od 0 do 9 oraz litery alfabetu łacińskiego od A do F.

Litery te odpowiadają odpowiednio wartościom:

A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.

Ogólnie, szesnastkowa reprezentacja liczby wygląda w sposób następujący:

$$Y = \{ \dots h_2 h_1 h_0, h_{-1} h_{-2} h_{-3} \dots \},$$

wartość Y można zapisać wzorem:

$$Y = \sum_i h_i 16^i$$

$$h_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

$$\begin{aligned} 5AF_{16} &= 5 \cdot 16^2 + 10 \cdot 16^1 + 15 \cdot 16^0 = \\ &= 5 \cdot 256 + 10 \cdot 16 + 15 \cdot 1 = 1455_{10} \end{aligned}$$

$$\begin{aligned} 0,ED5_{16} &= 0 \cdot 16^0 + 14 \cdot 16^{-1} + 13 \cdot 16^{-2} + 5 \cdot 16^{-3} = \\ &= 0 + 14 \cdot 0,0625 + 13 \cdot 0,00390625 + 5 \cdot 0,000244141 = \\ &= 0,927001953_{10} \end{aligned}$$

Tabela. Tabliczka mnożenia liczb całkowitych w systemie szesnastkowym

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E	20
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D	30
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C	40
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B	50
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A	60
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69	70
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78	80
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87	90
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96	A0
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5	B0
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4	C0
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3	D0
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2	E0
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1	F0
10	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Tabela. Tabliczka mnożenia ułamków w systemie szesnastkowym

	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,A	0,B	0,C	0,D	0,E	0,F	1,0
0,1	0,01	0,02	0,03	0,04	0,05	0,06	0,07	0,08	0,09	0,0A	0,0B	0,0C	0,0D	0,0E	0,0F	0,10
0,2	0,02	0,04	0,06	0,08	0,0A	0,0C	0,0E	0,10	0,12	0,14	0,16	0,18	0,1A	0,1C	0,1E	0,20
0,3	0,03	0,06	0,09	0,0C	0,0F	0,12	0,15	0,18	0,18	0,1E	0,21	0,24	0,27	0,2A	0,2D	0,30
0,4	0,04	0,08	0,0C	0,10	0,14	0,18	0,1C	0,20	0,24	0,28	0,2C	0,30	0,34	0,38	0,3C	0,40
0,5	0,05	0,0A	0,0F	0,14	0,19	0,1E	0,23	0,28	0,2D	0,32	0,37	0,3C	0,41	0,46	0,4B	0,50
0,6	0,06	0,0C	0,12	0,18	0,1E	0,24	0,2A	0,30	0,36	0,3C	0,42	0,48	0,4E	0,54	0,5A	0,60
0,7	0,07	0,0E	0,15	0,1C	0,23	0,2A	0,31	0,38	0,3F	0,46	0,4D	0,54	0,5B	0,62	0,69	0,70
0,8	0,08	0,10	0,18	0,20	0,28	0,30	0,38	0,40	0,48	0,50	0,58	0,60	0,68	0,70	0,78	0,80
0,9	0,09	0,12	0,18	0,24	0,2D	0,36	0,3F	0,48	0,51	0,5A	0,63	0,6C	0,75	0,7E	0,87	0,90
0,A	0,0A	0,14	0,1E	0,28	0,32	0,3C	0,46	0,50	0,5A	0,64	0,6E	0,78	0,82	0,8C	0,96	0,A0
0,B	0,0B	0,16	0,21	0,2C	0,37	0,42	0,4D	0,58	0,63	0,6E	0,79	0,84	0,8F	0,9A	0,A5	0,B0
0,C	0,0C	0,18	0,24	0,30	0,3C	0,48	0,54	0,60	0,6C	0,78	0,84	0,90	0,9C	0,A8	0,B4	0,C0
0,D	0,0D	0,1A	0,27	0,34	0,41	0,4E	0,5B	0,68	0,75	0,82	0,8F	0,9C	0,A9	0,B6	0,C3	0,D0
0,E	0,0E	0,1C	0,2A	0,38	0,46	0,54	0,62	0,70	0,7E	0,8C	0,9A	0,A8	0,B6	0,C4	0,D2	0,E0
0,F	0,0F	0,1E	0,2D	0,3C	0,4B	0,5A	0,69	0,78	0,87	0,96	0,A5	0,B4	0,C3	0,D2	0,E1	0,F0
1,0	0,10	0,20	0,30	0,40	0,50	0,60	0,70	0,80	0,90	0,A0	0,B0	0,C0	0,D0	0,E0	0,F0	1,00

Tabela. Liczby przedstawione w różnych systemach liczbowych.

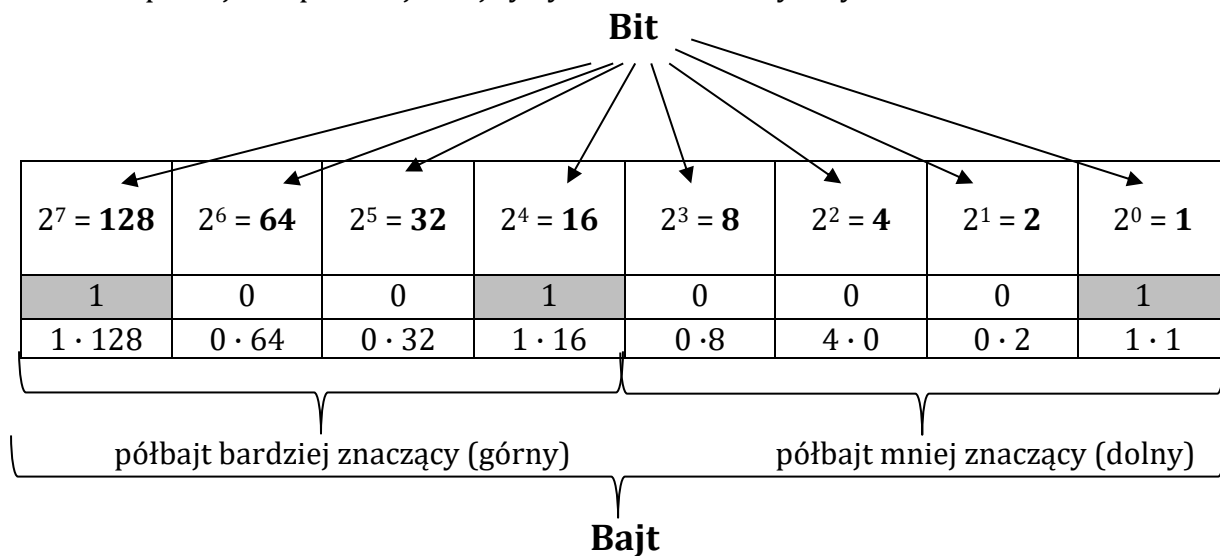
dziesiętne	binarne	ósemkowe	szesnastkowe
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Półbajt, tetrada

Bajt (ang. *byte*), który składa się z 8 bitów można podzielić na dwa nibble lub nybble (ang. *nibble* – kęsek), bądź tetrady. Wyróżnia się mniej znaczący (dolny) i bardziej znaczący (górny) nibble/tetradę, które, odpowiednim składają się z bitów 0–3 i 4–7.

Półbajt może przybierać jedną z 16 wartości. Możliwe jest przedstawienie każdej wartości półbajtu w postaci jednej cyfry w szesnastkowym systemie liczenia.



Mnożenie ułamków szesnastkowych

$$0,8_{16} * 0,8_{16} = \frac{8}{16_{10}} * \frac{8}{16_{10}} = \frac{8 * 8}{16 * 16_{10}} = \frac{64}{256_{10}} = \frac{64}{(16^2)_{10}} = \frac{40}{100_{16}} = 0,4_{16}$$

$$0,8_{16} \cdot 0,8_{16} = 0,1000_2 \cdot 0,1000_2 = 0,1_2 \cdot 0,1_2 = 0,01_2 = 0,0100_2 = 0,4_{16}$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Niepozycyjny system liczbowy

Rzymski system zapisywania liczb nie jest systemem pozycyjnym. Jest systemem addytywnym, czyli takim, w którym dodaje się (lub odejmuje) liczby oznaczane przez używane znaki i symbole. Cyfry mają zawsze tę samą wartość: **I** wynosi jeden, **V** równa się pięć, a **L** – pięćdziesiąt.

Na przykład, symbol **X** zawsze oznacza „dziesięć” niezależnie od tego, gdzie się znajduje:

- Na końcu ciągu cyfr (np. IX),
- Na przedostatniej pozycji (LXV),
- Na pozycji trzeciej od końca (LXVI) lub
- Czwartej od końca (LXVII).

Dla porównania 5 w systemie pozycyjnym może oznaczać: pięć, pięćdziesiąt, pięćset itd.

Przykład

- $LI = L (50) + I (1) = 51$
- $IX = I (-1) + X (10) = 9$

W takich systemach **wykonywanie działań arytmetycznych** jest bardziej skomplikowane niż w systemach pozycyjnych. Problem z przedstawianiem zera.

Tabela. Liczby rzymskie

1	I	11	XI	30	XXX	400	CD
2	II	12	XII	40	XL	500	D
3	III	13	XIII	50	L	600	DC
4	IV	14	XIV	60	LX	700	DCC
5	V	15	XV	70	LXX	800	DCCC
6	VI	16	XVI	80	LXXX	900	CM
7	VII	17	XVII	90	XC	1000	M
8	VIII	18	XVIII	100	C	2000	MM
9	IX	19	XIX	200	CC	3000	MMM
10	X	20	XX	300	CCC		

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Metody konwersji liczb dla różnych zapisów stałopozycyjnych systemów liczbowych.

Konwersje liczb z jednego systemu pozycyjnego na inny – metody ogólne

Ogólne ujęcie

Konwersja liczby z jednego systemu liczbowego na inny polega na znalezieniu jedynego jednoznacznego odpowiednika dla danej liczby w innym systemie liczbowym.

W przypadku systemów pozycyjnych będziemy się posługiwać zapisami ogólnymi:

$$X_p = C_p, U_p = (x_{m-1}, x_{m-2}, \dots, x_1, x_0, x_{-1}, x_{-2}, \dots, x_{-n+1}, x_n)$$

gdzie:

p – początkowy pozycyjny system liczbowy,

X_p – liczba zapisana w początkowym systemie liczbowym p ,

C_p – część całkowita liczby w początkowym systemie liczbowym p ,

U_p – część ułamkowa liczby w początkowym systemie liczbowym p .

Poszukujemy odpowiednika w docelowym systemie liczbowym d

$$X_d = C_d, U_d = (x_{p-1}, x_{p-2}, \dots, x_1, x_0, x_{-1}, x_{-2}, \dots, x_{-q+1}, x_q)$$

gdzie:

d – docelowy pozycyjny system liczbowy,

X_d – liczba zapisana w docelowym systemie liczbowym d ,

C_d – część całkowita liczby w docelowym systemie liczbowym d ,

U_d – część ułamkowa liczby w docelowym systemie liczbowym d .

Metoda bezpośrednia I (ang. 1. direct methods)

Opis

Poszczególne cyfry x_i liczby początkowej X_p w systemie pozycyjnym p wyrażamy przez odpowiedniki docelowego systemu liczbowego d , czyli $(x_i)_d \cdot (p)_d$, wykorzystując zależność:

$$X_d = \sum_{i=-n}^{m-1} (x_i)_s \cdot (p)_s^i$$

Niewątpliwie wykorzystanie tej metody wymaga znajomości wykonywania działań arytmetycznych w docelowym systemie pozycyjnym oraz tabliczki mnożenia (które, w systemach ósemkowych lub szesnastkowym wcale nie jest takie oczywiste). Dlatego ta metoda najczęściej jest wykorzystywana do konwersji na system dziesiętny.

Metoda jest przydatna i łatwa do konwersji systemu dwójkowego na dziesiętny, natomiast trudniejsza przy konwersji w drugą stronę.

Przykład

Konwersja dwójkowo – dziesiętna

Liczba w systemie dwójkowym $X_p = 11100,0011_2$

$$p = (10)_2 = (2)_{10}$$

$$p^{-1} = (10)_{2^{-1}} = (0,1)_2 = (0,5)_{10}$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

$$X_d = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4}$$

$$X_d = 16 + 8 + 4 + 0,125 + 0,0625 = 28,1875$$

Przykład

Konwersja dziesiętno – dwójkowa

Liczba dziesiętna $X_p = 29,25$

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

$$p^{-1} = (10)_{10}^{-1} = (0,1)_{10} = (0,0(0011))_2^{-1}$$

$$X_d = 10 \cdot 1010 + 1001 \cdot 1 + 10 \cdot 0,0(0011) + 101 \cdot 0,00(00001010001111010111)$$

Taka konwersja jest kłopotliwa, ponieważ dziesiętne części ułamkowe 0,1 i 0,01 nie mają skończonej reprezentacji liczbowej w systemie binarnym.

Przykład

Konwersja dwójkowo – szesnastkowa

Liczba w systemie dwójkowym $X_p = 11100,0011_2$

$$p = (10)_2 = (2)_{16}$$

$$p^{-1} = (10)_2^{-1} = (0,1)_2 = (0,8)_{16}$$

rozpisujemy wagi cyfr początkowego systemu liczbowego w nowym systemie liczbowym

$$2_{16}^1 = 2_{16}; 2_{16}^2 = 4_{16}; 2_{16}^3 = 8_{16}; 2_{16}^4 = 10_{16} (!!!!);$$

$$2_{16}^{-1} = 0,8_{16}; 2_{16}^{-2} = 0,4_{16}; 2_{16}^{-3} = 0,2_{16}; 2_{16}^{-4} = 0,1_{16}$$

$$X_d = 1 \cdot 2_{16}^4 + 1 \cdot 2_{16}^3 + 1 \cdot 2_{16}^2 + 0 \cdot 2_{16}^1 + 0 \cdot 2_{16}^0 + 0 \cdot 2_{16}^{-1} + 0 \cdot 2_{16}^{-2} + 1 \cdot 2_{16}^{-3} + 1 \cdot 2_{16}^{-4} = 10_{16} + 8_{16} + 4_{16} + 0,2_{16} + 0,1_{16} = 1C,3_{16}$$

Przykład

Konwersja szesnastkowo-czwórkowa

Liczba w systemie szesnastkowym $X_p = 3A,B_{16}$

Przeliczenie podstawy pierwotnej na podstawę docelowego systemu liczbowego:

$$p = (10)_{16} = (16)_{10} = (100)_4$$

Przeliczenie podstawy pierwszego systemu liczbowego do potęgi minus 1 na docelowy system liczbowy:

$$p^{-1} = (10)_{16}^{-1} = (0,1)_{16} = (0,01)_4$$

$$10_{16}^1 = 100_4; 10_{16}^0 = 1_4$$

Przeliczenie poszczególnych cyfr z systemu pierwotnego na system docelowy:

$$3_{16} = 3_4; A_{16} = 22_4; B_{16} = 23_4$$

$$X_d = 3_4 \cdot 100_4^1 + 22_4 \cdot 100_4^0 + 23_4 \cdot 100_4^{-1} = 300_4 + 22_4 + 0,23_4 = 322,23$$

Podsumowanie

Ta konwersja jest o prosta, ponieważ jednej cyfrze szesnastkowej odpowiadają dokładnie dwie cyfry w systemie czwórkowym (w przypadku pierwszej cyfry kodu szesnastkowego, pierwszą cyfrą kodu czwórkowego może być zero, które jest pomijane – przy takich rachunkach nie pisze się z przodu wiodącego zera).

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Metoda bezpośrednia II (ang. 2. *direct methods*)

Opis

Poszczególne cyfry x_i liczby początkowej X_p w systemie pozycyjnym p wyrażamy przez odpowiedniki docelowego systemu liczbowego d , czyli $(x_i)_d \cdot (p)_d$, wykorzystując zależność:

$$X_d = \sum_{i=-n}^{m-1} x_i p^i = \left(\dots \left((x_{m-1} \cdot p + x_{m-2}) \cdot p + x_{m-3} \right) \cdot p + \dots + x_1 \right) \cdot p + x_0 + p^{-1} \cdot \left(x_{-1} + p^{-1} \cdot (x_{-2} + \dots + p^{-1} \cdot (x_{-(n-1)} + p^{-1} \cdot x_{-n}) \dots) \right)$$

Przykłady

Konwersja dwójkowo-dziesiętna

Liczba w systemie dwójkowym $X_2 = 11100,0011_2$

$$C_2 = 11100$$

$$C_2 = \quad 1 \quad 1 \quad 1 \quad 0 \quad 0$$

$$C_{10} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 0 = 28_{10}$$

$$U_2 = 0011$$

$$U_2 = \quad 0 \quad 0 \quad 1 \quad 1$$

$$U_{10} = 2^{-1} \cdot (0 + 2^{-1} \cdot (0 + 2^{-1} \cdot (1 + 2^{-1} \cdot 1))) = 0,1875_{10}$$

$$X_{10} = 28,1875$$

Jak policzyć takie wyrażenie?

$$U_{10} = 2^{-1} \cdot (0 + 2^{-1} \cdot (0 + 2^{-1} \cdot (1 + 2^{-1} \cdot 1))) = 0,1875$$

Tak by to wyglądało w systemie dziesiętnym:

$$U_{10} = \frac{1}{2} \cdot \left(0 + \frac{1}{2} \cdot \left(0 + \frac{1}{2} \cdot \left(1 + \frac{1}{2} \cdot 1 \right) \right) \right) = \frac{1}{2} \cdot \left(\frac{1}{2} \cdot \left(\frac{1}{2} \cdot \left(\frac{2}{2} + \frac{1}{2} \right) \right) \right) = \frac{1}{2} \cdot \left(\frac{1}{2} \cdot \left(\frac{1}{2} \cdot \frac{3}{2} \right) \right) = \frac{1}{2} \cdot \left(\frac{1}{2} \cdot \frac{3}{4} \right) = \frac{1}{2} \cdot \frac{3}{8} = \frac{3}{16}$$

Konwersja dwójkowo-szesnastkowa

Liczba w systemie dwójkowym $X_2 = 11100,0011_2$

Przeliczenie podstawy jednego systemu liczbowego na docelowy

$$p = (2)_{10} = (10)_2 = (2)_{16}$$

przeliczenie podstawy do potęgi minus 1 na docelowy system liczbowy

$$p^{-1} = (2)_{10}^{-1} = (0,5)_{10} = (10)_2^{-1} = (0,1)_2 = (0,1000)_2 = (2)_{16}^{-1} = (0,8)_{16}$$

$$C_2 = 11100$$

$$C_2 = \quad 1 \quad 1 \quad 1 \quad 0 \quad 0$$

$$C_{16} = (((1 \cdot 2_{16} + 1) \cdot 2_{16} + 1) \cdot 2_{16} + 0) \cdot 2_{16} + 0 = ((7)_{16} \cdot 2_{16}) \cdot 2_{16} = 1C_{10}$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

$$U_2 = 0011$$

$$U_2 = 0 \quad 0 \quad 1 \quad 1$$

$$\begin{aligned} U_{16} &= 2_{16}^{-1} \cdot (0 + 2_{16}^{-1} \cdot (0 + 2_{16}^{-1} \cdot (1 + 2_{16}^{-1} \cdot 1))) = \\ &= (1/2_{16})^1 \cdot (0 + (1/2_{16})^1 \cdot (0 + (1/2_{16})^1 \cdot (1 + (1/2_{16})^1 \cdot 1))) = \\ &= 0,8_{16} \cdot (0 + 0,8_{16} \cdot (0 + 0,8_{16} \cdot (1 + 0,8_{16} \cdot 1))) = 0,3_{16} \end{aligned}$$

$$X_{16} = 1C,3_{16}$$

Jak jeszcze inaczej można policzyć takie wyrażenie?

$$U_{16} = 0,8_{16} \cdot (0 + 0,8_{16} \cdot (0 + 0,8_{16} \cdot (1 + 0,8_{16} \cdot 1))) = 0,3_{16}$$

Tak by to wyglądało w systemie dziesiętnym:

$$\begin{aligned} U_{16} &= \frac{8}{16} \cdot \left(0 + \frac{8}{16} \cdot \left(0 + \frac{8}{16} \cdot \left(1 + \frac{8}{16} \cdot 1 \right) \right) \right) = \frac{8}{16} \cdot \left(\frac{8}{16} \cdot \left(\frac{8}{16} \cdot \left(\frac{16}{16} + \frac{8}{16} \right) \right) \right) \\ &= \frac{8}{16} \cdot \left(\frac{8}{16} \cdot \left(\frac{8}{16} \cdot \frac{24}{16} \right) \right) = \frac{8}{16} \cdot \left(\frac{8}{16} \cdot \frac{12}{16} \right) = \frac{8}{16} \cdot \frac{6}{16} = \frac{3}{16} \end{aligned}$$

Metoda różnicowa (ang. *differential method*)

Opis

Metoda różnicowa (ang. *differential method*) polega na odejmowaniu od liczby początkowej X_p w systemie pozycyjnym p , a później od otrzymanych różnic, kolejnych potęg podstawy d począwszy od najwyższych, ale mniejszych od danej liczby X_p lub od otrzymanej różnicy w wcześniejszego odejmowania.

Kolejne wartości w liczbie X_d o nowej podstawie d uzyskuje się w następujący sposób:

$x_k = 1$ jeżeli odejmowanie potęgi d^k od liczby/różnicy daje wynik nieujemny, czyli jest nie większe od uzyskanej wcześniej różnicy d^k

$x_k = 0$ jeżeli odejmowanie potęgi d^k od liczby/różnicy daje wynik ujemny, czyli jest większe od uzyskanej wcześniej różnicy d^k

Konwersję dokonuje się tak długo, aż uzyskana różnica będzie równa zero lub jeżeli uzyska się wymaganą dokładność.

Przeliczanie całkowitych liczb dziesiętnych na binarne

Przykład

Rozbijamy daną liczbę na kolejne potęgi liczby dwa:

$$X_p = 28_{10} = (16 + 8 + 4)_{10} = (10000 + 1000 + 100)_2 = 11100_2$$

Przykład

Liczba $X_p = 347_{10}$

Ustawiamy w szereg kolejne potęgi liczby dwa:

1024 512 256 128 64 32 16 8 4 2 1

Znajdujemy najwyższą liczbę w szeregu, niższą niż analizowana przez nas liczba 347_{10} – jest to 256_{10} . Widać, że ta potęga dwójki w przykładowej liczbie mieści się 1 raz.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Najbliższa, większa potęga liczby dwa, czyli 512_{10} jest większa niż przykładowa liczba, więc w analizowanej liczbie 347_{10} liczba 512_{10} nie mieści się ani razu = mieści się 0 razy. Dlatego też bierzemy kolejną, mniejszą potęgę liczby 2, czyli wartość 256_{10} . Następnie odejmujemy $347_{10} - 256_{10} = 91_{10}$

W kolejnym kroku również szukamy w szeregu najwyższej liczby, która jednocześnie jest mniejsza niż nasza 91_{10} – jest to 64_{10}

Dalej wykonujemy odejmowanie: $91_{10} - 64_{10} = 27_{10}$

I postępujemy analogicznie

$$27_{10} - 16_{10} = 11_{10}$$

$$11_{10} - 8_{10} = 3_{10}$$

$$3_{10} - 2_{10} = 1_{10}$$

$$1_{10} - 1_{10} = 0_{10}$$

Dalej, jeżeli musieliśmy odejmować od przykładowej liczby, lub dalszych wynikowych różnic którąś daną wartość, to w tabeli wpisujemy 1, w przeciwnym wypadku liczbę 0.

2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
512	256	128	64	32	16	8	4	2	1
	1	0	1	0	1	1	0	1	1

Tak otrzymujemy wynik:

$$347_{10} = 101011011_2$$

Przykład

Liczba $X_p = 24,25_{10}$

$$\begin{array}{ll}
 24,25 - 2^5 = 24,25 - 32 < 0 & \text{– pomijamy lub można przyjąć, że } x_5 = 0 \\
 24,25 - 2^4 = 24,25 - 16 = 8,25 & \rightarrow x_4 = 1 \\
 8,25 - 2^3 = 8,25 - 8 = 0,25 & \rightarrow x_3 = 1 \\
 0,25 - 2^2 = 0,25 - 4 < 0 & \rightarrow x_2 = 0 \\
 0,25 - 2^1 = 0,25 - 2 < 0 & \rightarrow x_1 = 0 \\
 0,25 - 2^0 = 0,25 - 1 < 0 & \rightarrow x_0 = 0 \\
 0,25 - 2^{-1} = 0,25 - 0,5 < 0 & \rightarrow x_{-1} = 0 \\
 0,25 - 2^{-2} = 0,25 - 0,25 = 0 & \rightarrow x_{-2} = 1
 \end{array}$$

$$X_2 = 11000,01_2$$

Metoda ilorazowo-iloczynowa (ang. *quotient-product method*)

Opis

Cyframi x_k części całkowitej C_d liczby X_d są reszty z dzielenia r_k najpierw części całkowitej danej liczby X_p przez podstawę nowej liczby d , a następnie dzielenia kolejnych uzyskiwanych ilorazów I_k .

$$C_p : d = I_0 \rightarrow \text{reszta } r_0 \rightarrow x_0 = r_0$$

$$I_0 : d = I_1 \rightarrow \text{reszta } r_1 \rightarrow x_1 = r_1$$

....

$$I_{m-3} : d = I_{m-2} \rightarrow \text{reszta } r_{m-2} \rightarrow x_{m-2} = r_{m-2}$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

$$I_{m-2} : d = I_{m-1} \rightarrow \text{reszta } r_{m-1} \rightarrow X_{m-1} = r_{m-1}$$

Cyframi x_k części ułamkowej U_d liczby X_d są cyfry nadmiaru (części całkowite) r_k uzyskiwane w wyniku mnożenia najpierw części ułamkowej danej liczby X_p przez podstawę nowej liczby d , a następnie mnożenia kolejnych uzyskiwanych iloczynów I_k pomniejszonych o odjęte od nich części całkowite.

$$U_p \cdot d = I_0 \rightarrow \text{część całkowita } r_0 \rightarrow x_0 = r_0$$

$$I_0 \cdot d = I_1 \rightarrow \text{część całkowita } r_1 \rightarrow x_1 = r_1$$

....

$$I_{n-3} \cdot d = I_{n-2} \rightarrow \text{część całkowita } r_{n-2} \rightarrow x_{n-2} = r_{n-2}$$

$$I_{n-2} \cdot d = I_{n-1} \rightarrow \text{część całkowita } r_{n-1} \rightarrow x_{n-1} = r_{n-1}$$

$$I_{n-1} \cdot d = I_n \rightarrow \text{część całkowita } r_n \rightarrow x_n = r_n$$

Wykonujemy dzielenie z resztą danej dziesiętnej liczby przez dwa w kolejnych krokach, aż otrzymamy 0.

Przykład

Liczba $X_p = 57_{10}$

przepisanie części całkowitej ilorazu /2	/2	reszta z dzielenia
57	28	1 (liczba nieparzysta)
28	14	0
14	7	0
7	3	1
3	1	1
1	0	1
0		wynik → 111001 ₂

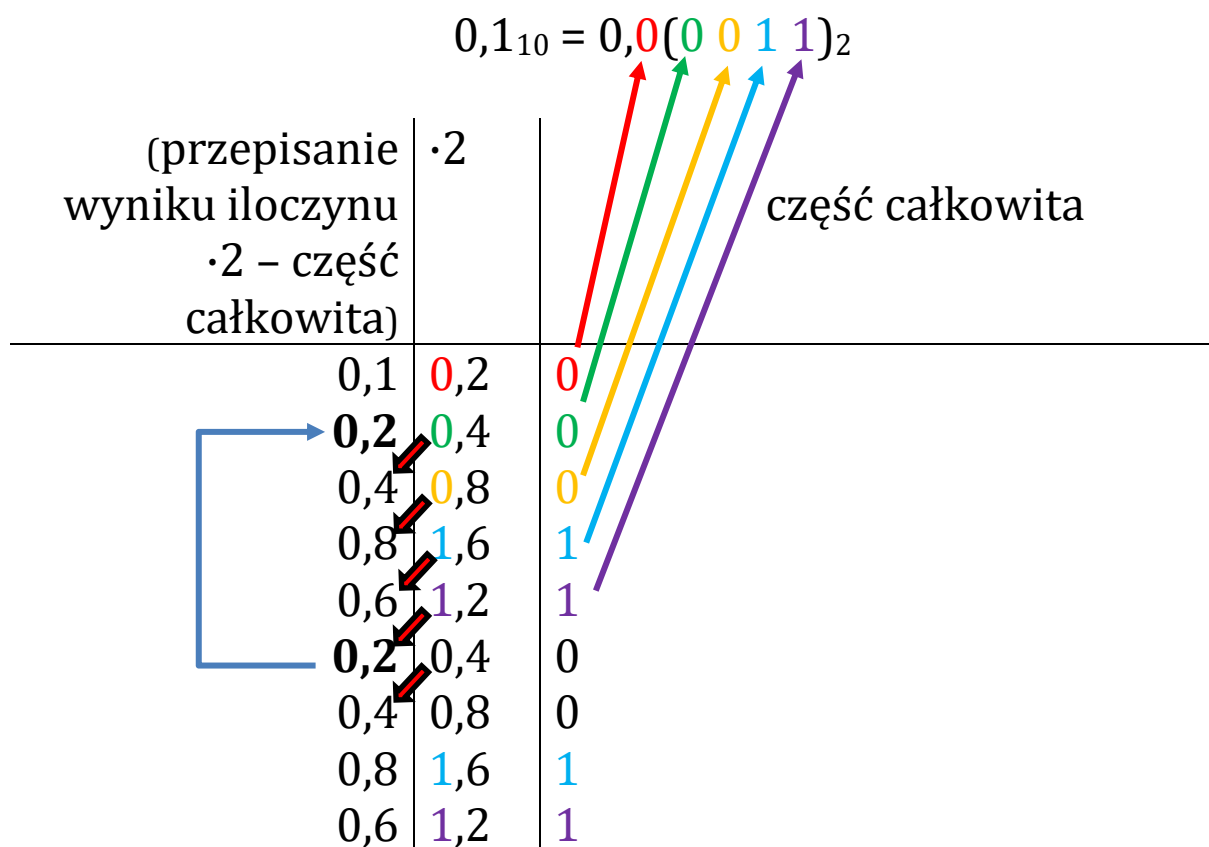
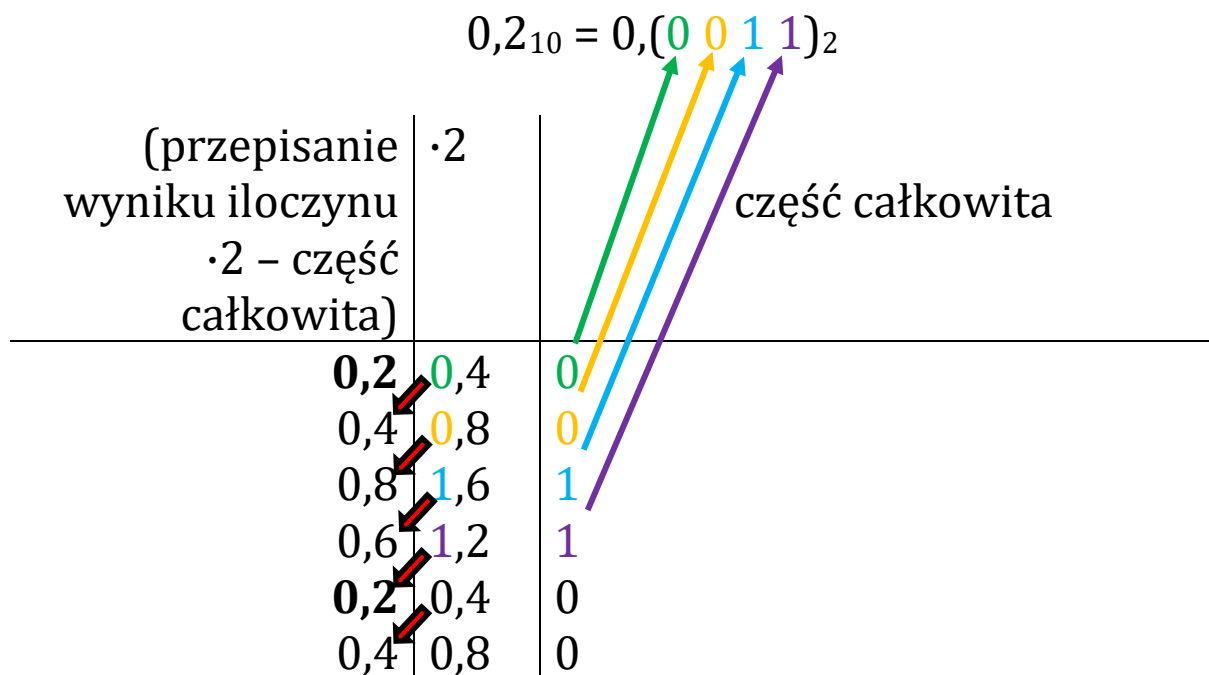
$57_{10} = (32 + 16 + 8 + 1)_{10} = 111001_2$

Przeliczanie ułamków dziesiętnych na binarne

Jeżeli chcemy przeliczyć wartości ułamkowe z systemu dziesiętnego na dwójkowy, to postępujemy analogicznie, jak w przypadku liczb całkowitych, tylko nie wykonujemy dzielenia przez dwa, tylko mnożenie.

$0,125_{10} = 0,001_2$

(przepisanie wyniku iloczynu ·2 – część całkowita)	·2	część całkowita
0,125	0,25	0
0,25	0,5	0
0,5	1	1
0		



„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”**Konwersja dziesiętno-szesnastkowa**

$$X_p = 113,8125_{10}$$

W systemie szesnastkowym występują cyfry {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

$$113:16 = 7 \text{ reszta } x_0 = 1$$

$$7:16 = 0 \text{ reszta } x_1 = 7$$

0 – koniec działania

Następnie część ułamkowa

$$0,8125 \cdot 16 = 13,0 \rightarrow \text{część całkowita } x_{-1} = 13_{10} = D_{16}$$

$$13,0 - 13 \text{ (część całkowita)} = 0 \text{ – koniec działania}$$

$$X_d = 71,D_{16}$$

Konwersja dziesiętno-szesnastkowa – błąd przedstawienia

$$X_p = 1000,1_{10}$$

Założenie: rozwinięcie może mieć co najwyżej 4 cyfry po przecinku. Należy wyliczyć błąd przedstawienia.

W systemie szesnastkowym występują cyfry {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

$$1000:16 = 62 \text{ reszta } x_0 = 8$$

$$62:16 = 3 \text{ reszta } x_1 = 14_{10} = E_{16}$$

$$3:16 = 0 \text{ reszta } x_2 = 3$$

0 – koniec działania

Następnie wyliczana jest część ułamkowa

$$0,1 \cdot 16 = 1,6 \rightarrow \text{część całkowita } x_{-1} = 1_{10} = 1_{16} \text{ reszta } = 0,60$$

$$0,6 \cdot 16 = 9,6 \rightarrow \text{część całkowita } x_{-2} = 9_{10} = 9_{16} \text{ reszta } = 0,60$$

$$0,6 \cdot 16 = 9,6 \rightarrow \text{część całkowita } x_{-3} = 9_{10} = 9_{16} \text{ reszta } = 0,60$$

$$0,6 \cdot 16 = 9,6 \rightarrow \text{część całkowita } x_{-4} = 9_{10} = 9_{16} \text{ reszta } = 0,60$$

W ten sposób wyliczono 4 cyfry po przecinku

Wyliczając kolejne cyfry części ułamkowej widać okresowość:

$$0,6 \cdot 16 = 9,6 \rightarrow \text{część całkowita } x_{-5} = 9_{10} = 9_{16} \text{ reszta } = 0,60$$

$$0,6 \cdot 16 = 9,6 \rightarrow \text{część całkowita } x_{-6} = 9_{10} = 9_{16} \text{ reszta } = 0,60$$

$$0,6 \cdot 16 = 9,6 \rightarrow \text{część całkowita } x_{-7} = 9_{10} = 9_{16} \text{ reszta } = 0,60$$

$$0,6 \cdot 16 = 9,6 \rightarrow \text{część całkowita } x_{-8} = 9_{10} = 9_{16} \text{ reszta } = 0,60$$

(...)

Zatem błąd przedstawienia liczby w kodzie szesnastkowym za pomocą jedynie 4 cyfr po przecinku wynosi $\Delta = 0,000099999... = 0,0000(9)$

$$X_d = 3E8,1999_{16}$$

$$\Delta = 0,0000(9)$$

Metoda podziału (ang. *split method*)

Metoda ta może być wykorzystywana do konwersji liczb z systemu o podstawie $p=2^k$ gdzie k jest liczbą naturalną, (np. dwójkowego, gdzie $k = 1$) do systemu liczbowego o

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

podstawie $q=2^{k+i}$, gdzie i jest liczbą całkowitą oraz $k+i>0$, $i = 1, 2, 3, 4, \dots$, czyli np. do systemu ósemkowego $q=2^{1+2}$ lub szesnastkowego $q=2^{1+3}$.

Metoda ta polega na pogrupowaniu bitów począwszy od znaku podziału – przecinka, w grupach równych wykładnikowi dwójki we wzorze na q (czyli np. 1+3 przy konwersji z systemu dwójkowego na szesnastkowy).

Najczęściej stosowane są:

- Konwersja dwójkowo-szesnastkowa – tetrazy, czyli podział na grupy czterobitowe
- Konwersja dwójkowo-ósemkowa – triady, czyli podział na grupy trzybitowe

Jeżeli grupa usytuowana najbardziej na lewo od przecinka (części całkowite) najbardziej po prawej stronie (części ułamkowe) nie jest pełna – nie ma wymaganej liczby cyfr, to uzupełnia się ją zerami, odpowiednio po lewej lub po prawej stronie.

Przykłady

Konwersja liczb binarnych do systemu ósemkowego

Liczba w systemie dwójkowym $X_p = 11100,0011_2 = 28,1875_{10}$

Konwersja dwójkowo-ósemkowa

$X_p = 11100,0011_2 = 011100,001100_2$

$X_d = 34,14_8$

Liczba binarna 11101101,010 to:

binarnie	11	101	101	010
ósemkowo	3	5	5	2

zatem

$11\ 101\ 101,010_2 = 355,2_8$

Konwersja liczb ósemkowych na binarne

Zamiana kolejnych cyfr liczby ósemkowej na trzycyfrowe reprezentacje binarne.

Przykład

$3_8 = 3_{10} = 011_2$

$6_8 = 6_{10} = 110_2$

$5_8 = 5_{10} = 101_2$

$1_8 = 1_{10} = 001_2$

$2_8 = 2_{10} = 010_2$

$365,12_8 = 11110101,001010_2$

Konwersja liczb binarnych na szesnastkowe i odwrotnie

Liczba w systemie dwójkowym $X_p = 11100,0011_2 = 28,1875_{10}$

Konwersja dwójkowo-ósemkowa

$X_p = 11100,0011_2 = 00011100,0011_2$

$X_d = 1C,3_{16}$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

binarnie	1111	1001	0011
szesnastkowo	F	9	3

zatem

$$1111\ 1001\ 0011_2 = F93_{\text{HEX}}$$

szesnastkowo	B	7	D
binarnie	1011	0111	1101

zatem

$$B7D_{\text{HEX}} = B7D_{\text{h}} = 1011\ 0111\ 1101_2$$

Konwersja liczb z systemu ósemkowego na szesnastkowy

ósemkowo	2	4	7	6
binarnie	10	100	111	110
binarnie	101	0011	1110	
szesnastkowo	5	3	C	

zatem

$$2476_8 = 10\ 101\ 111\ 110_2 = 101\ 0011\ 1110 = 53E_{16} = 53E_{\text{HEX}}$$

ósemkowo	2	4	7	6
binarnie	10	100	111	110
binarnie	1	0100	1111	1000
szesnastkowo	1	4	F	8

zatem

$$24,76_8 = 10\ 100,111\ 110_2 = 1\ 0100,1111\ 10 = 14,F8_{16} = 14,F8_{\text{HEX}}$$

Sposoby kodowania liczb

Ogólnie można wyróżnić dwa sposoby kodowania liczb:

- **Kody stałopozycyjne** (ang. *Fixed-point codes*) – mają ustalone miejsce przecinka, czyli oddzielenia części całkowitych od ułamkowych; jest to określona dokładność pamiętanych liczb i działań wykonywanych na nich np. trzy miejsca po przecinku. Jest stała odległość na osi liczbowej pomiędzy sąsiadującymi liczbami reprezentującymi kolejne wartości. Tu będziemy mieli na myśli również liczby całkowite.
- **Kody zmiennopozycyjne** (ang. *Floating point codes*) – dokładność/precyzja/liczba miejsc po przecinku zależy od wartości wykładnika mnożnika – liczby 10. Analogicznie odległość pomiędzy sąsiednimi liczbami na osi może być zmieniana wartością wykładnika – np. jak zmienia się różnica między takimi samymi wartościami metrów i centymetrów

W matematyce liczby dodatnie przedstawia się bez żadnego znaku np. bez „+”, który podkreślałby to, że są one dodatnie. Systemy komputerowe i cyfrowe muszą mieć możliwość wykorzystywać liczby ujemne i wykonywać na nich działania arytmetyczne. Zatem można zadać sobie pytanie: w jaki sposób za pomocą kodowania zero-jedynkowego przedstawiać liczby ujemne?

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

W matematyce takie liczby poprzedza się znakiem „-”. Jednak w systemach komputerowych, w sprzęcie elektronicznym itp. liczby przedstawia się za pomocą ciągu zer i jedynek.

Oprócz jednoznacznego sposobu określenia reprezentowania poszczególnych liczb jako binarne w systemach komputerowych, kolejnym wyzwaniem jest to, aby móc stosować uniwersalną arytmetykę binarną, która prawidłowo będzie wyliczała wyniki określonych działań, a do tego sama taka arytmetyka będzie stosunkowo prosta do zaimplementowania.

Dotychczas zostało już opracowanych kilka rozwiązań. Na chwilę obecną nie można wskazać definitywnego kryterium, dzięki któremu można byłoby powiedzieć, która z nich jest lepsza. Jak na razie znacząca większość urządzeń cyfrowych wykorzystuje kod uzupełnienia do dwóch U2. Jest to kod dominujący.

Uzupełnienia p -te i $(p-1)$ -sze**Opis uzupełnień**

W liczbowym systemie pozycyjnym o podstawie p , dla $(n+m)$ liczby nieujemnej

$$A = a_{n-1} a_{n-2} a_{n-3} \dots a_1 a_0, a_{-1} \dots a_{-m}$$

gdzie:

n – liczba cyfr części całkowitej liczby A

m – liczba cyfr części ułamkowej liczby A

a_i – i -ta cyfra liczby

p – podstawa systemu liczbowego

wykorzystuje się dwa rodzaje uzupełnień (ang. *complement*):

I. **uzupełnienie do podstawy** $\rightarrow p$ -te (ang. *radix complement*)

$$\bar{\bar{A}} = p^n - A \quad \text{dla } A \neq 0$$

$$(\bar{\bar{A}} = 0 \quad \text{dla } A = 0)$$

II. **uzupełnienie do podstawy zredukowanej** $\rightarrow (p-1)$ -sze (ang. *radix-minus-one complement*)

$$\bar{A} = \bar{\bar{A}} - p^{-m}$$

$$\bar{A} = p^n - A - p^{-m}$$

W zależności od wartości podstawy p systemu liczbowego:

- ($p = 2$) uzupełnienie dwójkowe (ang. *two's complement*) i jedynekowe (ang. *one's complement*),
- ($p = 8$) uzupełnienie ósemkowe (ang. *eight's complement*) i siódemkowe (ang. *seven's complement*),
- ($p = 10$) uzupełnienie dziesiętkowe (ang. *ten's complement*) i dziewiątkowe (ang. *nine's complement*),
- ($p = 16$) uzupełnienie szesnastkowe (ang. *sixteen's complement*) i piętnastkowe (ang. *fifteen's complement*).

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Własności uzupełnień

Dwukrotne uzupełnienie tej samej liczby w wyniku zwraca jej początkową wartość.

- Uzupełnienie p -te uzupełnienia p -tego liczby A

$$\overline{(\bar{A})} = \overline{p^n - A} = p^n - (p^n - A) = A$$

- Uzupełnienie $(p-1)$ -sze uzupełnienia $(p-1)$ -szego liczby A

$$\overline{(\bar{A})} = \overline{p^n - A - p^{-m}} = p^n - (p^n - A - p^{-m}) - p^{-m} = A$$

Uzupełnienia w różnych systemach pozycyjnych

Oto sposoby wyliczenia uzupełnień

System dziesiętny

Przykładowa liczba dziesiętna $A = 12345,678$

$n = 5, m = 3$

Uzupełnienie dziesiątkowe (ang. *ten's complement*) wylicza się w następujący sposób

$$\bar{A} = 10^5 - 12345,678 = 87654,322$$

Uzupełnienie dziewiątkowe od dziesiątkowego różni się tym, że dodatkowo odejmuje się wartość 1 od cyfry na najmniej znaczącej pozycji.

$$\bar{A} = \bar{A} - 10^{-3} = 12345,678 = 87654,321$$

Uzupełnienie dziewiątkowe (ang. *nine's complement*) można również wyliczyć przez odjęcie od liczby składającej się z największych cyfr w danym systemie pozycyjnym (w systemie dziesiętnym jest to cyfra 9) kolejnych cyfr liczby.

$$\bar{A} = 99999,999 - 12345,678 = 87654,321$$

Inaczej można przedstawić to w następujący sposób:

Liczba dziesiętna $A = 13322,7654$



Uzupełnienie dziewiątkowe $\bar{A} = 86677,2345$

$$\begin{array}{r} +10^{-4} \qquad \qquad \qquad +1 \\ \hline \end{array}$$

Uzupełnienie dziesiątkowe $\bar{A} = 86677,2346$

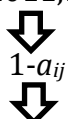
System binarny

W systemie binarnym uzupełnienie jedynekowe można wyznaczyć na kilka różnych sposobów:

- 1) Wyliczając uzupełnienie dwójkowe, a następnie odjęcie wartości 1 od najmniej znaczącej cyfry liczby (czyli od cyfry stojącej najbardziej po prawej stronie),
- 2) Negując poszczególne cyfry liczby, czyli zamieniając jedynki na zera i zera na jedynki,
- 3) Poprzez odjęcie od wartości 1 (największej cyfry występującej w tym systemie liczbowym) poszczególnych cyfr liczby.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Liczba dwójkowa $A = 11011,1010$



Uzupełnienie jedynekowe $\bar{A} = 00100,0101$

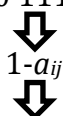
$$\begin{array}{r} +2^{-4} \qquad +1 \\ \hline \end{array}$$

Uzupełnienie dwójkowe $\bar{\bar{A}} = 00100,0110$

Kod BCD

Liczba dziesiętna $A = 68,3$ przedstawiona w kodzie BCD

$$\begin{array}{r} A_{BCD} = 0110\ 1000,0011 \\ + 0110\ 0110,0110 \text{ - korekta} \\ \hline = 1100\ 1110,1001 \end{array}$$



Uzupełnienie dziewiątkowe $\bar{A}_{BCD} = 0011\ 0001,0110$

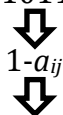
$$\begin{array}{r} +1 \\ \hline \end{array}$$

Uzupełnienie dziesiętkowe $\bar{\bar{A}}_{BCD} = 0011\ 0001,0111$

Kod EXCESS-3

Liczba dziesiętna $A = 68,3$ przedstawiona w kodzie EXCESS-3

$$\begin{array}{r} A_{BCD} = 0110\ 1000,0011 \\ A_{EX3} = 1001\ 1011,0110 \end{array}$$



Uzupełnienie dziewiątkowe $\bar{A}_{EX3} = 0110\ 0100,1001$

$$\begin{array}{r} +1 \\ \hline \end{array}$$

Uzupełnienie dziesiętkowe $\bar{\bar{A}}_{EX3} = 0110\ 0100,1010$

EXCESS-3 ma własność do samouzupełnienia – uzupełnienie dziewiątkowe cyfr dziesiętnych uzyskuje się w wyniku uzupełnienia jedynekowego każdego bitu kodu, czyli przez zanegowanie bitu. Nie każdy kod BCD ma taką własność.

Odejmowanie liczb za pomocą uzupełnień

Uzupełnienia są pomocne podczas realizowania układów cyfrowych dokonujących odejmowania dwóch liczb.

Dwie liczby nieujemne A i B

$$A = a_{n-1} \dots a_1 a_0, a_{-1} \dots a_{1-m}$$

$$B = b_{n-1} \dots b_1 b_0, b_{-1} \dots b_{1-m}$$

Działanie $(A - B)$ może być zrealizowane jako dodanie do odjemnej A uzupełnienia p -tego liczby B , czyli $(A + \bar{\bar{B}})$ lub $(p-1)$ -szego $(A + \bar{B})$ przy zastosowaniu określonych zasad.

$$(A + \bar{\bar{B}}) = A + (p^n - B) = A + p^n - B$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

jeżeli $A \geq B$,

$$A + p^n - B \geq p^n$$

zatem wystąpi przeniesienie.

Jeżeli przeniesienie zostanie zignorowane uzyska się wynik

$$A + \bar{\bar{B}} = A - B \quad (\text{dodatni wynik})$$

jeżeli $A < B$,

$$A + p^n - B < p^n$$

przeniesienie nie będzie miało miejsca.

Zatem wylicza się p-te uzupełnienie $(A + p^n - B)$ ze znakiem minus uzyskuje się:

$$-\overline{A + (p^n - B)} = -\{p^n - (A + p^n - B)\} = -(B - A)$$

$$A + \bar{\bar{B}} = -(A - B) \quad (\text{ujemny wynik})$$

$$(A + \bar{\bar{B}}) = A + (p^n - B) = A + p^n - B$$

jeżeli $A \geq B$,

$$A + p^n - B \geq p^n$$

zatem wystąpi przeniesienie.

Jeżeli przeniesienie zostanie zignorowane uzyska się następujący wynik:

$$A + \bar{\bar{B}} = A - B \quad (\text{dodatni wynik})$$

Kodowanie

Wprowadzenie

Poszukujemy takiego sposobu zapisu liczb – kodowania, który można najprościej wprowadzić do komputerów oraz w którym działania arytmetyczne będą realizowane w najprostszy sposób (proste automatycznie realizowane działania, bez wyjątków).

W cyfrowych układach i systemach wszelkie dane i informacje kodowane są za pomocą symboli, z których najmniejsze części (bity) mogą przyjmować tylko jedną z dwóch cyfr – wartości.

Z punktu widzenia efektywności zarówno przechowywania tych danych jak i ich przetwarzania pożądanym jest, aby wszystkie potrzebne znaki i informacje były kodowane za pomocą jednakowych, najlepiej jak najkrótszych, słów kodowych.

Występują kody detekcyjne i korekcyjne, w których dodaje się nadmiarowe bity do słów kodowych, dzięki którym możliwe jest wykrycie i eliminacja niektórych błędów.

W matematyce liczby dodatnie przedstawia się bez żadnego znaku np. bez „+”, który podkreślałby to, że są dodatnie.

Systemy komputerowe i cyfrowe muszą mieć możliwość stosować i wykonywać działania również na liczbach ujemnych. Zatem pojawia się pytanie: w jaki sposób za pomocą kodowania zero-jedynkowego przedstawiać liczby ujemne? W matematyce takie liczby poprzedza się znakiem „-”. Jednak na komputerach, czy w sprzęcie elektronicznym liczby przedstawia się za pomocą **ciągu zer i jedynek**, bez żadnych **dodatkowych znaków**. Dodatkowo chodzi o to, aby **stosować uniwersalną arytmetykę binarną**, która prawidłowo będzie wyliczała wyniki określonych działań.

Opracowano kilka różnych sposobów kodowania. Na chwilę obecną nie ma definitywnego kryterium, dzięki któremu można powiedzieć, który z nich jest lepszy.

Słowo

Słowo (ang. *word*) – to naturalna jednostka danych, wykorzystywana przez procesor o określonej konstrukcji. Słowo to fragment danych o ustalonym rozmiarze.

Liczba bitów w słowie (długość słowa) jest ważną cechą charakterystyczną każdego opracowanego procesora lub architektury komputera. Rozmiar słowa ma swoje odzwierciedlenie w wielu aspektach struktury i działania komputera; większość rejestrów w procesorze ma zwykle wielkość słowa.

W zależności od organizacji komputera różne jednostki rozmiaru słowa (lub jego wielokrotności) mogą być używane do: przechowywania liczb stałoprzecinkowych i zmiennoprzecinkowych, jako rejestry itp.

Słowo to skończony ciąg symboli/znaków danego alfabetu. Słowem może być np. bajt = 8 bitów.

Kody binarne

W przypadku kodów binarnych dąży się do tego, aby wszystkie liczby przedstawiane w tym kodzie były tej samej długości, stosowano ten sam kod oraz żeby wykorzystywać jak najmniejszą liczbę bitów, żeby przetwarzanie było jak najbardziej efektywne.

Aby dobrać w sposób optymalny sposób kodowania liczb i długość takich kodów należy rozpatrzyć, czy będą rozpatrywane tylko liczby całkowite, czy również i ułamki, oraz z jakiego zakresu liczb będą brane liczby i w jakim zakresie liczb zmieszczą się wartości otrzymane jako wyniki wykonywanych działań.

Oczywiście mówiąc jak najmniejszą liczbę bitów należy rozumieć, że liczba będzie przedstawiona na jak najmniejszej liczbie bajtów. Oczywiście wykorzystanie tylko części bitów może zwiększyć efektywność wykonywania działań na takich liczbach.

Występują również kody, w których nadmiarowe bity wykorzystuje się do kontroli poprawności liczb. Umożliwiają one wychwycenie błędnych wartości i dzięki temu ograniczenie liczby błędów. Zwykle takie kody korekcyjne nie umożliwiają wychwycenia wszystkich błędnych wartości.

Naturalny kod binarny (NKB)

Naturalnym kodem binarnym NKB (ang. *natural binary code NBC*) nazywa się reprezentowanie liczb naturalnych za pomocą słów – liczb binarnych. Jest to system stałopozycyjny. Uporządkowany zbiór cyfr ze zbioru {0, 1} tworzy słowa. Długością słowa nazywamy liczbę cyfr w słowie.

Przyporządkowanie liczbom poszczególnych słów nazywa się kodowaniem. Kodem liczbowym nazywa się sposób, w jaki słowom przyporządkowuje się liczby. Przykładowo zbiór wszystkich słów dwójkowych o długości 10 bitów może reprezentować wszystkie liczby całkowite od 0 do 1023.

Nawet jeżeli mamy liczbę, którą można przedstawić na 2 bitach np. liczbę dziesiętną 3 można przedstawić binarnie jako 11, to jeżeli przyjęta jest **notacja 8 bitowa**, czyli liczby przedstawiane są na 8 bitach niezależnie od tego, jaką wartość przedstawiają to **musimy przedstawiać ją na 8 bitach**, czyli pozostałym bitom, w pewnym sensie nieużywanym i nic nie wnoszącym do liczby, nadać wartości 0. w tym przypadku otrzymamy 00000011.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

To tak, jakbyśmy mieli liczbę dziesiętną 5 i musieli ją każdorazowo zapisywać 8 cyframi, czyli w postaci 00000005. Ta zasada będzie obowiązywać dla bardzo wielu kodów przedstawiających liczby w pamięci komputera.

Kod Graya

Kod Graya (ang. *Gray code*) jest to dwójkowy kod, którego charakterystyczną cechą jest to, że dwa kolejne słowa kodowe różnią się między sobą tylko stanem jednego bitu. Czasami nazywany jest kodem cyklicznym, ponieważ pierwsze i ostatnie słowo również różnią się jedynie stanem jednego bitu. Wykorzystywany jest w przetwornikach analogowo-cyfrowych, zwłaszcza w systemach, w których po sobie występują kolejne wartości np. czujniki położenia/obrotu.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Tworzenie kodu Graya

kod jednobitowy	odbicie lustrzane	kod dwubitowy tworzony przez dopisanie zer i jedynek
0	0	00
1	1	01
	1	11
	0	10

kod dwubitowy	odbicie lustrzane	kod trójbitowy tworzony przez dopisanie zer i jedynek
00	00	000
01	01	001
11	11	011
10	10	010
	10	110
	11	111
	01	101
	00	100

kod trójbitowy	odbicie lustrzane	kod czterobitowy tworzony przez dopisanie zer i jedynek
000	000	0000
001	001	0001
011	011	0011
010	010	0010
110	110	0110
111	111	0111
101	101	0101
100	100	0100
	100	1100
	101	1101
	111	1111
	110	1110
	010	1010
	011	1011
	001	1001
	000	1000

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

lp.	NKB	Kod Graya
0	0	0000
1	1	0001
2	10	0011
3	11	0010
4	100	0110
5	101	0111
6	110	0101
7	111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Konwersja NKB – kod Graya

Nie trzeba tworzyć tablicy i lustrzanych odbić liczb binarnych. Liczbę w kodzie Graya odpowiadającą danej liczbie NKB można znaleźć w następujący sposób:

- 1) Podzielić liczbę przez 2 (binarnie przesunąć ją o jedną pozycję w prawo), najmłodszy bit zostanie stracony, liczbę z przodu uzupełnia się zerem,
- 2) Wykonać różnicę symetryczną (XOR) na kolejnych bitach, biorąc parami bity z liczby w kodzie NKB i uzyskanej liczby podzielonej przez dwa.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

NKB	XOR (\oplus)	Kod Gray'a
000	0000 000 000	000
001	0001 001 001	001
010	0010 010 011	011
011	0011 011 010	010
100	0100 100 110	110
101	0101 101 111	111
110	0110 110 101	101
111	0111 111 100	100

Konwersja kod Gray'a – NKB

Odpowiednie cyfry wyznacza się w sposób iteracyjny, rozpoczyna się od najbardziej znaczącej.

Kolejne cyfry NKB wyznacza się bazując na kolejnych cyfrach kodu Gray'a oraz poprzednio wyznaczoną cyfrę kodu naturalnego.

Krok	Kod Gray'a	XOR	Kod naturalny
1.	1100	1 \rightarrow 1	1---
2.	1100	1 xor 1 \rightarrow 0	10--
3.	1100	0 xor 0 \rightarrow 0	100-
4.	1100	0 xor 0 \rightarrow 0	1000

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Algorytm:

- Pierwsza cyfra NKB (najbardziej znacząca) jest równa najbardziej znaczącej cyfrze kodu Graya,
- Kolejne cyfry NKB wyznaczają się jako różnicę symetryczną (XOR) odpowiedniej cyfry kodu Graya i poprzednio wyznaczonej cyfry kodu NKB.

Zatem odpowiednikiem liczby 1100 w kodzie Graya jest liczba 1000 w kodzie NKB i 8 w kodzie dziesiętnym.

Liczba 1100 w kodzie Graya jest 9 liczbą w tym kodzie przy rozpoczynaniu numeracji od zera.

Kod Greya jest wykorzystywany przy tworzeniu map/tablic Karnaugh'a. Bity/cyfry w liczbie odpowiadają wartościom zmiennych {0, 1} przypisanym do danej pozycji w liczbie. Kolejne liczby kodu Greya odpowiadają odpowiednim kombinacjom zmiennych, przy założeniu, że sąsiadujące ze sobą pola różnią się zmianą wartości tylko jednego bitu.

Kody zrównoważone

W teorii kodowania kod zrównoważony (ang. *balanced code*) jest binarnym kodem korekcji błędów, w którym każde słowo kodowe zawiera równą liczbę zera i jednego bitu.

Kody „ m z n ”

Kod „ m z n ” (ang. *m-of-n code*) jest rozłącznym kodem wykrywania błędów o długości słowa kodowego n bitów, gdzie każde słowo kodowe zawiera dokładnie m wystąpień "jedynek". Wystąpienie pojedynczego błędu bitowego spowoduje, że słowo kodowe będzie miało $m+1$ lub $m-1$ "jedynek", co jednoznacznie wskaże na niepoprawność tego słowa kodowego.

Najprostszą implementacją takiego kodu jest dołączenie ciągu jedynek do oryginalnych danych, aż będzie zawierał on m jedynek, a następnie dołączenie zer w celu utworzenia kodu o długości n .

Kod „1 z n ”

Kod 1 z n (ang. *1-of-n*) – jest to kod, w którym na jednym wyjściu o numerze od 0 do $n-1$ (zmienna logiczna $y_0 - y_{n-1}$) jest zwracana wartość jedynki, a na pozostałych wartość 0.

Jest to kod ze stałą liczbą jedynek i zer w słowie kodowym. Dzięki temu istnieje dodatkowy sposób rozpoznawania błędów w odbieranych lub odczytywanych słowach kodowych.

Kody z tej grupy czasem nazywane są **jeden gorący** (ang. *one hot*)

Kod „1 z 10”

Kod „1 z 10” (ang. *one-out-of-ten code*) jest to przykład kodu „1 z n ”.

Jest to kod wagowy, w którym jedynka umieszczona jest na pozycji (wadze), która odpowiada odpowiedniej cyfrze dziesiętnej.

Tabela. Kod „1 z 10”

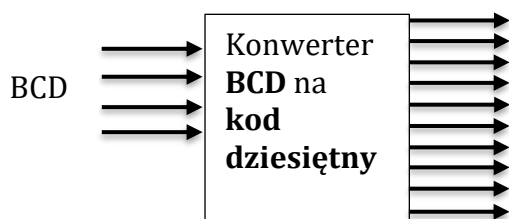
„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Waga	9	8	7	6	5	4	3	2	1	0
Cyfra										
0.	0	0	0	0	0	0	0	0	0	1
1.	0	0	0	0	0	0	0	0	1	0
2.	0	0	0	0	0	0	0	1	0	0
3.	0	0	0	0	0	0	1	0	0	0
4.	0	0	0	0	0	1	0	0	0	0
5.	0	0	0	0	1	0	0	0	0	0
6.	0	0	0	1	0	0	0	0	0	0
7.	0	0	1	0	0	0	0	0	0	0
8.	0	1	0	0	0	0	0	0	0	0
9.	1	0	0	0	0	0	0	0	0	0

Konwersja BCD do 1 z 10

Konwersja kodu BCD do 1 z 10 (ang. *BCD to one-out-of-ten code*) – wynikiem jest liczba zapisana w kodzie „1 do 10”.

Kod 1 z 10 jest wykorzystywany w konwerterach kodu binarnego – cyfr zakodowanych w kodzie BCD na wartość dziesiętną. Jedynek wystawiona na danym wyjściu oznacza wartość dziesiętną liczby zakodowanej binarnie podanej na wejście.



	Wejście BCD				Wyjście dziesiętne									
Nr	x ₃	x ₂	x ₁	x ₀	y ₀	y ₁	y ₂	y ₃	y ₄	y ₅	y ₆	y ₇	y ₈	y ₉
0.	0	0	0	0	0	1	1	1	1	1	1	1	1	1
1.	0	0	0	1	1	0	1	1	1	1	1	1	1	1
2.	0	0	1	0	1	1	0	1	1	1	1	1	1	1
3.	0	0	1	1	1	1	1	0	1	1	1	1	1	1
4.	0	1	0	0	1	1	1	1	0	1	1	1	1	1
5.	0	1	0	1	1	1	1	1	1	0	1	1	1	1
6.	0	1	1	0	1	1	1	1	1	1	0	1	1	1
7.	0	1	1	1	1	1	1	1	1	1	1	0	1	1
8.	1	0	0	0	1	1	1	1	1	1	1	1	0	1
9.	1	0	0	1	1	1	1	1	1	1	1	1	1	0
Błędne	1	0	1	0	1	1	1	1	1	1	1	1	1	1
	1	0	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	0	0	1	1	1	1	1	1	1	1	1	1
	1	1	0	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	0	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Jest to najprostszy i najbardziej przyjazny dla człowieka sposób do przedstawiania liczb dodatnich i ujemnych. Interpretacja takich liczb jest również dość prosta. Bardzo proste jest negowanie liczb, czyli zmienianie znaku. Zmianie ulega tylko wartość jednego bitu.

Przykładowe liczby w kodzie znak-moduł:

$$00110001_{\text{ZM}} = 49_{10}$$

$$10000111_{\text{ZM}} = (-7)_{10}$$

Wadą tego kodu jest podwójna reprezentacja liczby 0. Dla słowa ośmiobitowego wygląda to następująco:

- 10000000 – zero ujemne (-0),
- 00000000 – zero dodatnie (+0).

Dodatnie (+0) i ujemne zero (-0) są to dwie formy przedstawienia, czyli dwie postacie tej samej wartości. Stwarza to pewne problemy podczas tworzenia algorytmów arytmetycznych dla komputerów i układów cyfrowych.

Bez bitu znaku w ośmiobitowym bajcie można przedstawić liczby $0 \div 255$. Rezygnując z jednego bitu w wyniku przeznaczenia go do reprezentowania znaku liczby, na pozostałych siedmiu bitach można przedstawić liczby od 0 do 127. Jednak przy zastosowaniu kodu moduł-znak na ośmiu bitach można przedstawić liczby od (-127) do (+127). Można zatem powiedzieć, że przesuwają się zakres liczb możliwych do przedstawienia, a ich liczba pozostaje taka sama.

Ten kod może przedstawiać nie tylko liczby całkowite, ale również ułamkowe. Może być określone miejsce położenia przecinka. Przykładowo może być tak, że 3 bity odpowiadają za część całkowitą, natomiast pozostałe 4 – za część ułamkową.

Przykłady zakodowania liczb w kodzie znak-moduł, z trzeba bitami przeznaczonymi na część całkowitą liczby i 4 na część ułamkową:

$$0010,0000 = 2_{10}$$

$$0011,1000 = 3,5_{10}$$

$$0111,0100 = 7,25_{10}$$

$$1110,1000 = (-6,5)_{10}$$

$$1011,1100 = (-3,75)_{10}$$

$$1111,1110 = (-7,875)_{10}$$

Zaletą kodu znak-moduł jest łatwiejsze rozkodowywanie liczb zapisanych w pamięci. Jednak układy cyfrowe musiały być bardziej skomplikowane niż przy stosowaniu innych kodów np. U1, czy U2. Kod moduł-znak dawniej był wykorzystywany przez niektóre komputery z serii 704, 709 i 709x produkowane przez IBM.

Odejmowanie binarnej liczby dodatniej od binarnej liczby dodatniej, kiedy wynikiem ma być binarna liczba ujemna jest skomplikowane. Trzeba pamiętać, że kod jest dla maszyny, a nie dla człowieka. Zatem należy dokonać kompromisu pomiędzy łatwością interpretacji przez człowieka i bardziej skomplikowanymi układami elektronicznymi, a trudniejszym kodem dla człowieka, łatwiejszym do zaimplementowania w układach elektronicznych.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Kod uzupełnień do jedności U1

Kod uzupełnień do jedności U1 (ang. *1's complement*). W tym kodzie **liczby dodatnie** reprezentowane są tak, **jak w kodzie NKB**, pod warunkiem, że długość słowa (liczba bitów, na których przedstawiana jest liczba) jest wystarczająco duża, aby najbardziej znaczący bit miał wartość 0. Przykładowo dziesiętna wartość 15 nie może być przedstawiona na jako słowo czterobitowe 1111, tylko na co najmniej pięciobitowe 01111, sześciobitowe 001111 lub siedmiobitowe 0001111.

Liczby ujemne na najbardziej znaczącej pozycji mają **jedynkę**, natomiast pozostałe bity **mają wartości przeciwne**, niż miałyby dla liczby dodatniej w kodzie NKB, czyli wszystkie wartości 1 są zamieniane na 0, a 0 na 1. Zatem podczas budowania cyfrowych układów arytmetycznych najprościej jest to zrobić za pomocą inwerterów.

W kodzie uzupełnień do jedności wykorzystywane jest uzupełnienie *p*-sze, czyli jedynkowe. Zatem można powiedzieć, że liczba ujemna to dopełnienie liczby dodatniej do wartości 11111111. Zakres liczb jest taki sam, jak dla kodu moduł-znak.

Zero, podobnie jak w kodzie znak-moduł, ma dwie reprezentacje: dodatnią 00000000 i ujemną 11111111, niejako przesunięte o 1 bit.

Problem z arytmetyką w kodzie U1

Liczba (+17) przedstawiona w kodzie U1 to 00010001.

Liczba (-17) przedstawiona w tym kodzie to 11101110.

Suma (+17)+(-17) wynosi 11111111 – czyli zero ujemne

Liczba (+18) przedstawiona w kodzie U1 to 00010010.

Liczba (-17) przedstawiona w tym kodzie to 11101110.

Suma (+18)+(-17) wynosi 00000000 – czyli zero dodatnie

Ten wynik jest nieprawidłowy.

Działania arytmetyczne liczb dodatnich z ujemnymi

Trzeba uzupełnić o wartość przeniesienia 1 – stąd nazwa.

(-6) 11111001

+7 00000111

1 00000000 – błędna wartość (+0)

↑ +1 – konieczna korekta wyniku o bit przeniesienia

1 00000001 – prawidłowa wartość (1)

Przeniesienie (ang. *carry*)

(+2) 00000010

(-3) 00000011

(+2) 1 00000010 – powstaje konieczność pożyczki

(-3) 00000011

11111111 – błędna wartość (-0)

-1 – konieczna korekta wyniku o bit pożyczki

11111110 – prawidłowa wartość (-1)

pożyczka (ang. *borrow*)

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Kod uzupełnień do dwóch U2

Konieczny był kod, w którym nie byłoby dwóch sposobów reprezentacji jednej liczby 0. Tak wymyślono kod uzupełnień do dwóch.

Kod uzupełnień do dwóch U2 (ang. *2's complement*) różni się jedynie nieznacznie od kodu uzupełnień do jedności. Tutaj wykorzystywane jest uzupełnienie p-te, czyli uzupełnienie dwójkowe.

Wartość liczby można wyliczyć ze wzoru:

$$liczba = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

W tym kodzie **liczby dodatnie** przedstawiane są tak, jak **w kodzie NKB**. Liczba zero ma tylko jedną reprezentację 0000000.

Liczby ujemne są przedstawiane podobnie jak w kodzie U1, czyli wszystkie bity są odwrócone w stosunku do wartości dodatniej, a następnie do tego wyniku dodaje się liczbę 1.

Liczba (+17) przedstawiona w kodzie **U2** to 00010001.

Liczba (-17) przedstawiona w tym kodzie to 11101111.

Liczba (-17) przedstawiona w kodzie **U1** to 11101110.

Takie rozwiązanie może być zaletą, ale również i wadą. Bardziej skomplikowane jest negowanie liczb, czyli zamiana dodatnie-ujemne. Jednak okazuje się, że w urządzeniach można zastosować go w najprostszy sposób. Zapewne to zadecydowało o jego popularności.

Kod U2 a system binarny – jedna z interpretacji

128	64	32	16	8	4	2	1	
0	0	0	1	0	1	1	0	
			+ 16		+ 4	+ 2		=22
Kod uzupełnień do dwóch								
-128	64	32	16	8	4	2	1	
1	1	1	0	1	0	1	0	
-128	+64	+32		+8		+2		=-22
	=-64	=-32		=-24		=-22		

Kod dwójkowo-dziesiętny BCD

Kod dwójkowo-dziesiętny BCD (ang. *binary coded decimal*), zwany również kodem BCD 8421 – jest to zapis dziesiętny kodowany dwójkowo. Polega on na tym, że każdą cyfrę liczby zapisanej w postaci dziesiętnej reprezentuje grupa 4 cyfr binarnych (tetradą) – 4-bitowa liczba binarna. Czyli każda cyfra dziesiętna jest zapisywana oddzielnie w kodzie dwójkowym.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Czterocyfrowa liczba binarna może przedstawiać 16 różnych kombinacji bitów, czyli 16 możliwych wartości, natomiast w kodzie BCD wykorzystuje się jedynie 10 wartości, ponieważ tyle jest cyfr dziesiętnych – jedna tetrada może zatem odpowiadać jedynie cyfrom od 0 do 9. Pozostałe 6 niewykorzystywanych kombinacji bitów staje się pewnym problemem, dlatego podczas wykonywania niektórych działań na tych liczbach konieczne jest dokonywanie korekt.

Przykład: liczba dziesiętna 73 może być przedstawiona w postaci dwóch tetrad:
 $73_{10} = 0111\ 0011_{\text{BCD}}$
pierwsza tetrada koduje cyfrę 7, a druga cyfrę 3.

Na przedstawienie tej liczby w kodzie NKB wystarczyłoby 7 bitów, natomiast w kodzie BCD potrzeba aż 8 bitów. Zatem, aby przedstawić liczbę w kodzie BCD zazwyczaj potrzeba większej liczby bitów niż gdyby była prezentowana w kodzie NKB.

Z tego wynika, że wadą kodu BCD jest nadmiarowość, wynikająca z istnienia niewykorzystanych sekwencji bitów.

Liczba dziesiętna 117 w kodzie NKB będzie przedstawiona 7 bitach, natomiast w kodzie BCD – aż na 12 bitach.

Kod BCD jest bardzo podobny do kodu dziesiętnego. Wykorzystywany jest w elektronice i informatyce, przede wszystkim w urządzeniach elektronicznych z wyświetlaczem cyfrowym (np. w kalkulatorach, miernikach cyfrowych). Taka forma zapisu umożliwia prostą konwersję do systemu dziesiętnego i z niego.

Przykładowo może być wykorzystywany zapis dwóch cyfr w jednym bajcie. Takie rozwiązanie nazywa się spakowanym kodem BCD (ang. *packed BCD*).

Arytmetyka z wykorzystaniem kodu BCD jest bardziej skomplikowana niż w kodzie NKB lub w innych kodach. Dlatego też działania realizowane w tym kodzie są wolniejsze niż np. w NKB. Rozróżnia się wiele wariantów kodu BCD. Dla uporządkowania wersja podstawowa zwana jest też BCD 8421 lub NBCD.

Zaletami kodu BCD są m.in.:

- bardzo intuicyjna interpretacja,
- bardzo prosta konwersja dziesiętne na kod dwójkowo-dziesiętny i z powrotem,

Wadą jest to, że istnieją dwa zera, czyli dwa różne sposoby reprezentowania wartości zero:

0 0000 0000 – zero dodatnie,

1 0000 0000 – zero ujemne.

Porównanie kodu BCD z dwójkowym**Zalety:**

- Wiele wartości niecałkowitych, takich jak dziesiętne 0,1, nie ma skończonej reprezentacji w postaci binarnej, ale ma skończoną reprezentację danej wartości w postaci dziesiętnej zakodowanej binarnie (0,0010). W konsekwencji system oparty na dziesiętnych reprezentacjach ułamków dziesiętnych zakodowanych

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

binarnie pozwala uniknąć błędów w reprezentowaniu i obliczaniu takich wartości. Jest to przydatne w obliczeniach finansowych.

- Skalowanie o potęgę 10 (mnożenie razy 10 lub dzielenie przez 10) jest bardzo proste.
- Zaokrąglanie na granicy cyfry dziesiętnej jest prostsze. Dodawanie i odejmowanie liczb dziesiętnych nie wymagają zaokrąglania.
- Wyrównanie dwóch liczb dziesiętnych (na przykład $1,3 + 27,08$) to proste, dokładne przesunięcie.
- Konwersja do postaci znakowej lub do wyświetlania (np. Do formatu tekstowego, takiego jak XML, lub do sterowania sygnałami dla wyświetlacza siedmiosegmentowego) jest prostym mapowaniem na cyfrę i można ją przeprowadzić liniowo.

Wady

- Niektóre działania są bardziej skomplikowane do wykonania niż w NKB. W dodatku wymagają dodatkowej logiki, aby wcześniej miały dopuszczalne wartości i generowały przeniesienie. Do dodania BCD potrzeba 15 do 20 procent więcej obwodów w porównaniu do czystego binarnego. Mnożenie wymaga użycia algorytmów, które są nieco bardziej złożone niż przesunięcie-maski-dodawanie.
- Standardowy BCD wymaga czterech bitów na cyfrę, około 20 procent więcej miejsca niż kodowanie binarne.
- Praktyczne istniejące implementacje BCD są zwykle wolniejsze niż działania na reprezentacjach binarnych.

Kod BCD 84-2-1**Zasady**

Jest to zmodyfikowany kod BCD. Wartości: 8, 4, (-2) i (-1) określają wagę poszczególnych pozycji. Liczbę w tym kodzie przedstawia się następująco:

$$5 = 1011_{\text{BCD } 84-2-1} = 8 - 2 - 1 = 1 \cdot 8 + 0 \cdot 4 + 1 \cdot (-2) + 1 \cdot (-1)$$

BCD 8421 a BCD 84-2-1**BCD 8421**

$$9 = 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 8 + 1$$

$$6 = 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 4 + 2$$

$$96 = 1001\ 0110_{\text{BCD}}$$

BCD 84-2-1

$$9 = 1 \cdot 8 + 1 \cdot 4 + 1 \cdot (-2) + 1 \cdot (-1) = 8 + 4 - 2 - 1$$

$$6 = 1 \cdot 8 + 0 \cdot 4 + 1 \cdot (-2) + 0 \cdot (-1) = 8 - 2$$

$$96 = 1111\ 1010_{\text{BCD } 84-2-1}$$

Kod Aikena

Kod Aikena (ang. *Aiken code*) (znany również jako kod 2421) jest podobny do kodu BCD. Grupa czterech bitów jest przypisana do cyfr dziesiętnych od 0 do 9 zgodnie z poniższą tabelą.

Kod Aikena różni się od standardowego kodu BCD 8421 tym, że kod Aikena nie waży czwartej cyfry jako 8, jak w przypadku standardowego kodu BCD, ale z 2.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Można by pomyśleć, że dla jednej liczby możliwe są podwójne kody, na przykład 1011 i 0101 może oznaczać 5. Jednak tutaj należy upewnić się, że cyfry od 0 do 4 są lustrzanym odbiciem uzupełniającym do liczb od 5 do 9.

Można inaczej powiedzieć, że wartości $5 \div 9$ to wartości uzupełnień dla liczb $0 \div 4$. Dzięki temu uzupełnienia w tym kodzie wylicza się zamieniając zera na jedynki i jedynki na zera.

Tabela. Wartości w kodzie Aikena

	3	2	1	0	bit
	2	4	2	1	waga
0					
1					
2					
3					
4					
nieużywane					
5					
6					
7					
8					
9					

https://en.wikipedia.org/wiki/Aiken_code

Kod EXCESS-3

Kod EXCESS-3 (Nadmiar 3, Plus 3, BCD+3) jest to kod typu BCD, jednak różni się od niego tym, że jest przesunięty wagowo o 3 względem kodu BCD 8421. Aby zapamiętać liczbę trzycyfrową np. 123 to do każdej cyfry dodaje się wartość 3, czyli otrzymuje się 456 i taką wartość konwertuje się tak, jak cyfry w kodzie BCD. Zatem otrzymuje się:

Liczba dziesiętna: 123
Przesunięcie o 3: 456 +3
Reprezentacja w tym kodzie: 0100 0101 0110

Liczby rozkodowuje się w odwrotny sposób:
Reprezentacja w tym kodzie: 1010 0111 1000
Liczba przesunięta o 3: 1078 -3
Liczba dziesiętna: 745

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Tabela. Liczby zakodowane w różnych kodach typu BCD

dziesiętne	binarne	BCD 8421	Kody odbite		
			BCD 84-2-1	EXCESS-3	Aiken
0	0000	0000	0000	0011	0000
1	0001	0001	0111	0100	0001
2	0010	0010	0110	0101	0010
3	0011	0011	0101	0110	0011
4	0100	0100	0100	0111	0100
5	0101	0101	1011	1000	1011
6	0110	0110	1010	1001	1100
7	0111	0111	1001	1010	1101
8	1000	1000	1000	1011	1110
9	1001	1001	1111	1100	1111

Cechy:

- Jest to nieważony binarny kod dziesiętny.
- Jest to kod samouzupełniający się. Słowa kodowe dla $0 \div 4$ są uzupełnieniami słów kodowych $5 \div 9$.
- Kody 0000 i 1111 nie są używane dla żadnej cyfry, co jest korzystne dla organizacji pamięci, ponieważ kody te mogą powodować awarię linii transmisyjnej.

<https://en.wikipedia.org/wiki/Excess-3>

Inne kody BCD

Innymi kodami typu BCD są: kod 3321, kod 4221, kod 5211.

Kod spolaryzowany

Kod spolaryzowany jest to kod z przesunięciem liczb względem kodu NKB. Nosi czasem nazwę kodu z obciążeniem (ang. *bias*), kodu z nadmiarem lub z offsetem binarnym (ang. *Offset binary*). Często jest to kod z przesunięciem o połowę zakresu. Liczba 0 (jest jedno zero) jest reprezentowana przez słowo n-bitowe 100...00, co odpowiada liczbie $2^{(n-1)}$. Czyli wartość zero jest reprezentowane przez najbardziej znaczący bit i zera na wszystkich pozostałych bitach.

Czyli liczba dodatnia jest wyliczana następująco:

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

gdzie:

n – liczba bitów

Wartość liczby z bitów rozkodowuje się w następujący sposób (dla liczby 8-bitowej)
 liczba = liczba jaką przedstawiałyby tak ustawione bity w kodzie NKB-128.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Liczba 17 przedstawiana na ośmiu bitach, w tym kodzie będzie miała wartość 10010001, czyli jeżeli potraktować to jako kod NKB to dziesiętnie byłaby to wartość: $128+17=145$.

Liczba (-17) będzie przedstawiana jako 01101111, w kodzie NKB będzie to liczba dziesiętna $128-17 = 107$

Podczas konwersji **liczby dodatniej** na kod spolaryzowany, **dodaje się jedynkę** dla najbardziej znaczącego bitu, czyli dla liczby 8-bitowej dodaje się (+128).

Liczbę ujemną przedstawia się w taki sposób, że od wartości dziesiętnej reprezentowanej przez najbardziej znaczący bit np. 128 odejmuje się wyznaczoną w tym kodzie wartość, czyli np. (-17). W wyniku wyjdzie wartość 111.

Większość standardowych procesorów komputerowych nie może obsłużyć bezpośrednio formatu offsetu binarnego.

Dla słowa n -bitowego przesunięcie wynosi $2^{(n-1)}$ to oznacza, że **pierwszy bit jest odwrócony względem kodu uzupełnienia do dwóch U2**.

Kod spolaryzowany, czy inaczej nazywane **przesunięcie binarne** może być przekształcone w kod uzupełnienia do dwóch U2 poprzez **odwrócenie najbardziej znaczącego bitu**.

Zmiennoprzecinkowy standard reprezentacji binarnej i wykonywania działań na liczbach zmiennoprzecinkowych IEEE 754 (ang. *IEEE floating-point standard*), który jest przedstawiony na kolejnych stronach, wykorzystuje kod spolaryzowany do przedstawiania wartości wykładnika. Może to być mylące, ponieważ zamiast przesunięcia $2^{(n-1)}$ w IEEE 754 jest wykorzystywane przesunięcie (ang. *offset, bias*) $2^{(n-1)}-1$.

W zależności od liczby bitów offset/przesunięcie może być różnie nazywane: *excess-15*, *excess-127*, *excess-1023*.

Porównanie liczb w różnych kodach

Liczba	NKB	ZM	U1	U2	BIAS	BCD
-127		11111111	10000000	10000001	00000001	1 0001 0010 0111
-126		11111110	10000001	10000010	00000010	1 0001 0010 0110
-1		10000001	11111110	11111111	01111111	1 0000 0000 0001
			11111111			
0	00000000	x0000000	00000000	00000000	10000000	x 0000 0000 0000
1	00000001	00000001	00000001	00000001	10000001	0 0000 0000 0001
2	00000010	00000010	00000010	00000010	10000010	0 0000 0000 0010
3	00000011	00000011	00000011	00000011	10000011	0 0000 0000 0011
4	00000100	00000100	00000100	00000100	10000100	0 0000 0000 0100
126	01111110	01111110	01111110	01111110	11111110	0 0001 0010 0110
127	01111111	01111111	01111111	01111111	11111111	0 0001 0010 0111

Różnica pomiędzy kodem U2 a kodem spolaryzowanym (BIAS) polega na odwróceniu najbardziej znaczącego bitu.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Własność samouzupełnienia i sekwencyjność kodów

Własność **samouzupełnienia** – kod odbity (ang. *reflective code*) – uzupełnienie $(p-1)$ -sze cyfr uzyskuje się w wyniku uzupełnienia jedynkowego każdego bitu kodu, czyli przez zanegowanie bitu.

Kod nazywamy **sekwencyjnym**, jeżeli każda kolejna liczba w tym kodzie jest o jedną liczbę binarną większa niż poprzedzająca ją liczba. To znacznie pomaga w wykonywaniu działań arytmetycznych. Przykładowo kody BCD 8421 i EXCESS-3 są kodami sekwencyjnymi.

Nie każdy kod BCD posiada takie własności.

Arytmetyka

Przeniesienie

Jeżeli dodaje się dwie jedynki na jednym polu w dwóch liczbach, to wtedy bit na tej pozycji w liczbie oznaczającej sumę ma wartość 0, dodatkowo uzyskuje się tzw. **przeniesienie C** (ang. *carry*).

Dodawanie zatem jest identyczne w tych wszystkich kodach. Dlatego też przy dodawaniu kolejnej, bardziej znaczącej pozycji w liczbie sumuje się dwa odpowiadające pola z dwóch liczb, dodatkowo dodaje się wartość 1 z przeniesienia z młodszego bitu – z mniej znaczącej pozycji.

Czasami sumowanie liczb z jednej pozycji powoduje przeniesienie do kolejnej pozycji, a z niej do kolejnej. W takich przypadkach mówi się o propagacji przeniesienia przez np. cztery pozycje.

Liczbę dodatnie zapisane w kodach U1, U2, ZM reprezentowane są **analogicznie, jak w kodzie NKB**.

Sumaryczna wartość bitu w wyniku zależy nie tylko od wartości bitów w sumowanych składnikach na tej pozycji, ale również od ewentualnego przeniesienia wynikającego z sumowania liczb mniej znaczących pozycji.

Przeniesienia

			0	1			
	+	+	+	+	+	+	+
			1	1			
	=	=	=	=	=	=	=
			1+1	0			
		1	0	0			

Kiedy na **najwyższej pozycji sumuje się co najmniej dwie jedynki**, uzyskuje się tzw. **przeniesienie wychodzące**, czyli wartość, której nie można przedstawić w danej reprezentacji liczb. Zatem sumowanie dwóch liczb o danej liczbie bitów może dać wynik wykraczający poza zakres wartości, mogących być reprezentowanych w danym kodzie na określonej liczbie bitów. Wtedy powstaje **nadmiar** (ang. *overflow*).

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

	0	1						
	+	+	+		+	+	+	+
	1	1						
	=	=	=		=	=	=	=
	1+1	0						
1	0	0						

Przepełnienie, nadmiar arytmetyczny (ang. *overflow, arithmetic overflow*) jest to sytuacja, kiedy wartość wyniku działania jest większa niż możliwości zaprezentowania jej w przypisanej docelowo ilości pamięci lub przekracza wielkość rejestru. W takiej sytuacji niezależnie od sposobu przedstawienia prezentowany będzie **nieprawidłowy wynik**, ale w taki sam sposób, jak wynik prawidłowy. Generalnie w omawianych kodach **brak jest bitu, który pokazuje wiarygodność wyniku**.

Mogą występować dwie sytuacje nadmiaru:

- podczas dodawania lub odejmowania otrzymany wynik jest większy, niż największy możliwy do przedstawienia na przypisanej liczbie bitów; w takich sytuacjach dobrze jest sprawdzać flagę przeniesienia (ang. *carry flag*) – zwana flagą C;
- wynik ma inny znak niż możliwy do otrzymania w wyniku określonego działania np. po dodaniu dwóch liczb dodatnich, otrzymuje się wynik ujemny. W takiej sytuacji należy sprawdzać flagę przepełnienia (ang. *overflow flag*) – czasem nazywana flagą V.

Dopisywanie nieznaczących cyfr

Jeżeli daną liczbę zapisuje się przy użyciu większej liczby cyfr, niż ona zawiera, to dopisuje się określone cyfry z przodu części całkowitej liczby lub z tyłu części ułamkowej. To, jakie konkretnie cyfry dopisuje się do liczby zależy od tego, w jaki sposób dana liczba jest kodowana, aby nie zmienić jej wartości.

W taki sposób dopisuje się:

- Zera dla liczb kodowanych naturalnie w NKB oraz dodatnich liczb kodowanych w U1, U2 i BCD,
- Jedynki dla liczb ujemnych kodowanych w U1,
- Jedynki z przodu oraz zera z tyłu dla liczb ujemnych kodowanych w U2,
- Dla liczb kodowanych w kodzie EXCESS-3 zerem jest wartość 0011,
- Ogólna zasada: dla liczb kodowanych z użyciem uzupełnień U_p lub U_{p-1} , w sposób analogiczny jak dla liczb w kodzie U1 i U2, dopisuje się wartość 0 lub podstawę minus 1, czyli $p-1$.

Działania na liczbach binarnych

W wyniku wykonywania działań na liczbach binarnych wynik działania może mieć inną liczbę bitów niż liczby

- dodawanie:** liczba ośmiobitowa plus liczba ośmiobitowa – wynik może mieć maksymalnie **8+1** (np. przeniesienie) bitów ($255+255 = 510$). W ogólności

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

wynik dodawania może mieć o jeden bit więcej niż składnik dodawania dwóch liczb, która ma więcej bitów od drugiej.

- **odejmowanie:** liczba ośmiobitowa minus liczba **ośmiobitowa** – wynik może mieć maksymalnie **8** bitów (np. **254** – **0** = 254). W ogólności wynik odejmowania dwóch liczb może mieć tyle bitów, ile ma składnik odejmowania o większej liczbie bitów np. $0 - 257 = (-257)$
- **mnożenie:** liczba **ośmiobitowa** razy liczba **ośmiobitowa** – wynik może mieć maksymalnie **8+8** bitów (**255**·**255** = 65025). W ogólności wynik mnożenia może mieć tyle bitów, ile wynosi suma bitów czynników mnożenia.
- **dzielenie:** liczba szesnastobitowa podzielić przez liczbę ośmiobitową – wynik może mieć maksymalnie **16** bitów (**65535**/**1** = 65535). W przypadku działań na liczbach całkowitych wynik dzielenia dwóch liczb będzie miał tyle bitów, ile dzielną. W przypadku liczb rzeczywistych wynikiem może być ułamek okresowy lub mający nieskończoną liczbę cyfr.

Negowanie liczb stałopozycyjnych

Negowanie w kodzie znak-moduł

Do danej liczby należy dodać liczbę zawierającą ustawiony jedynie najstarszy bit (czyli ten, odpowiadający za znak).

Przykład

0 0000101 – dziesiętne 5
+ **1** 0000000 – współczynnik do zmiany znaku na przeciwny (128 dla 8 bitów)
= **1** 0000101 – czyli wartość przeciwną (-5)

Analogicznie

1 0000101 – dziesiętne (-5)
+ **1** 0000000 – współczynnik do zmiany znaku na przeciwny (128 dla 8 bitów)
= **0** 0000101 – czyli wartość przeciwną (5) – (+ bit przepełnienia, który w tym przypadku nie jest wykorzystywany).

Negowanie w kodzie U1

Najprostsze negowanie liczb w kodzie U1, czyli zamienienie ich znaku, wykonuje się przez zamianę wszystkich zer na jedyne, a jedynek na zera.

0 000 1110 – liczba 14
1 111 0001 – liczba (-14)

I odwrotnie

1 001 0001 – liczba (-110)
0 110 1110 – liczba 110

Inny sposób realizacji działania arytmetycznego polegającego na zmianie znaku na przeciwny w kodzie U1:

Daną liczbę odejmujemy od samych jedynek (dla 8 bitów będzie to wartość 255)

1 111 1111 – wartość do zmiany znaku na przeciwny dla kodu U1
- **0** 000 1101 – liczba, której wartość przeciwną chcemy otrzymać 13
= **1** 111 0010 – otrzymana liczba przeciwna (-13)

I odwrotnie

1 111 1111 – wartość do zmiany znaku na przeciwny dla kodu U1

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

- 1 001 0001 – liczba (-110)

= 0 110 1110 – otrzymana liczba przeciwna 110

Negowanie liczb w kodzie U2

Działanie polegające na zmianie znaku liczby w kodzie U2 można zrealizować również na kilka sposobów.

Pierwsza metoda

Znajdź w liczbie **pierwszą jedynekę od prawej strony**, a następnie obróć następujące po niej wartości (zamień jedynki na zera, a zera na jedynki) we wszystkich cyfrach na lewo od tej jedynki).

Liczba (+12) – 00001**1**00

Liczba (-12) – **1**1110**1**00

Liczba (+112) – 0**1**1**1**0000

Liczba (-112) – **1**00**1**0000

(czyli wartość przeciwna do dodatniej 1000111 + 0000001)

Druga metoda negacji liczb:

Odwróć wszystkie bity, a następnie **dodaj wartość jeden** – czyli 00000001

Liczba (+14) – 00001110

Liczba (-14) – 11110001 + 00000001 = 11110010

I w drugą stronę

Liczba (-14) – 11110010

Liczba (+14) – 00001101 + 00000001 = 00001110

Negowanie liczb w kodzie spolaryzowanym

Negowanie liczb w kodzie spolaryzowanym odbywa się przez zamianę wszystkich bitów 0 na 1 i 1 na 0, oraz dodanie liczby 1, czyli 00000001

Przykład

1 000 0101 – liczba 5

0 111 1011 – liczba (-5)

Analogicznie negowanie wykonuje się w drugą stronę.

Dodawanie i odejmowanie liczb stałopozycyjnych

Różne rodzaje działań na liczbach

1) **Odejmowanie**/Pomniejszanie wyniku z możliwością przejścia przez zero:

1. Odejmowanie liczby dodatniej od innej liczby dodatniej,
2. Odejmowanie liczby ujemnej od innej liczby ujemnej,
3. Dodawanie liczby ujemnej i dodatniej (i odwrotnie, bo jest przemienność dodawania),

2) **Dodawanie**/sumowanie liczb z możliwością przekroczenia zakresu dla liczb przez otrzymany moduł wynikowy:

- a) Dodawanie do siebie dwóch liczb dodatnich,
- b) Dodawanie do siebie dwóch liczb ujemnych,
- c) Odejmowanie od liczby ujemnej liczby dodatniej,

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

1 + 1 = 10 → 0 + przeniesienie 1 do następnej kolumny/pozycji po lewej stronie
To tak jak w liczbach dziesiętnych
5 + 5 = 10 → 0 + przeniesienie 1 do następnej kolumny/pozycji po lewej stronie
To nie jest tak jak suma logiczna. Kiedy 2 bity na jednej pozycji mają wartość 1, to sumarycznie w tej pozycji będzie 0!

	ZM, U1, U2	BCD
+35	0 0100011	0 0011 0101
+51	0 0110011	0 0101 0001
+86	0 1010110	0 1000 0110

W tym przypadku, dla kodu BCD jako wynik sumowania na obu tetradach pojawiły się wartości należące do kodu BCD.

	ZM, U1, U2	BCD
+90	0 1011010	0 1001 0000
+42	0 0101010	0 0100 0010
+132	0 0000011	0 1101 0001
Przeniesienie	1	

Występuje problem z przedstawieniem wyniku. W kodzie ZM, U1, U2 widać, że jest za mało bitów, aby możliwe było przedstawienie wyniku dodawania. Dla kodu BCD jako wynik sumowania na drugiej tetradzie pojawiła się wartość, która nie należy do kodu BCD. Wartość na drugiej tetradzie nie jest liczbą dziesiętną. Konieczny jest krok korekcyjny – do liczby dodaje się 6, czyli binarnie 0110, jest to liczba niewykorzystywanych kombinacji zer i jedynek w tym kodzie. Jednak w takiej reprezentacji liczby w kodzie BCD nie ma miejsca na trzecią cyfrę wyniku.

Taka korekta wyniku jest konieczna. Proces korygowania w tym kodzie jest wielostopniowy.

Dodawanie liczb +90 i +42 nie dało pozytywnego wyniku, nawet jeżeli wiadomo, że wynik można byłoby przedstawić na 8 bitach. Jednak w rozpatrywanych kodach jeden bit odpowiada za znak. W konsekwencji otrzymany wynik jest większy niż największy możliwy do przedstawienia na 7 bitach (0 ÷ 127) dostępnych dla reprezentowania cyfr. Wynik wykracza poza przyjętą liczbę bitów.

Prawidłowy wynik można otrzymać wykorzystując większą liczbę bitów. To pokazuje, że projektując układy arytmetyczne należy uwzględnić przypadki, jakie działania będą wykonywane, na jakich liczbach i jaki będzie docelowy zakres wykorzystywanych wartości.

Działanie może zostać wykonane prawidłowo przy wykorzystaniu np. 9 bitów dla kodów ZM, U1, U2 oraz 13 bitów dla kodu BCD.



„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

	ZM, U1, U2	BCD
+90	0 001011010	0 0000 1001 0000
+42	0 000101010	0 0000 0100 0010
		0 0000 1101 0010
Korekta dla kodu BCD (+6)		110 .
Przeniesienie	1 .	1 .
Wynik +132	0 010000100	0 0001 0011 0010

Jak dodać dwie liczby?

999976755 (0x3B9A6F33) i 1235456789 (0x49A39315)

	999976755	1235456789	suma	2235433544		przeniesienie
	3B9A6F33	49A39315		85330248		
szesnastkowo	33	15	48	48	48	
dziesiętnie	51	21	72	72	72	0
szesnastkowo	6F	93	102	02	02	
dziesiętnie	111	147	258	2	2	1
szesnastkowo	9A	A3	133	33	33	
dziesiętnie	154	163	317	51	52	1
szesnastkowo	3B	49	84	84	85	
dziesiętnie	59	73	132	132	133	0 Flaga przen.

Uprozczone dodawanie długich ciągów liczb

$$\begin{array}{r}
 1000011111101011111 \\
 + \\
 101010101010101010 \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 10000\overset{1}{\text{11111}}1010\overset{1}{\text{11111}}1 \\
 + \\
 1010101010\overset{1}{\text{10101010}}\overset{1}{\text{10}} \\
 \hline
 1011\overset{0}{\text{001010010}}\overset{1}{\text{101001}}
 \end{array}$$

Odejmowanie liczb stałopozycyjnych

Jako odejmowanie będziemy też rozumieć wykonywanie działań na liczbach ujemnych np.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

- Dodawanie do liczby dodatniej liczby ujemnej,
- Dodawanie do liczby ujemnej innej liczby ujemnej,
- Wykonywanie działania odejmowania od liczby dodatniej innej liczby dodatniej.

$$1 - 1 = 0$$

$$0 - 0 = 0$$

$$1 - 0 = 1$$

$0 - 1 = (-1)$ albo 1, tylko, że wtedy konieczna jest pożyczka z bardziej znaczącego bitu.

W dodawaniu występuje ryzyko, że uzyskany wynik wykroczy poza dostępny zakres liczb możliwych do reprezentowania.

W odejmowaniu występuje ryzyko, że konieczne będzie pożyczanie wartości spoza zakresu, czyli że odjemnik będzie większy, niż odjemna.

Najprostsze odejmowanie to takie, bez pożyczek, czyli bez odejmowania 1 od 0.

Pożyczka w odejmowaniu liczb binarnych

W celu wyliczenia wartości liczby w kodzie binarnym, poszczególne cyfry na poszczególnych pozycjach mnożone są razy wagę tych pozycji.

7	6	5	4	3	2	1	0	-1	-2	-3	indeks
1	1	1	1	1	0	1	1	1	0	1	
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	Waga pozycji
10.000.	1.000.	100.	10.	1.	100	10	1	0,1	0,01	0,001	Mnożnik cyfry na danej pozycji
000	000	000	000	000							
128	64	32	16	8	4	2	1	0,5	0,25	0,125	Mnożnik dziesiętny cyfry na danej pozycji

Nieco upraszczając można napisać: $X = 8_{10} = 1000_2 = 1 \cdot 2^3 + 0 \cdot 2^2 = 0 \cdot 2^3 + 2 \cdot 2^2$

Ponieważ wiadomo, że $2^3 = 2 \cdot 2^2$

Czyli tak, jakbyśmy zapisali tę liczbę w postaci (nieprawidłowej, ale dla celów odejmowania z pożyczką) jako $X = 0200_2$

		012001121	Zastępujemy ciąg 100 jako 012, a ciąg 1000 jako 0112.
273		100010001	
71-		001000111	
wynik 202		011001010	

Najprostsze odejmowanie w kodzie ZM

00001111 (dziesiętnie 15)

- 00000111 (dziesiętnie 7)

= 00001000 (dziesiętnie 8)

Jest to przykład odejmowania bez konieczności pożyczania. Wynik odejmowania ma taki znak, jaki miała liczba o większym module, czyli taki jak odjemna.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Odejmowanie dwóch liczb np. $40 - 20 = ?$

to działanie sumowania $40 + (-20)$.

Porównanie modułów liczb pokazuje znak wyniku.

$|40| > |-20|$ – moduł liczby dodatniej jest większy, zatem wynik (+)

Sens pożyczki – kod ZM

$\boxed{1}00 = \boxed{0}20$
 $-010 = -010$
 $010 \quad 010$

Przykład 113 – $87 = ?$

Odejmowanie wykonuje się w analogiczny sposób, jak w przypadku odejmowania liczb dziesiętnych. Najlepiej zacząć od najmniej znaczących bitów/cyfr. Czasem konieczne jest odejmowanie z pożyczką wziętą z bitów na bardziej znaczących pozycjach. Bit znajdujący się na bardziej znaczącej pozycji ma wartość dwa razy większą niż ten na sąsiedniej, niższej.

113	01110001	01021121
87	01010111	01010111
wynik (-)	00011010	00011010

W zależności od znaku dodaje się moduły (np. kiedy od ujemnej odejmuje się dodatnią, od ujemnej ujemną itp.) albo odejmuje (dodatnia odjąć dodatnia, ujemna dodać dodatnia itp.).

Na podstawie **wielkości modułów** określa się, jaki **znak** będzie miał **wynik** odejmowania, **wynik odejmowania = znak liczby o większym module**,

W przypadku odejmowania modułów od większego modułu odejmuje się mniejszy.

Osobno otrzymuje się wynik znaku, osobno wartość modułu.

Odejmowanie w kodzie ZM ze zmianą znaku/ przejściem przez zero

$+ 0000111 \quad (7_{10})$
 $- 0000111 \quad (15_{10})$
 $- 00001000 \quad (8_{10})$ – taki powinien być wynik

Wynik ma taki sam znak, jak odjemna.

Kłopotliwe opracowanie algorytmów realizujących takie działania w sposób automatyczny.

Odejmowanie w kodzie U1

Niewątpliwą zaletą kodu U1 i U2 jest łatwość wykonywania działań arytmetycznych w tym kodzie. Bit odpowiedzialny za znak liczby nie musi być traktowany w sposób szczególny, jak to jest przy notacji Znak-moduł.

Po prostu może być dodawany, odejmowany podczas wykonywania działań na liczbach. Jeżeli tylko wynik działań arytmetycznych nie wykracza poza dostępny, dla danej liczby bitów, zakres dostępnych wartości, to wszystkie działania w efekcie dają prawidłowy wynik.


„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Zadanie.


Wyliczyć $(-16) + 7 = ?$, przy czym wykorzystać dodawanie uzupełnienia U1 liczby (-16) .

11101111 (-16) – odejmowanie dodatniej
 $-$ **00001111** (7) – w takiej sytuacji **odejmujemy** bity
 $=$ **11101000** – przeciwna wartość do liczby 00010111_2 – czyli do 23_{10} .
 Nie ma bitu przeniesienia, więc nie ma potrzeby dokonywania korekty wyniku.

Zadanie. Wyliczyć $8 - 3 = ?$, wykorzystując uzupełnienie U1 liczby (-3)

00001000 (8) – w takiej sytuacji – dodajemy bity
 $+$ **11111100** (-3) – (dodatnia + ujemna i wynik dodatni)
(1) 00000100 (4) – konieczna korekta
 **1** – korekta
00000101 (5) – prawidłowy wynik

Zamiana kolejności, czyli wykonywanie działań $-3+8$ nie zmienia wyniku

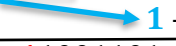
11111100 (-3) – dodajemy bity (ujemna i dodatnia)
00001000 (8)
(1) 00000100 (4) – konieczna korekta – bo jest bit przeniesienia
 **1** – korekta
00000101 (5) – prawidłowy wynik

Poważnym problemem przy działaniach odejmowania w kodzie U1 jest konieczność dokonywania korekty wyniku. Komplikuje i wydłuża czas wykonywania działania odejmowania. Co więcej, jeżeli takie przeniesienie będzie dalej przenoszone przez kolejne bity (bo będą one miały wartość 1) to wtedy czas wykonywania działania może wydłużyć się prawie dwukrotnie.

Dodawanie dwóch liczb ujemnych w kodzie U1


Wynik działania na liczbach mieszczący się w dopuszczalnym zakresie.

Zadanie. Należy dodać liczby $(-20) + (-30) = ?$

11101011 (-20)
 $+$ **11100001** (-30) – dodawanie bitów (również bitów wskazujących znak)
(1) 11001100
 **1** – bit przeniesienia
11001101 $(-50; \text{binarnie } 110010)$

Prawidłowy wynik po korekcji

Zadanie. Należy dodać liczby $(-80) + (-48) = ?$

10101111 (-80)
 $+$ **11001111** (-48) – dodawanie bitów
(1) 01111110 $(+126)$
 **1** – bit przeniesienia,
01111111 $(+127)$

Algorytm nieprawidłowo liczy, jeżeli wynik wykracza poza dopuszczalny zakres.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład wykonywania działań arytmetycznych na liczbach binarnych

Liczba A:

$$A = 14 \frac{23}{32}$$

Liczba B:

$$B = 22 \frac{29}{32}$$

Wykonać działanie $A - B = ?$, wykorzystując uzupełnienie U1 liczby B.

Dwie liczby	
01110,10111	Liczba A
10110,11101	Liczba B
01001,00010	1 tworzenie uzupełnienia U1 liczby B (zamiana zer na jedynek i jedynek na zera)
01110,10111 01001,00010	2. Sumowanie liczby A oraz U1 liczby B
10111,11001	Suma
01000,00110	Jest to liczba ujemna, więc 3. rozkodowujemy U1 zamieniając zera na jedynek i jedynek na zera Otrzymano wynik $wynik = 8 \frac{6}{32}$

Odejmowanie w kodzie U2

$$\begin{array}{rcl}
 \textcolor{red}{1}1110000 \text{ (-16)} & - & \text{odejmowanie dodatniej} \\
 - \textcolor{red}{0}0000111 \text{ (7)} & - & \text{w takiej sytuacji odejmujemy bity} \\
 \hline
 = \textcolor{red}{1}1101001 & - & \text{wartość przeciwna do 00010111 (+ 1 bit najmniej znaczący), czyli} \\
 \text{(-23)} & &
 \end{array}$$

Dodawanie liczby ujemnej w kodzie U2

$$16 + (-7) = ?$$

$$\begin{array}{rcl}
 \textcolor{red}{0}0010000 \text{ (+16)} & & \\
 + \textcolor{red}{1}1111001 \text{ (-7)} & - & \text{dodawanie bitów} \\
 \hline
 = \textcolor{red}{0}0001001 \text{ (9)} & - & \text{wartość prawidłowa}
 \end{array}$$

$$7 + (-16) = ?$$

$$\begin{array}{rcl}
 \textcolor{red}{0}0000111 \text{ (+7)} & & \\
 + \textcolor{red}{1}1110000 \text{ (-16)} & - & \text{dodawanie bitów} \\
 \hline
 = \textcolor{red}{1}1110111 \text{ (-9)} & - & \text{bo po zamianie na przeciwne jest 00001000 + 1, czyli przy} \\
 \text{konwersji na dodatnie, to jest (9)} & &
 \end{array}$$

$$(-80) + (-49) = ?$$

$$\begin{array}{rcl}
 \textcolor{red}{1}0110000 \text{ (-80)} & & \\
 + \textcolor{red}{1}1001111 \text{ (-49)} & - & \text{dodawanie bitów} \\
 \hline
 \textcolor{red}{0}1111111 \text{ (+127 lub)} & & \\
 \textcolor{blue}{1} & - & \text{bit przeniesienia?}
 \end{array}$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Do liczb dodatnich nie dodaje się bitu przeniesienia, aby zwiększyć liczbę binarną o 1. Z sumowania dwóch liczb ujemnych wyszła wartość dodatnia. Algorytm nieprawidłowo liczy, jeżeli wynik wykracza poza dopuszczalny zakres.

W przypadku takich przypadków konieczne jest zwiększenie liczby bitów, na których będą wykonywane działania arytmetyczne. Konieczny jest bit sygnalizujący nieprawidłowy wynik jako skutek wystąpienia nadmiaru. Aby wykryć powstanie nadmiaru należy porównać znak składników sumowania ze znakiem wyniku.

Inne podejście do wykorzystania uzupełnień

W Internecie można znaleźć trochę ograniczoną i nieco zmodyfikowaną metodę wykorzystania uzupełnienia liczby odejmowanej. Prezentowane tam algorytmy ograniczają się do zastąpienia wykonywania $A - B$, jako $A + (-B)$, przy założeniu, że A i B są liczbami dodatnimi oraz wykorzystujemy w niej uzupełnienie liczby B do przedstawienia odejmowania.

$A - B = A + (-B) = A + \bar{B}$ – przy wykorzystaniu $(p-1)$ -szego uzupełnienia albo

$A - B = A + (-B) = A + \bar{\bar{B}}$ – przy wykorzystaniu p -tego uzupełnienia

Przeniesienie/przepełnienie można pominąć

Jeżeli odejmuje się jedną liczbę dodatnią od drugiej liczby dodatniej to moduł wyniku będzie co najwyżej równy modułowi tej liczby, która ma większy moduł.

Zatem przeniesienie poza cyfry znaczące można pominąć i nie trzeba tworzyć nowej, najbardziej znaczącej pozycji dla cyfry w tej liczbie, ponieważ to przeniesienie nie niesie za sobą istotnej informacji co do wartości modułu.

Jeżeli w wyniku dodawania występuje przepełnienie (przeniesienie poza możliwe cyfry do zapisania), oznacza to, że wynik jest liczbą dodatnią.

Jeżeli natomiast nie występuje takie przeniesienie, to wtedy wynikiem jest liczba ujemna zapisana jako $(p-1)$ -sze uzupełnienie.

Metoda wykorzystania $(p-1)$ -szego uzupełnienia

Metoda wykorzystania $(p-1)$ -szego uzupełnienia polega na wykonaniu 3 kroków:

1. Wykonuje się uzupełnienie $(p-1)$ -sze liczby B ,
2. Wykonuje się dodawanie $A + \bar{B}$
3. Jeżeli w wyniku dodawania występuje przepełnienie (przeniesienie poza możliwe cyfry do zapisania), wtedy to przeniesienie dodawane jest do wyniku, oraz oznacza to, że wynik jest liczbą dodatnią; jeżeli natomiast nie występuje takie przeniesienie, to wtedy wynikiem jest liczba ujemna zapisana jako $(p-1)$ -sze uzupełnienie.

Przykład 1.

$$A = 12_{10} = 1100_2$$

$$B = 7_{10} = 0111_2$$

$$\bar{B} = 1000_2$$

Wykonujemy działanie:

$$A - B = A + (-B) = A + \bar{B}$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

$$\begin{array}{rcl} A = 12_{10} & & 1100_2 \\ \bar{B} = & & 1000_2 \\ A + \bar{B} & & {}^10100_2, \text{ gdzie } {}^1 - \text{oznacza przeniesienie i to, że wynik jest dodatni} \\ & & + 1 \\ A + \bar{B} & & 0101_2, \text{ wynik końcowy} = 5 \end{array}$$

Przykład 2.

$$A = 7_{10} = 0111_2$$

$$B = 12_{10} = 1100_2$$

$$\bar{B} = 0011_2$$

Wykonujemy działanie:

$$A - B = A + (-B) = A + \bar{B}$$

$$\begin{array}{rcl} A = 7_{10} & & 0111_2 \\ \bar{B} = & & 0011_2 \\ A + \bar{B} & & 1010_2, \text{ gdzie brak przeniesienia oznacza to, że wynik jest ujemny } (-5) \\ |A + \bar{B}| & & 0101_2, \text{ moduł wyniku końcowego} = 5 \end{array}$$

Metoda wykorzystania (p)-tego uzupełnienia

Metoda wykorzystania (p)-tego uzupełnienia polega na wykonaniu 3 kroków:

1. Wykonuje się uzupełnienie (p)-te liczby B,
2. Wykonuje się dodawanie $A + \bar{B}$
3. Jeżeli w wyniku dodawania występuje przepełnienie (przeniesienie poza możliwe cyfry do zapisania), wtedy oznacza to, że wynik jest liczbą dodatnią. jeżeli natomiast nie występuje takie przeniesienie, to wtedy wynikiem jest liczba ujemna zapisana jako (p)-te uzupełnienie.

Przykład 1.

$$A = 12_{10} = 1100_2$$

$$B = 7_{10} = 0111_2$$

$$\bar{B} = 1001_2$$

Wykonujemy działanie:

$$A - B = A + (-B) = A + \bar{B}$$

$$\begin{array}{rcl} A = 12_{10} & & 1100_2 \\ \bar{B} = & & 1001_2 \\ A + \bar{B} & & {}^10101_2, \text{ gdzie } {}^1 - \text{oznacza przeniesienie i to, że wynik jest dodatni} = 5 \end{array}$$

W przypadku wykorzystania (p)-te uzupełnienia przeniesienia nie dodaje się do wyniku końcowego.

Przykład 2.

$$A = 7_{10} = 0111_2$$

$$B = 12_{10} = 1100_2$$

$$\bar{B} = 0100_2$$

Wykonujemy działanie:

$$A - B = A + (-B) = A + \bar{B}$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

$$\begin{aligned} A &= 7_{10} && 0111_2 \\ \bar{\bar{B}} &= && 0100_2 \\ A + \bar{\bar{B}} &&& 1011_2, \text{ gdzie brak przeniesienia oznacza to, że wynik jest ujemny } (-5) \\ |A + \bar{\bar{B}}| &&& 0101_2, \text{ moduł wyniku końcowego } = 5 \end{aligned}$$

Działania arytmetyczne: dodawanie i odejmowanie

Jeżeli są dwie liczby to zasadniczo można dodać lub odjąć ich moduły. W zależności od tego, jakie znaki one posiadają i jaki znak posiada liczba.

Przy odejmowaniu liczby ujemnej można znak odjąć odejmowania i znak minus liczby ujemnej zamienić na jeden znak plus.

W przypadku dodawania dwóch liczb dodatnich lub dwóch ujemnych (lub odejmowania dodatniej od ujemnej), czyli dodawaniu dwóch liczb posiadających ten sam znak modułu, wykonuje się jedynie działanie dodawania modułów, znak pozostawia się taki sam, jaki był przy tych dwóch liczbach.

W przypadku dodawania liczby ujemnej do dodatniej lub odejmowania liczby dodatniej od dodatniej wykorzystuje się odpowiednie uzupełnienie, aby przedstawić liczbę ze znakiem minus.

Następnie wykonuje się działanie jak powyżej. Jeżeli jest przeniesienie to znaczy, że wynik jest dodatni, a jeżeli nie występuje przeniesienie – wynik jest ujemny.

Mnożenie liczb stałopozycyjnych – całkowitych**Informacje o mnożeniu liczb:**

W systemie binarnym, analogicznie jak w systemie dziesiętnym, liczby mnoży się w tylu krokach, z ilu bitów składają się składniki. Przykładowo, jeżeli mnoży się jedną liczbę przez drugą – czterocyfrową, to mnożenie wykonuje się w 4 krokach, mnożąc przez każdą cyfrę z kolejnych pozycji. Mnożąc dwie liczby czterocyfrowe wynik może być ośmiocyfrowy. Analogicznie mnożąc dwie liczby ośmiobitowe, wynik może wyjść szesnastobitowy.

Mnożenie pisemne polega na tym, że mnożoną liczbę mnoży się przez kolejne liczby drugiego czynnika mnożenia, otrzymując iloczyn cząstkowy, zapisywany z odpowiednim przesunięciem, adekwatnym do pozycji cyfry mnożnika w tym czynniku mnożenia. Po wykonaniu wszystkich działań mnożenia sumuje się wyniki uzyskując kolejne iloczyny cząstkowe.

W przypadku liczb binarnych sprawa jest o tyle prosta, że mnoży się razy 0 lub razy 1, czyli do iloczynu cząstkowego dodaje się przesuniętą wartość mnożoną lub zero.

Podstawy mnożenia binarnego

$$0 \cdot 0 = 0$$

$$1 \cdot 0 = 0$$

$$0 \cdot 1 = 0$$

$$1 \cdot 1 = 1$$

Aby wartość iloczynu była różna niż zero muszą spotkać się dwie jedynki.

W przypadku mnożenia liczb ze znakiem,

$$\text{cyfra znaku wynosi} = (\text{cyfra_znaku_1_liczby} + \text{cyfra_znaku_2_liczby}) \bmod 2$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Podstawy mnożenia binarnego

W metodzie mnożenia binarnego wykorzystywane są następujące funkcje, możliwe do łatwej implementacji programowej lub sprzętowej:

- 1) Przesuwanie bitów w lewo – mnożenie $\cdot 2$;
- 2) Przesuwanie bitów w prawo – dzielenie przez dwa;
- 3) Sumowanie dwóch wartości.

Założenia są następujące:

- Algorytm powinien być „bezmyślny” – czyli powinien wykonywać cyklicznie pewne kroki, które niezależnie od wartości mnożonych liczb zwrócą prawidłową wartość iloczynu;
- Algorytm powinien mieć w sobie jak najmniej warunków typu: jeżeli znak mnożonej liczby, jeżeli dwa znaki mnożonych wartości itp. Warunki mogą natomiast być uzależnione od działania algorytmu np. jeżeli występuje przeniesienie, jeżeli dwa bity są sobie równe itp.

Mnożenie binarne – metody bezpośrednie

Reguły ogólne

- Mnożenie zgodnie z metodami bezpośrednimi polega na tym, że w kolejnych krokach wylicza się iloczyny cząstkowe.
- Mnożenie można rozpocząć od najmłodszego bitu (najmniej znaczącego bitu) mnożnika, aż do najstarszego (najbardziej znaczącego bitu). W takim przypadku w kolejnych krokach stosujemy narastającą wartość – każdorazowo mnożoną $\cdot 2$.
- Jeżeli mnożenie rozpoczyna się od najstarszego bitu mnożnika aż do najmłodszego, wtedy wykorzystuje się dzielenie przez 2 – przesuwanie w prawą stronę.
- Ze względu na przemienność w mnożeniu i dodawaniu to, czy rozpoczyna się mnożenie od najstarszego bitu mnożnika, czy on najmniejszego nie ma większego znaczenia.
- W metodzie bezpośredniej I mnożona/dzielona, czyli przesuwana jest mnożna, natomiast w metodzie bezpośredniej II – wyliczony iloczyn cząstkowy.

Mnożenie binarne (ang. *Binary Multiplication*)

$$\begin{array}{r}
 1100 (12_{10}) \\
 \cdot 1011 (11_{10}) \\
 \hline
 1100 \\
 11000 \\
 000 \\
 1100000 \\
 \hline
 111 \text{przeniesienie} \\
 10000100 (132_{10})
 \end{array}$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Dopisujemy mnożne, które w każdym cyklu są przesunięte o jedną pozycję (wartość pomnożona razy 2), jeżeli bit mnożnika jest równy 1.

Mnożenie binarne – metoda bezpośrednia I

W tej metodzie, za każdym krokiem przesuwamy o jedną cyfrę ($\cdot 2$) mnożną, która w kolejnych krokach jest dodawana do wyniku.

Algorytm mnożenia binarnego

W urządzeniach cyfrowych zwykle wykorzystywane są sumatory dwuelementowe, dlatego konieczne jest zmodyfikowanie algorytmu mnożenia, aby nie sumować wielu elementów. Dlatego zwykle jak otrzymuje się kolejny iloczyn cząstkowy, jest on dodawany do sumy. Zatem w każdym kroku wykonywanym w algorytmie mnożenia będą wykonywane dwa działania: przesunięcia iloczyn cząstkowego o jedną pozycję w prawo i dodawania.

Jednym elementem sumowania będzie bieżący produkt/iloczyn cząstkowy. Jeżeli brany do mnożenia bit mnożnika ma wartość 1, to do iloczyn cząstkowego dodawana jest liczba mnożona z odpowiednim przesunięciem. Natomiast jeżeli brany do mnożenia bit mnożnika ma wartość 0, to składnik, który należałoby dodać również będzie równy 0.

Mnożenie binarne – metoda bezpośrednia I

W dużym uproszczeniu można powiedzieć, że metoda ta polega na:

- 1) Dodawaniu do iloczynu cząstkowego wartości **mnożnej, przesuniętej** o tyle pozycji w lewo, na której znajduje się cyfra mnożnika, jeżeli kolejna, rozpoczynając od końca (od najmniej znaczącego bitu cyfra mnożnika jest równa 1 lub dodawaniu wartości zera. **W tej metodzie mnożna przesuwana jest w lewo.**

12	1100	mnożna
· 14	x1110	mnożnik
<hr/>	0000	iloczyn cząstkowy
168	+ 0000	mnożna
	<hr/>	
	00000	iloczyn cząstkowy
	+ 11000	mnożna ←
	<hr/>	
	011000	iloczyn cząstkowy
	+ 110000	mnożna ←
	<hr/>	
	1001000	iloczyn cząstkowy
	+ 1100000	mnożna ←
	<hr/>	
	10101000	wynik 128 + 32 + 8 = 168 ₁₀

przeniesienia

W kolejnych krokach dopisywane zero z przodu przed iloczynem cząstkowym, chyba że występuje przeniesienie, to wtedy wpisuje się 1.

Liczby przedstawiane są np. na 8 bitach. Wynik może być 16 bitowy.

Przykładowo liczba dziesiętna 3 w pamięci jest przechowywana jako 00000011, a nie jako 11.

Dlatego dziesiętne $3 \cdot 4$ jest wykonywane jako

$$00000011 \cdot 00000100 = ?$$

$$\begin{array}{r} 00000011 \\ \cdot 00000100 \\ \hline \end{array}$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Mnożenie binarne z wykorzystaniem liczb ujemnych – metoda bezpośrednia I

Iloczyn dwóch liczb n -bitowych może potrzebować $2 \cdot n$ -bitów, aby można było zapisać tam wszystkie możliwe wartości iloczynów. W tym przykładzie został użyty kod U2.

Kroki w tym mnożeniu:

- 1) Podwojenie precyzji czynników mnożenia – dwukrotne zwiększenie liczby bitów – dodanie zer z przodu dla liczby dodatniej i jedynek dla liczby ujemnej,
- 2) Wykonanie mnożenia,
- 3) Odrzucenie bitów powyżej przyjętej precyzji wyników, czyli w tym przypadku dwukrotności liczby bitów czynników. Odrzucone bity zaznaczone są znakiem (x).

Takie działanie jest bardzo nieefektywne. Zachodzi konieczność podwojenia precyzji składników przed samym mnożeniem, a potem wszystkie wartości muszą być podwójnej precyzji. Potrzeba również co najmniej dwa razy więcej iloczynów częściowych niż w przypadku bardziej wydajnych algorytmów mnożenia faktycznie zaimplementowanych w komputerach.

Przykład 1

```

00001010  X=10
· 11110110 Y=(-10)
-----
0
10100
101000
0000
10100000
x01000000
xx10000000
xxx00000000
-----
xxx10011100 Wynik w U2 X·Y
01100100  Moduł wyniku =64+32+4=100

```

Przykład 2

```

11110110  X=(-10)
· 11110110 Y=(-10)
-----
0
x11101100
x11011000
0000
xxx01100000
xxxx11000000
xxxxx10000000
xxxxxxx00000000
-----
xxxxxx01100100 Wynik X·Y=64+32+4=100

```

Niektóre metody mnożenia binarnego liczb nie działają z liczbami ujemnymi bez zastosowania odpowiedniej adaptacji metody. W takim mnożeniu zwykle wykorzystuje się liczby ujemne zapisane w kodzie U2 (uzupełnienia dwójkowe).

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Zwykle nie ma problemu, gdy mnożna (ang. *multiplicand*) (czyli wartość, która jest wielokrotnie dodawana w celu utworzenia iloczynu) jest ujemna. Problem polegający na prawidłowym ustawieniu początkowych bitów iloczynu (wyniku) może pojawić się wtedy, gdy mnożnik jest ujemny. Zatem zwykle w takich przypadkach stosuje się następujące działania:

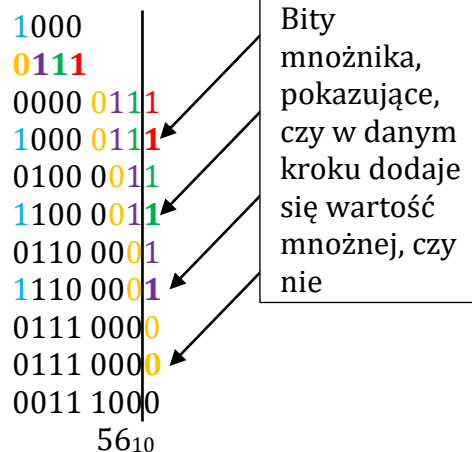
- 1) jeżeli mnożnik jest ujemny, to zmień znaki (użyj ich dopełnień do dwóch) mnożnej i mnożnika. Mnożnik wtedy będzie dodatni, więc algorytm będzie działał prawidłowo. Ponieważ obydwa operandy zostają negowane, wynik nadal będzie miał poprawny znak,
- 2) w przypadku mnożenia przez najważniejszy bit MSB (pseudo-znak, ponieważ ten bit mówi o tym, czy liczba jest dodatnia (0), czy ujemna (1)), odejmij iloczyn częściowy wynikający z mnożenia przez ten bit zamiast dodawać go jak inne iloczyny częściowe.

Mnożenie binarne – metoda bezpośrednia II

Tu za każdym krokiem dodajemy mnożną do wyniku końcowego, a następnie przesuwamy o jedną cyfrę ($/2$) iloczyn cząstkowy.

Przykład mnożenia binarnego dwóch liczb dodatnich

mnożna	(+8)
mnożnik	(+7)
zapisanie argumentów	
krok 1	dodanie +8
	przesunięcie →
krok 2	dodanie +8
	przesunięcie →
krok 3	dodanie +8
	przesunięcie →
krok 4	dodanie 0
	przesunięcie →



Bity mnożnika zastępowane są bitami wyników cząstkowych. Po tylu krokach, ile bitów miał mnożnik (w tym przypadku po czterech) ośmiobitowe słowo zawiera końcowy wynik.

Wada tego mechanizmu – jeżeli mnożna byłaby większa niż 8, wtedy dochodziłoby do przepełnienia.

W tej metodzie **przesuwa się iloczyn cząstkowy w prawo**.

Mnożenie liczb rzeczywistych

Przykład

$$X = 4,25_{10} = 100,01_2$$

$$Y = 125,125_{10} = 1111101,001_2$$

$$X \cdot Y = ?$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Krok 1 – zapisujemy poszczególne liczby w formacie najmniejsza możliwa liczba_całkowita pomnożona przez podstawę systemu liczbowego podniesionego do określonej potęgi:

$$X = 4,25_{10} = 100,01_2 = 10001_2 \cdot 2^{-2}$$

$$Y = 125,125_{10} = 1111101,001_2 = 1111101001_2 \cdot 2^{-3}$$

$$X \cdot Y = 10001_2 \cdot 2^{-2} \cdot 1111101001_2 \cdot 2^{-3} = 10001_2 \cdot 1111101001_2 \cdot 2^{-3} \cdot 2^{-2} = 10001_2 \cdot 1111101001_2 \cdot 2^{-5}$$

Krok 2 – mnożymy te liczby całkowite:

$$\begin{array}{r} 1111101001_2 \\ \cdot \quad 10001_2 \\ \hline 1111101001_2 \\ + 1111101001_2 \\ \hline 100001001111001 \end{array}$$

I dodajemy wykładniki

$$2^{-2} + 2^{-3} = 2^{-5}$$

Zatem wynik mnożenia jest następujący:

$$X \cdot Y = 100001001111001_2 \cdot 2^{-5} = 1000010011,11001_2$$

Czyli analogicznie jak w przypadku mnożenia liczb dziesiętnych, liczba miejsc po przecinku wyniku jest sumą liczb po przecinku poszczególnych czynników mnożenia.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Mnożenie binarne – Metoda Bootha

Reguły ogólne

- Mnożenie zgodnie z metodami Bootha polega na wyliczaniu iloczynów częściowych w kolejnych krokach.
- Mnożenie można rozpocząć od najmłodszego bitu mnożnika, aż do najstarszego. W takim przypadku w kolejnych krokach stosujemy narastającą wartość – każdorazowo mnożoną $\cdot 2$.
- Jeżeli mnożenie rozpoczyna się od najstarszego bitu mnożnika aż do najmłodszego, wtedy wykorzystuje się dzielenie przez 2 – przesuwanie w prawą stronę.
- Ze względu na przemienność w dodawaniu to, czy rozpoczyna się mnożenie od najstarszego bitu mnożnika, czy on najmniejszego nie ma większego znaczenia.
- W metodzie **Bootha I** mnożona/dzielona, czyli **przesuwana jest mnożna**, natomiast w metodzie **Bootha II** – wyliczony **iloczyn częściowy**

Mnożenie binarne – Metoda Bootha II

Mnożymy dwie liczby:

- M – mnożna
- Q – mnożnik

Mnożna i mnożnik muszą mieć tyle samo cyfr. Działania wykonujemy w tylu krokach, ile jest cyfr mnożnika (lub mnożnej).

Wykorzystujemy uzupełnienie U2 mnożnika M .

Wynik zapisujemy w dwóch liczbach:

A – tam będą starsze bity; na początku $A = 0$;

Q – tam na początku jest mnożnik, a następnie przesuwane są bity z A (w jednym kroku przesunięcia najmłodszy bit A wchodzi jako najstarszy bit Q).

Liczby A , M i Q mają tyle samo cyfr. Działanie mnożenia ma tyle kroków, ile jest cyfr w każdej liczbie.

W zależności od ostatniej cyfry mnożnika Q_0 i tej, która była poprzednio Q_{-1}

(za pierwszym razem „poprzednio” nie było żadnej cyfry, więc przyjmujemy $Q_{-1} = 0$) realizujemy następujące działania:

- $Q_0=1$ i $Q_{-1} = 1$ lub $Q_0=0$ i $Q_{-1} = 0$, wtedy tylko **przesuwamy** otrzymywany iloczyn częściowy o jedną pozycję w prawo (dzielenie przez 2)
- $Q_0=0$ i $Q_{-1} = 1$, wtedy wykonujemy działanie $A = A+M$, a następnie **przesuwamy** otrzymywany iloczyn częściowy o jedną pozycję w prawo (dzielenie przez 2)
- $Q_0=1$ i $Q_{-1} = 0$, wtedy wykonujemy działanie $A = A-M$, czyli $A = A + (-M)$, gdzie $(-M)$ to uzupełnienie U2 liczby M , a następnie **przesuwamy** otrzymywany iloczyn częściowy o jedną pozycję w prawo (dzielenie przez 2)

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład mnożenia wykorzystując metodę Bootha

$$M \cdot Q = 11 \cdot (-14) = ?$$

$$M = +11_{10} = 01011$$

$$-M = \bar{M} = (-11)_{10} = 10101_{U2}$$

$$Q = (-14)_{10} = 10010_{U2}$$

$$A = 00000$$

Krok	A	Q	Q _{n-1}	Opis działania
0	00000	10010	0	Na początku jako wartość Q _{n-1} wpisujemy 0
1	00000	01001	0	Porównujemy podkreślone liczby: Q ₀ =0 i Q _{n-1} =0, zatem jedynie przesuwamy wartości/cyfry w prawo →
2	+10101 =10101 11010	01001 01001 10100	1	Porównujemy podkreślone liczby: Q ₀ =1 i Q _{n-1} =0, zatem wykonujemy A = A + (-M), i przesuwamy cyfry w prawo →
3	+01011 =00101 00010	10100 11010	0	Porównujemy podkreślone liczby: Q ₀ =0 i Q _{n-1} =1, zatem wykonujemy A = A + M (pomijamy przeniesienie!!!), i przesuwamy cyfry w prawo →
4	00001	01101	0	Porównujemy podkreślone liczby: Q ₀ =0 i Q _{n-1} =0, zatem jedynie przesuwamy wartości/cyfry w prawo →
5	+10101 10110 11011	01101 00110		Porównujemy podkreślone liczby: Q ₀ =1 i Q _{n-1} =0, zatem wykonujemy A = A + (-M), i przesuwamy cyfry w prawo →

Uzyskaliśmy wynik mnożenia:

$$M \cdot Q = 11 \cdot (-14) = 1101100110_{U2} = 1.010011010_{ZM} = (-1) \cdot (128+16+8+2) = (-154)$$

lub

$$M \cdot Q = 11 \cdot (-14) = 1101100110_{U2} = (-1) \cdot 512+256+64+32+4+2 = (-154)$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład mnożenia wykorzystując metodę Bootha

$$M \cdot Q = \left(-6\frac{3}{8}\right) \cdot 7\frac{12}{16} = ?$$

$$M = \left(-6\frac{3}{8}\right)_{10} = 111001,101_{U2} = (111001101 \cdot 10^{-11})_{U2}$$

$$-M = \bar{M} = +6\frac{3}{8}_{10} = 000110,011_{U2} = (000110011 \cdot 10^{-11})_{U2}$$

$$Q = +7\frac{12}{16}_{10} = 00111,1100_{U2} = (001111100 \cdot 10^{-100})_{U2}$$

$$A = 000000000$$

W przykładzie dodano dwie **cyfry wiodące** przed cyframi znaczącymi.

Jeżeli mnożymy ułamki to nie muszą mieć one tyle samo cyfr/miejsc po przecinku. Wszystkie liczby używane w tym algorytmie jednak muszą mieć tyle samo cyfr (całkowite + ułamki + ewentualne zera wiodące). W tym przykładzie założono, że cyfr będzie 9.

Dlatego też z przodu przed liczbami M i $-M$ dodano po jednej **dodatkowej cyfrze**.

Na etapie wykonywania obliczeń pominiemy przecinki.

Podczas pisania ujemnych wartości wykładników nie używamy dopełnień, tylko piszemy je w notacji znak-moduł.

Krok	A	Q	Q _{n-1}	Opis działania
0	000000000	001111100	0	Na początku jako wartość Q _{n-1} = 0
1	000000000	000111110	0	Porównujemy podkreślone liczby: Q ₀ =0 i Q _{n-1} = 0, zatem jedynie przesuwamy wartości/cyfry w prawo →
2	000000000	000011111	0	Porównujemy podkreślone liczby: Q ₀ =0 i Q _{n-1} = 0, zatem jedynie przesuwamy wartości/cyfry w prawo →
3	000000000 +000110011 =000110011 000011001	000011111 000011111 000011111 100001111	1	Porównujemy podkreślone liczby: Q ₀ =1 i Q _{n-1} = 0, zatem wykonujemy A = A + (-M), i przesuwamy cyfry w prawo →
4	000001100	110000111	1	Porównujemy podkreślone liczby: Q ₀ =1 i Q _{n-1} = 1, zatem jedynie przesuwamy wartości/cyfry w prawo →
5	000000110	011000011	1	Porównujemy podkreślone liczby: Q ₀ =1 i Q _{n-1} = 1, zatem jedynie przesuwamy wartości/cyfry w prawo →
6	000000011	001100001	1	Porównujemy podkreślone liczby: Q ₀ =1 i Q _{n-1} = 1, zatem jedynie przesuwamy wartości/cyfry w prawo →
7	000000001	100110000	1	Porównujemy podkreślone liczby: Q ₀ =1 i Q _{n-1} = 1, zatem jedynie przesuwamy wartości/cyfry w prawo →
8	000000001 +111001101 =111001110 111100111	100110000 010011000	0	Porównujemy podkreślone liczby: Q ₀ =0 i Q _{n-1} = 1, zatem wykonujemy A = A + M (pomijamy przeniesienie!!!), i przesuwamy cyfry w prawo →
9	111110011	101001100		Porównujemy podkreślone liczby: Q ₀ =0 i Q _{n-1} = 0, zatem jedynie przesuwamy wartości/cyfry w prawo →

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Uzyskaliśmy wynik mnożenia:

$$M \cdot Q = 111110011101001100_{U2}$$

Mnożyliśmy takie dwie liczby, które w sumie mają 7 miejsc/cyfr po przecinku

$$M = (-5 \frac{3}{8})_{10} = 111001,101_{U2} = (111001101 \cdot 10^{-11})_{U2}$$

$$Q = +7 \frac{12}{16_{10}} = 00111,1100_{U2} = (001111100 \cdot 10^{-100})_{U2}$$

Po uwzględnieniu wykładników:

$$\begin{aligned} M \cdot Q &= (111001101 \cdot 10^{-11})_{U2} \cdot (001111100 \cdot 10^{-100})_{U2} = \\ &= (111001101 \cdot 001111100 \cdot 10^{-11} \cdot 10^{-100})_{U2} = (111110011101001100 \cdot 10^{-111})_{U2} \end{aligned}$$

Zatem nasz wynik również będzie miał 7 cyfr po przecinku

$$M \cdot Q = (111110011101001100 \cdot 10^{-111})_{U2} = 11111001110,1001100_{U2}$$

Jedynka z przodu świadczy o tym, że otrzymaliśmy liczbę ujemną

$$\begin{aligned} M \cdot Q &= \left(-6 \frac{3}{8}\right) * 7 \frac{12}{16} = 11111001110,1001100_{U2} = 1.00000110001,0110100_{ZM} = \\ &= \left(-49 \frac{52}{128}\right)_{10} \end{aligned}$$

Mechanizm Bootha – jak działa?

Aby wyjaśnić mechanizm założmy, że jest mnożona jakaś liczba **M** razy mnożnik, który składa się ze zgrupowanych jedynek otoczonych zerami np. $00011110_2 = 30_{10}$

$$A = M \cdot 00011110 = M \cdot (2^4 + 2^3 + 2^2 + 2^1) = M \cdot 30$$

Ale okazuje się, że zamiast czterech działań mnożenia można wykonać jedynie dwie:

$$A = M \cdot 001000-10 = M \cdot (2^5 - 2^1) = M \cdot 30$$

Zatem dowolne sekwencję samych jedynek w liczbie można zastąpić przez różnicę dwóch liczb binarnych

$$(...01...10...) = (...10...00...) - (...00...10...) - \text{kolorami oznaczono te same pozycje w liczbach}$$

Inaczej przedstawiając to samo zagadnienie – mając ciąg jedynek

- Ostatnią (najmłodszą) zamieniamy na **-1**
- Wszystkie środkowe oraz pierwszą na zera
- Zero przed pierwszą jedyneką zamieniamy na **1**

$$B = 0001110$$

Zastępuje się przez:

$$B = 00100-10$$

Jeżeli mamy przypadek z jedną jedyneką np.

$$C = 0001000_2 = 8_{10}$$

Razem z zerem przed nią zostaje ona zamieniona w następujący sposób:

$$C = 001-1000_2 = 16_{10} - 8_{10} = 8_{10}$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Dlatego też mnożenie mnożnika M razy ciąg/łańcuch jedynek może być zastąpiony przez prostsze działania: dodanie mnożnika, kilka przesunięć iloczynu częściowego o wymaganą liczbę pozycji (liczba jedynek w łańcuchu pomniejszona o 1), a następnie odjęcie mnożnika.

Podczas przesuwania iloczynu częściowego nie trzeba nic robić (to tak, jakby mnożnik mnożyć razy 0 dla kolejnych pozycji).

W przypadku wykonywania działań na liczbach dziesiętnych można postąpić podobnie:
 $M \cdot 999 = M \cdot (1000 - 1)$

Opisany schemat można rozszerzyć na większą liczbę bloków jedynek. Przykładowo:
 $M \cdot 01110110 = M \cdot (2^6 + 2^5 + 2^4 + 2^2 + 2^1) = M \cdot 118$

$$M \cdot 10011010 = M \cdot (2^7 - 2^4 + 2^3 - 2^1) = M \cdot 118$$

Algorytm Bootha postępuje według tego samego schematu:

- wykonuje dodawanie, gdy napotyka pierwszą cyfrę bloku jedynek (01),
- wykonuje odejmowanie, gdy napotyka koniec bloku jedynek (10),
- jeżeli znajduje się w bloku jedynej (11), lub poza blokami jedynek (00) to nie wykonuje żadnego działania (tylko przesuwanie).

Ten algorytm działa również prawidłowo w przypadku ujemnego mnożnika.

Gdy te w multiplikatorze są pogrupowane w długie bloki, algorytm Bootha wykonuje mniej dodawań i odejmowań niż normalny algorytm mnożenia.

Dzielenie liczb binarnych

Algorytm dzielenia binarnego jest bardziej skomplikowany niż mnożenia. Iloraz musi być zapisany na określonej liczbie bitów = wynik może zostać przedstawiony jedynie z określoną dokładnością.

Istnieje bardzo wiele algorytmów dzielenia. Większość algorytmów dzielenia zwraca iloraz i resztę z dzielenia. Dzielną może być podwójnej długości w porównaniu do dzielnika.

Tu nie ma odwrotności jak w mnożeniu. Jeżeli mnoży się dwie liczby 8 bitowy to można otrzymać wynik 16 bitowy. To wcale nie oznacza, że dzieląc liczbę 16 bitową przez 8 bitową otrzyma się wynik, który zmieści się w 8 bitach. Przykładowo dzielnik może wynosić 00000010.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

dzielna	(+50)	0011 0010
dzielnik	(+6)	0110
krok 1	przesunięcie ← odejmowanie	0110 010_ 0110 0000 010 1
krok 2	przesunięcie ← odejmowanie	0000 101_ 0110 wynik ujemny
krok 3	przywrócenie przesunięcie ← odejmowanie	0000 101 0 0001 010_ 0110 wynik ujemny
	przywrócenie	0001 010 0
krok 4	przesunięcie ← odejmowanie	0010 100_ 0110 wynik ujemny
	przywrócenie	0010 100 0 ↑ ↑ reszta wynik 2₁₀ 8₁₀

Arytmetyka w kodzie BCD

Negowanie liczb w kodzie BCD

Zamiana bitu odpowiadającego za znak

Liczba **+87**

0 1000 0111

Liczba **(-87)**

1 1000 0111

Dodawanie w kodzie BCD

Obliczamy $Z = \pm X \pm Y$ dla liczb dziesiętnych $X=50$ i $Y=60$ przedstawionych w zapisach odwrotnym [jeżeli któraś liczba jest ujemna (znak -) wykorzystuje się uzupełnienie dziewiątkowe] i dopełnieniowym [jeżeli któraś liczba jest ujemna (znak -) wykorzystuje się uzupełnienie dziesiątkowe] o cyfrach przedstawionych w kodzie BCD lub EXCESS-3.

Dodając dwie liczby, które mają po n cyfr przed przecinkiem należy uwzględnić, że wynik może mieć $n+1$ cyfr przed przecinkiem. Należy to brać pod uwagę przy dokonywaniu wyliczeń. Można do każdej liczby – składnika dodawania dodać dodatkową cyfrę z przodu – tzw. zero wiodące.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Uzupełnienia w kodzie BCD

Liczba dziesiętna $A = 68,3$ przedstawiona w kodzie BCD 8421

$$\begin{array}{r} A_{8421} = 0110\ 1000,0011 \\ + 0110\ 0110,0110 \text{ – korekta} \\ \hline = 1100\ 1110,1001 \end{array}$$



Uzupełnienie dziewiątkowe $\bar{A}_{8421} = 0011\ 0001,0110$

$$\begin{array}{r} \bar{A}_{8421} = 0011\ 0001,0110 \\ + 1 \\ \hline \bar{A}_{8421} = 0011\ 0001,0111 \end{array}$$

$$\begin{array}{r} 23 + 47 = \quad 31 - 17 = \\ 0010\ 0011 \quad 0011\ 0001 \\ + 0100\ 0111 \quad - 0001\ 0111 \end{array}$$

$$\begin{array}{r} 0110\ 1010 \quad 0001\ 1001 \\ \quad \quad \quad 19 \\ \text{Wynik zły!} \quad \text{Wynik zły!} \end{array}$$

$$\begin{array}{r} 23 + 47 \quad 31 - 17 \\ 0010\ 0011 \quad 0011\ 0001 \\ + 0100\ 0111 \quad - 0001\ 0111 \\ \hline 0110\ 1010 \quad 0001\ 1001 \\ + 0000\ 0110 \quad - 0000\ 0110 \\ \hline 0111\ 0000 \quad 0001\ 0100 \\ \text{wynik } 70_{10} \quad \text{wynik } 14_{10} \end{array}$$

pożyczka
korekta

Liczby ujemne w kodzie BCD

Aby przedstawić liczby ujemne w kodzie BCD stosowane są dwa typy notacji:

- Znak moduł – czyli do tetrad odpowiadających dziesiętnym wartościom liczby dodany jest z przodu bit, który pokazuje znak liczby
- Analogicznie jak w przypadku kodów U1 i U2, zakłada się, że pierwsza cyfra liczby mówi o znaku. Jeżeli jest to 0000 – czyli zero, wtedy mamy do czynienia z liczbą dodatnią, natomiast jeżeli jest to 1001 – czyli dziewięć, wtedy jest to dopełnienie liczby (rozumiane jako liczba ujemna).

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Uzupełnienia w kodzie BCD i EXCESS-3

Przykładowa liczba: $A = 2039$

$$A_{BCD} = 0010\ 0000\ 0011\ 1001$$

$$\text{Korekta} = 0110\ 0110\ 0110\ 0110$$

$$1000\ 0110\ 1001\ 1111$$

$$0 \leftrightarrow 1 \quad 0111\ 1001\ 0110\ 0000 = \bar{A}_{BCD} + 1$$

$$0111\ 1001\ 0110\ 0000 = \bar{\bar{A}}_{BCD}$$

Ogólnie stosowany mechanizm tworzenia uzupełnienia U10 z uzupełnienia U9, polegający na dodaniu do najmniej znaczącego bitu (lub tetrady w kodach typu BCD) ma pewne ograniczenia i jest znacznym uproszczeniem, **jednak powszechnie stosowanym**. Widać to na przykładzie tworzenia uzupełnienia U10 z U9 w kodzie Excess-3, jeżeli ostatnia cyfra równałaby się zero.

Przykładowa liczba:

$$A = 200$$

$$A_{BCD} = 0010\ 0000\ 0000$$

$$A_{EX-3} = 0101\ 0011\ 0011 \quad (200)$$

$$0 \leftrightarrow 1 \quad 1010\ 1100\ 1100 = \bar{A}_{EX3} \quad (799) \quad \begin{array}{l} \text{zamiana zer na jedynek i jedynek na zera} \\ \text{dodanie 1, żeby przejść z U9 na U10} \end{array}$$

$$+1$$

$$1010\ 1100\ 1101 = \bar{\bar{A}}_{EX3} \quad (\text{powinno wyjść } 800!)$$

Można tu przyjąć **uproszczone rozwiązanie**

Przykładowa liczba:

$$A = 200$$

$$A_{BCD} = 0010\ 0000\ 0000$$

$$A_{EX-3} = 0101\ 0011\ 0011 \quad (200)$$

$$0 \leftrightarrow 1 \quad 1010\ 1100\ 1100 = \bar{A}_{EX3} \quad (799) \quad \begin{array}{l} \text{zamiana zer na jedynek i jedynek na zera} \\ \text{dodanie 1, żeby przejść z U9 na U10} \end{array}$$

$$+1$$

$$1010\ 1100\ 1101 = \bar{\bar{A}}_{EX3} \quad (\text{powinno wyjść } 800!)$$

$$\text{Korekta} \quad +0110$$

$$1010\ 1101\ 0011 = \bar{\bar{A}}_{EX3} \quad (\text{powinno wyjść } 800!)$$

$$\text{Korekta} \quad +0110$$

$$1011\ 0011\ 0011 = \bar{\bar{A}}_{EX3} \quad (\text{prawidłowy wynik } 800!)$$

Poniżej podane jest **w pełni ściśle rozwiązanie**, jednak na świecie stosuje się rozwiązania uproszczone.

Podobny problem w tym kodzie wystąpiłby, jeżeli tetrada równałaby się 9, czyli 1100, i należałoby do niej dodać przeniesienie z poprzedniej tetrady. Również podobnie wygląda kwestia, kiedy jest w kodzie U9 przeniesienie z najwyższej tetrady, które dodaje się do ostatniej, najmniej znaczącej cyfry.

Zarówno w przypadku dodawania 1 jako przeniesienia, jak również 1 jako przejścia z kodu U9 na U10 należy pamiętać, że jest to kod przesunięty, i zatem należałoby dodawać przesuniętą jedynekę, czyli 0100, a następnie zastosować korekty wynikające z otrzymanego wyniku dodawania dwóch liczb w tym kodzie.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykładowa liczba: $A = 200$

$$A_{BCD} = 0010\ 0000\ 0000$$

$$A_{EX-3} = 0101\ 0011\ 0011 \quad (200)$$

$0 \leftrightarrow 1$ $1010\ 1100\ 1100 = \bar{A}_{EX3} \quad (799)$ zamiana zer na jedynki i jedynek na zera
 $+0100$ – dodanie przesuniętej o 3 jedynki, by przejść z U9 na U10

$$1010\ 1100\ 0000 = \bar{\bar{A}}_{EX3}$$

Przeniesienie $+0100\ 0011$ – **korekta** wyniku dodawania dwóch liczb i przeniesienie

$$1010\ 0000\ 0011 = \bar{\bar{A}}_{EX3}$$

Przenies. $+0100\ 0011$ – **korekta** wyniku dodawania dwóch liczb z przeniesieniem

$$1110\ 0011\ 0011 = \bar{\bar{A}}_{EX3}$$

-0011 – **korekta** wyniku dodawania dwóch liczb bez przeniesienia

$$1011\ 0011\ 0011 = \bar{\bar{A}}_{EX3} \quad (\text{prawidłowy wynik } 800!)$$

Przykładowa liczba: $A = 2039$

$$A_{BCD} = 0010\ 0000\ 0011\ 1001$$

$$A_{EX-3} = 0101\ 0011\ 0110\ 1100 \quad (2039)$$

$$0 \leftrightarrow 1 \quad 1010\ 1100\ 1001\ 0011 = \bar{A}_{EX-3} \quad (7960)$$

+1

$$1010\ 1100\ 1001\ 0100 = \bar{\bar{A}}_{EX-3} \quad (7961)$$

Bardzo ścisły sposób tworzenia uzupełnień U9 i U10

Liczba dziesiętna $A = 68,3$ przedstawiona w kodzie EXCESS-3

$$A_{BCD} = 0110\ 1000,0011$$

$$A_{EX3} = 1001\ 1011,0110$$



$$\text{Uzupełnienie dziewiątkowe } \bar{A}_{EX3} = 0110\ 0100,1001$$

$+0100$

$$\text{Uzupełnienie dziesiątkowe } \bar{\bar{A}}_{EX3} = 0110\ 0100,1101$$

-0011 korekta po sumowaniu

$$\text{Uzupełnienie dziesiątkowe } \bar{\bar{\bar{A}}}_{EX3} = 0110\ 0100,1010$$

Zalety kodu EXCESS-3

Chcemy otrzymać wynik następującego działania:

uzupełnienie = 9-cyfra

Negowanie bitów w tetradach kodów liczb dziesiętnych kodowanych binarnie to w zasadzie działanie polegające na odjęciu od liczby 15 wartości prezentowanej na bitach.

Zatem w przypadku kodu BCD po takim odjęciu $15-7 = 8$ i wynik nie jest tożsamy z oczekiwanym wynikiem działania. Dlatego też jest wprowadzona poprawka. Wtedy działanie wynosi $15-(7+6) = 2$.

Kod EXCESS-3 każdą cyfrę pamięta, jako jej kod NKB powiększony/przesunięty o trzy. W takim przypadku otrzymamy $15-(7+3) = 5$. Rozkodowując otrzymaną wartość uzyskuje się wartość prawidłową.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Korekty przy dodawaniu liczb w kodzie BCD

Jeżeli w wyniku sumowania:

1. **nie wystąpiło przeniesienie** oraz wartość tetrady jest w zakresie $0 \div 9$, czyli $0000 \div 1001$, wtedy korekta jest niepotrzebna,
2. **nie wystąpiło przeniesienie** oraz wartość tetrady jest większa niż 9_{10} , czyli w zakresie $10 \div 15$ ($1010 \div 1111$) i nie należy już do kodu BCD, wtedy konieczna jest korekta i wynosi $+0110$ (czyli $+6_{10}$). Jest to przesunięcie otrzymanej wartości o 6 nieużywanych pozycji kodu BCD. Występujące wówczas przeniesienie dodaje się do starszej tetrady. Jeżeli w wyniku takich działań w którejś wartości którejś kolejnej tetrady jest większa niż 9_{10} , wtedy należy ponownie zastosować poprawkę, czyli dodać korektę $+0110$ (czyli $+6_{10}$),
3. **występuje przeniesienie** do starszej tetrady np. podczas sumowania dwóch wartości np. $1001 + 1001$ ($9_{10} + 9_{10}$), to w tej tetradzie konieczne jest również skorygowanie wyniku i dodanie do niego wartości $+0110$ (czyli $+6_{10}$). W wyniku bowiem sumowania podanych wyżej liczb, bez dokonania korekty wyszłaby wartość $1\ 0010$, czyli w tej tetradzie zapisana zostałaby wartość 0010 , czyli 2_{10} , a powinna być wartość 8_{10} , czyli większa o sześć.

W wyżej wymienionym przypadku faktu stosowania korekty nie można uzależnić od wystąpienia przeniesienia. Przy sumowaniu i wystąpieniu przeniesienia do wyniku tetrady dodaje się 0110 . Wprowadzanie korekt może być wielostopniowe.

Korekty przy dodawaniu liczb w kodzie EXCESS-3

Dla wyniku dodawania z zakresu $0110 \div 1111$ korekta przy konwersji na kod BCD wynosi -0011 lub $+1101$, ale wtedy należy pominąć występujące przeniesienie z tetrady.

W przypadku wartości $10000 \div 11001$ korekta wynosi $+0011$.

Szczególny przypadek korekty występuje, kiedy tetradą ma wartość $9 - 1100$ i dodaje się do niej przeniesienie z poprzedniej tetrady lub dodaje się wartość $+1$, jeżeli jest to ostatnia cyfra i wylicza się uzupełnienie U_{10} z uzupełnienia U_9 . W takich przypadkach należy dodać tę jedynekę przekonwertowaną na kod Excess-3, czyli wartość 0100 , a następnie zastosować ww. korekty wg potrzeby.

Wartość poprawki można uzależnić od występującego przeniesienia:

- jest przeniesienie z tetrady, to należy ją skorygować $+0011$
- nie ma przeniesienia z tetrady – korekta wynosi -0011 albo $+1101$, ale w tym drugim przypadku należy pominąć ewentualne przeniesienie.

Dodawanie liczb ujemnych

$$\begin{array}{rcl}
 0001\ 0010\ 0000 & \rightarrow & 120_{10} \\
 +\ 0000\ 1000\ 0101 & \rightarrow & 85_{10} \\
 \hline
 0001\ 1010\ 0101 & & \\
 0001\ 0110 & & \text{korekta i przeniesienie} \\
 \hline
 0010\ 0000\ 0101 & \rightarrow & 205_{10}
 \end{array}$$

str. 78

Zasady ogólne

Generalnie należy założyć, że suma dwóch liczb, z których liczba mają więcej cyfr przed przecinkiem ma ich n , to w wyniku można otrzymać liczbę mającą $n+1$ cyfr/tetrad w kodzie BCD.

W wyniku odejmowania jednej liczby z dwoma cyframi przed przecinkiem od drugiej liczby z dwoma cyframi przed przecinkiem wynik może mieć maksymalnie dwie cyfry znaczące przed przecinkiem.

W przypadku wykonywania odejmowania z wykorzystaniem uzupełnień (U9 lub U10) należy dodać z przodu jeszcze jedną cyfrę, która w wyniku będzie nam mówiła, czy otrzymaliśmy liczbę dodatnią, czy uzupełnienie (czyli musimy ją rozkodować, żeby otrzymać ujemną wartość wynikową).

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład odejmowania - kod BCD z U9 – wynik <0

Należy wyliczyć różnicę $A-B$ posługując się uzupełnieniami (U9)

Dane są dwie liczby:

$A = 14,37$

$B = 22,65$

Kroki postępowania

Dwie liczby	
0000 0001 0100,0011 0111	Liczba <i>A</i> = 014,37 przedstawiona w kodzie BCD
0000 0010 0010,0110 0101	Liczba <i>B</i> = 022,65 przedstawiona w kodzie BCD
Tworzenie uzupełnienia U9 liczby <i>B</i>	
0110 0110 0110 0110 0110 – korekta	1) wymagana korekta – należy do każdej tetrazy liczby <i>B</i> dodać wartość 6₁₀ – 0110₂ – liczba <i>B</i> po korekcie. 2) uzupełnienie U9 skorygowanej liczby <i>B</i> (zamiana zer na jedynki i jedynek na zera) → 977,34
0110 1000 1000,1100 1011	
1001 0111 0111,0011 0100	
Sumowanie liczby <i>A</i> i U9 liczby <i>B</i>	
0000 0001 0100,0011 0111	Liczba <i>A</i>
1001 0111 0111,0011 0100	Uzupełnienie U9 liczby <i>B</i>
1001 1000 1011 0110 1011	1) Sumowanie liczb <i>A</i> i U9 liczby <i>B</i>
1001 1001 0001,0111 0001	2) Korekta liczb, które nie należą do kodu BCD ; na czerwono zaznaczono przeniesienia
	3) uzyskany wynik <i>A-B</i>
Przy sumowaniu liczby dodatniej i uzupełnienia innej liczby dodatniej nie ma przeniesienia, stąd wniosek, że wynik jest uzupełnieniem (wynikiem jest liczba ujemna). Liczba 1001, czyli 9 ₁₀ na początku wyniku mówi nam, że mamy do czynienia z uzupełnieniem. Wynik zatem jest ujemny.	
Rozkodowanie wyniku z U9	
1001 1001 0001,0111 0001 – wynik	Wynik – otrzymano liczbę ujemną. 1) Korygujemy od każdej tetrazy dodając wartość 6₁₀ – 0110₂
0110 0110 0110 0110 0110 – korekta	
1111 1111 0111,1101 0111	Rozkodowujemy, tworząc uzupełnienie U9 otrzymanego wyniku, czyli powinien wyjść wynik dodatni, czyli moduł otrzymanej liczby – wyniku 2) zamieniamy zera na jedynki i jedynki na zera. Jest to dziesiętna wartość 8,28. Wynik jest ujemny, czyli wynosi <i>A-B</i> = (-8,28).
0000 0000 1000,0010 1000	

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład odejmowania - kod BCD z U9 – wynik >0

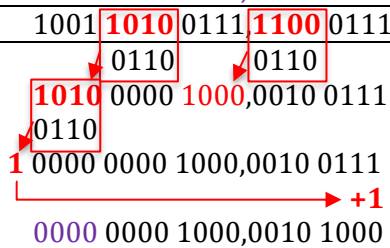
Należy wyliczyć różnicę $B-A$ posługując się uzupełnieniami (U9)

Dane są dwie liczby:

$A = 14,37$

$B = 22,65$

Kroki postępowania

Dwie liczby	
0000 0010 0010,0110 0101	Liczba $B = 022,65$ przedstawiona w kodzie BCD
0000 0001 0100,0011 0111	Liczba $A = 014,37$ przedstawiona w kodzie BCD
Tworzenie uzupełnienia U9 liczby A	
0110 0110 0110 0110 0110 – korekta	1) wymagana korekta – należy do każdej tetrady liczby A dodać wartość $6_{10} - 0110_2$ – liczba A po korekcie.
0110 0111 1010,1001 1101	
1001 1000 0101,0110 0010	2) uzupełnienie U9 skorygowanej liczby A (zamiana zer na jedynki i jedynek na zera) → 985,62
Sumowanie liczby B i U9 liczby A	
0000 0010 0010,0110 0101	Liczba B
1001 1000 0101,0110 0010	Uzupełnienie U9 liczby A
1001 1010 0111 1100 0111	1) Sumowanie liczby B i U9 liczby A
	2) Korekta liczb , które nie należą do kodu BCD ; na czerwono zaznaczono przeniesienia Przeniesienie z najbardziej znaczącej cyfry – należy dodać do najmniej znaczącej cyfry
0000 0000 1000,0010 1000	3) uzyskany wynik $B-A$
Przy sumowaniu liczby dodatniej i uzupełnienia innej liczby dodatniej jest przeniesienie (nadmiar, przepełnienie), stąd wniosek, że wynikiem jest liczba dodatnia. Liczba 0000, czyli 0_{10} na początku wyniku mówi nam, że mamy do czynienia z wynikiem dodatnim!	
0000 0000 1000,0010 1000	Jest to dziesiętna wartość $B-A = 8,28$, czyli wynik prawidłowy

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład działania - kod BCD z U9 $\rightarrow (-A)+(-B)$

Należy wyliczyć sumę $(-A)+(-B)$ posługując się uzupełnieniami (U9)

Dane są dwie liczby:

$A = 14,37$

$B = 22,65$

Kroki postępowania

Dwie liczby	
0000 0001 0100,0011 0111	Liczba $A = 014,37$ przedstawiona w kodzie BCD
Tworzenie uzupełnienia U9 liczby A	
0110 0110 0110 0110 0110 – korekta	1) wymagana korekta – należy do każdej tetrady liczby A dodać wartość $6_{10} - 0110_2$ – liczba A po korekcie.
0110 0111 1010,1001 1101	
1001 1000 0101,0110 0010	2) uzupełnienie U9 skorygowanej liczby A (zamiana zer na jedyńki i jedynek na zera) $\rightarrow 985,62$
Dwie liczby	
0000 0010 0010,0110 0101	Liczba $B = 022,65$ przedstawiona w kodzie BCD
Tworzenie uzupełnienia U9 liczby B	
0110 0110 0110 0110 0110 – korekta	1) wymagana korekta – należy do każdej tetrady liczby B dodać wartość $6_{10} - 0110_2$ – liczba B po korekcie.
0110 1000 1000,1100 1011	
1001 0111 0111,0011 0100	2) uzupełnienie U9 skorygowanej liczby B (zamiana zer na jedyńki i jedynek na zera) $\rightarrow 977,34$
Sumowanie U9 liczby A i U9 liczby B	
1001 1000 0101,0110 0010 1001 0111 0111,0011 0100	Uzupełnienie U9 liczby A Uzupełnienie U9 liczby B
10010 1111 1100,1001 0110 10110 1111 0010,1001 0110 1001 0000 0010,1001 0110 0110 1001 0110 0010,1001 0110 1001 0110 0010,1001 0111	1) Sumowanie U9 liczby A i U9 liczby B 2) Korekty liczb , które nie należą do kodu BCD i tam, gdzie wystąpiło przeniesienie ; na czerwono zaznaczono przeniesienia 3) uzyskany wynik $(-A)+(-B)$
Liczba 1001 , czyli 9_{10} na początku wyniku mówi nam, że mamy do czynienia z uzupełnieniem. Wynik zatem jest ujemny.	
Rozkodowanie wyniku z U9	
1001 0110 0010,1001 0111 – wynik 0110 0110 0110 0110 0110 – korekta 1111 1100 1000,1111 1101	Wynik – otrzymano liczbę ujemną. 1) Korygujemy od każdej tetrady dobijając wartość $6_{10} - 0110_2$
0000 0011 0111,0000 0010	Rozkodowujemy, tworząc uzupełnienie U9 otrzymanego wyniku, czyli powinien wyjść wynik dodatni, czyli moduł otrzymanej liczby – wyniku 2) zamieniamy zera na jedyńki i jedynek na zera . Jest to dziesiętna wartość $37,02$. Wynik jest ujemny, czyli wynosi $(-A)+(-B) = (-37,02)$.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład odejmowania - kod BCD z U10 – wynik <0

Należy wyliczyć różnicę $A-B$ posługując się uzupełnieniami (U10)

Dane są dwie liczby:

$A = 14,37$

$B = 22,65$

Kroki postępowania

Dwie liczby	
0000 0001 0100,0011 0111	Liczba <i>A</i> = 014,37 przedstawiona w kodzie BCD
0000 0010 0010,0110 0101	Liczba <i>B</i> = 022,65 przedstawiona w kodzie BCD
Tworzenie uzupełnienia U10 liczby <i>B</i>	
0110 0110 0110 0110 0110 – korekta	1) wymagana korekta – należy do każdej tetrady liczby <i>B</i> dodać wartość 6₁₀ – 0110₂ – liczba <i>B</i> po korekcie. 2) uzupełnienie U10 skorygowanej liczby <i>B</i> (zamiana zer na jedyne i jedynek na zera +1) → 977,35
0110 1000 1000,1100 1011	
1001 0111 0111,0011 0101	
Sumowanie liczby <i>A</i> i U10 liczby <i>B</i>	
0000 0001 0100,0011 0111	Liczba <i>A</i>
1001 0111 0111,0011 0101	U10 liczby <i>B</i>
1001 1000 1011 0110 1100	1) Sumowanie liczby <i>A</i> i U10 liczby <i>B</i>
1001 1001 0001,0111 0010	2) Korekta liczb , które nie należą do kodu BCD ; na czerwono zaznaczono przeniesienia
	3) uzyskany wynik <i>A-B</i>
Przy sumowaniu liczby dodatniej i uzupełnienia innej liczby dodatniej nie ma przeniesienia, stąd wniosek, że wynik jest uzupełnieniem (wynikiem jest liczba ujemna). Liczba 1001, czyli 9 ₁₀ na początku wyniku mówi nam, że mamy do czynienia z uzupełnieniem – wynik jest ujemny!	
Rozkodowanie wyniku z U10	
1001 1001 0001,0111 0010 – wynik	Otrzymano liczbę ujemną.
0110 0110 0110 0110 0110 – korekta	1) Korygujemy od każdej tetrady dodając wartość 6₁₀ – 0110₂
1111 1111 0111,1101 1000	
0000 0000 1000,0010 1000	Rozkodowujemy, tworząc uzupełnienie U10 otrzymanego wyniku, czyli powinien wyjść wynik dodatni, czyli moduł otrzymanej liczby – wyniku
	2) zamieniamy zera na jedyne i jedyne na zera +1. Jest to dziesiętna wartość 8,28. Wynik jest ujemny, czyli wynosi <i>A-B</i> = (-8,28).

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład odejmowania - kod BCD z U10 – wynik >0

Należy wyliczyć różnicę $B-A$ posługując się uzupełnieniami (U10)

Dane są dwie liczby:

$A = 14,37$

$B = 22,65$

Kroki postępowania

Dwie liczby	
0000 0010 0010,0110 0101	Liczba <i>B</i> = 022,65 przedstawiona w kodzie BCD
0000 0001 0100,0011 0111	Liczba <i>A</i> = 014,37 przedstawiona w kodzie BCD
Tworzenie uzupełnienia U10 liczby <i>A</i>	
0110 0110 0110 0110 0110 – korekta	1) wymagana korekta – należy do każdej tetrady liczby <i>A</i> dodać wartość $6_{10} - 0110_2$ – liczba <i>A</i> po korekcie. 2) uzupełnienie U10 skorygowanej liczby <i>A</i> (zamiana zer na jedyne i jedynek na zera +1) → 985,63
0110 0111 1010,1001 1101	
1001 1000 0101,0110 0011	
Sumowanie liczby <i>B</i> i U10 liczby <i>A</i>	
0000 0010 0010,0110 0101	Liczba <i>B</i>
1001 1000 0101,0110 0011	Uzupełnienie U10 liczby <i>A</i>
1001 1010 0111 1100 1000	1) Sumowanie liczby <i>B</i> i U10 liczby <i>A</i>
0110 0110	2) Korekta liczb , które nie należą do kodu BCD ; na czerwono zaznaczono przeniesienia
1010 0000 1000,0010 1000	
0110	3) uzyskany wynik <i>B-A</i>
0000 0000 1000,0010 1000	
Przy sumowaniu liczby dodatniej i uzupełnienia innej liczby dodatniej jest przeniesienie (nadmiar, przepełnienie), stąd wniosek, że wynikiem jest liczba dodatnia. Liczba 0000, czyli 0_{10} na początku wyniku mówi nam, że mamy do czynienia z wynikiem dodatnim!	
0000 0000 1000,0010 1000	Jest to dziesiętna wartość <i>B-A</i> = 8,28, czyli wynik prawidłowy

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład działania - kod BCD z U10 $\rightarrow (-A)+(-B)$

Należy wyliczyć sumę $(-A)+(-B)$ posługując się uzupełnieniami (U10)

Dane są dwie liczby:

$A = 14,37$

$B = 22,65$

Kroki postępowania

Dwie liczby	
0000 0001 0100,0011 0111	Liczba $A = 014,37$ przedstawiona w kodzie BCD
Tworzenie uzupełnienia U10 liczby A	
0110 0110 0110 0110 0110 – korekta	1) wymagana korekta – należy do każdej tetrady liczby A dodać wartość $6_{10} - 0110_2$
0110 0111 1010,1001 1101	– liczba A po korekcie.
1001 1000 0101,0110 0011	2) uzupełnienie U10 skorygowanej liczby A (zamiana zer na jedynek i jedynek na zera +1) $\rightarrow 985,63$
Dwie liczby	
0000 0010 0010,0110 0101	Liczba $B = 022,65$ przedstawiona w kodzie BCD
Tworzenie uzupełnienia U10 liczby B	
0110 0110 0110 0110 0110 – korekta	1) wymagana korekta – należy do każdej tetrady liczby B dodać wartość $6_{10} - 0110_2$
0110 1000 1000,1100 1011	– liczba B po korekcie.
1001 0111 0111,0011 0101	2) uzupełnienie U10 skorygowanej liczby B (zamiana zer na jedynek i jedynek na zera +1) $\rightarrow 977,35$
Sumowanie U10 liczby A i U10 liczby B	
1001 1000 0101,0110 0011	Uzupełnienie U10 liczby A
1001 0111 0111,0011 0101	Uzupełnienie U10 liczby B
10010 1111 1100,1001 1000	1) Sumowanie U10 liczby A i U10 liczby B
10110 1111 0010,1001 1000	2) Korekty liczb , które nie należą do kodu BCD i tam, gdzie wystąpiło przeniesienie; na czerwono zaznaczono przeniesienia
1001 0000 0010,1001 1000	
0110	
1001 0110 0010,1001 1000	3) uzyskany wynik $(-A)+(-B)$
Liczba 1001 , czyli 9_{10} na początku wyniku mówi nam, że mamy do czynienia z uzupełnieniem. Wynik zatem jest ujemny.	
Rozkodowanie wyniku z U10	
1001 0110 0010,1001 1000 – wynik	Wynik – otrzymano liczbę ujemną.
0110 0110 0110 0110 0110 – korekta	1) Korygujemy od każdej tetrady dodając wartość $6_{10} - 0110_2$
1111 1100 1000,1111 1110	
0000 0011 0111,0000 0010	Rozkodowujemy, tworząc uzupełnienie U10 otrzymanego wyniku, czyli powinien wyjść wynik dodatni, czyli moduł otrzymanej liczby – wyniku
	2) zamieniamy zera na jedynek i jedynek na zera +1.
	Jest to dziesiętna wartość $37,02$. Wynik jest ujemny, czyli wynosi $(-A)+(-B) = (-37,02)$.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład odejmowania - kod EXCESS-3 z U9 – wynik <0 EXCESS-3

Należy wyliczyć różnicę $A-B$ posługując się uzupełnieniami (U9)

Dane są dwie liczby:

$$A = 14,37$$

$$B = 22,65$$

Kroki postępowania

Dwie liczby	
0000 0001 0100,0011 0111 0011 0011 0011 0011 (+3)	Liczba $A = 014,37$ przedstawiona w kodzie BCD
0011 0100 0111,0110 1010	Liczba $A = 014,37$ przedstawiona w EXCESS-3
0000 0010 0010,0110 0101 0011 0011 0011 0011 (+3)	Liczba $B = 022,65$ przedstawiona w kodzie BCD
0011 0101 0101,1001 1000	Liczba $B = 022,65$ przedstawiona w EXCESS-3
Tworzenie uzupełnienia U9 liczby B	
1100 1010 1010,0110 0111	Uzupełnienie U9 liczby $B \rightarrow 977,34$ (zamiana zer na jedyne i jedynek na zera)
Sumowanie liczby A i U9 liczby B	
0011 0100 0111,0110 1010 1100 1010 1010,0110 0111	Liczba A Uzupełnienie U9 liczby B
1111 1111 0001,1101 0001	1) Sumowanie liczby A i U9 liczby B ; na czerwono zaznaczono przeniesienia
-0011-0011+0011-0011+0011 – korekty 1100 1100 0100,1010 0100	2) Korekta liczb po dodawaniu 3) uzyskany wynik $A-B$
Przy sumowaniu liczby dodatniej i uzupełnienia innej liczby dodatniej nie ma przeniesienia, stąd wniosek, że wynik jest uzupełnieniem (wynikiem jest liczba ujemna). Liczba 1100, czyli 9_{10} na początku wyniku mówi nam, że mamy do czynienia z uzupełnieniem. Wynik zatem jest ujemny.	
Rozkodowanie wyniku z U9	
1100 1100 0100,1010 0100	Wynik – otrzymano liczbę ujemną.
0011 0011 1011,0101 1011 -0011-0011-0011-0011-0011 (-3) 0000 0000 1000,0010 1000	1) zamieniamy zera na jedyne i jedyne na zera. 2) Rozkodujemy, tworząc uzupełnienie U9 otrzymanego wyniku, czyli powinien wyjść wynik dodatni, czyli moduł otrzymanej liczby – wyniku Jest to dziesiętna wartość 8,28. Wynik jest ujemny, czyli wynosi $A-B = (-8,28)$.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład odejmowania - kod EXCESS-3 z U9 – wynik >0

Należy wyliczyć różnicę $B-A$ posługując się uzupełnieniami (U9)

Dane są dwie liczby:

$A = 14,37$

$B = 22,65$

Kroki postępowania

Dwie liczby	
0000 0010 0010,0110 0101 0011 0011 0011 0011 (+3)	Liczba $B = 022,65$ przedstawiona w kodzie BCD
0011 0101 0101,1001 1000	Liczba $B = 022,65$ przedstawiona w EXCESS-3
0000 0001 0100,0011 0111 0011 0011 0011 0011 (+3)	Liczba $A = 014,37$ przedstawiona w kodzie BCD
0011 0100 0111,0110 1010	Liczba $A = 014,37$ przedstawiona w EXCESS-3
Tworzenie uzupełnienia U9 liczby A	
1100 1011 1000,1001 0101	Uzupełnienie U9 liczby $A \rightarrow 985,62$ (zamiana zer na jedynki i jedynek na zera)
Sumowanie liczby B i U9 liczby A	
0011 0101 0101 1001 1000 1100 1011 1000 1001 0101 1 0000 0000 1110,0010 1101	Liczba B Uzupełnienie U9 liczby A
+0011+0011-0011+0011-0011 – korekty 0011 0011 1011,0101 1010 0011 0011 1011,0101 1011	1) Sumowanie liczby B i U9 liczby A ; na czerwono zaznaczono przeniesienia 2) Korekta liczb po dodawaniu Przeniesienie z najbardziej znaczącej cyfry – należy dodać do najmniej znaczącej cyfry 3) uzyskany wynik $B-A$
Przy sumowaniu liczby dodatniej i uzupełnienia innej liczby dodatniej jest przeniesienie (nadmiar, przepełnienie), stąd wniosek, że wynikiem jest liczba dodatnia. Liczba 0011, czyli 0_{10} na początku wyniku mówi nam, że mamy do czynienia z liczbą dodatnią	
Rozkodowanie wyniku	
0011 0011 1011,0101 1011 – wynik -0011-0011-0011-0011 (-3) 0000 0000 1000,0010 1000	Rozkodowujemy wynik. Jest to dziesiętna wartość $B-A = 8,28$, czyli wynik prawidłowy

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład działania - kod EXCESS-3 z U9 $\rightarrow (-A)+(-B)$

Należy wyliczyć sumę $(-A)+(-B)$ posługując się uzupełnieniami (U9)

Dane są dwie liczby:

$$A = 14,37$$

$$B = 22,65$$

Kroki postępowania

Dwie liczby	
0000 0001 0100,0011 0111 0011 0011 0011 0011 0011 (+3)	Liczba A = 014,37 przedstawiona w kodzie BCD
0011 0100 0111,0110 1010	Liczba A = 014,37 przedstawiona w EXCESS-3
Tworzenie uzupełnienia U9 liczby A	
1100 1011 1000,1001 0101	Uzupełnienie U9 liczby A $\rightarrow 985,62$ (zamiana zer na jedyne i jedynek na zera)
0000 0010 0010,0110 0101 0011 0011 0011 0011 0011 (+3)	Liczba B = 022,65 przedstawiona w kodzie BCD
0011 0101 0101,1001 1000	Liczba B = 022,65 przedstawiona w EXCESS-3
Tworzenie uzupełnienia U9 liczby B	
1100 1010 1010,0110 0111	Uzupełnienie U9 liczby B $\rightarrow 977,34$ (zamiana zer na jedyne i jedynek na zera)
Sumowanie uzupełnień U9 liczby B i U9 liczby A	
<div> <div>1100 1011 1000 1001 0101</div> <div>1100 1010 1010 0110 0111</div> <div>1 1001 0110 0010,1111 1100</div> <div>+0011+0011+0011-0011-0011 - korekty</div> <div>1100 1001 0101,1100 1001</div> <div>1100 1001 0101,1100 1010</div> </div>	<div>Uzupełnienie U9 liczby A</div> <div>Uzupełnienie U9 liczby B</div> <div>1) Sumowanie uzupełnień U9 liczby B i U9 liczby A; na czerwono zaznaczono przeniesienia</div> <div>2) Korekta liczb po dodawaniu</div> <div>Przeniesienie z najbardziej znaczącej cyfry – należy dodać do najmniej znaczącej cyfry</div> <div>3) uzyskany wynik $(-A)+(-B)$</div>
Liczba 1100, czyli 9 ₁₀ na początku wyniku mówi nam, że mamy do czynienia z uzupełnieniem. Wynik zatem jest ujemny.	
Rozkodowanie wyniku	
<div>1100 1001 0101,1100 1010 – wynik</div> <div>0011 0110 1010,0011 0101</div> <div>-0011-0011-0011-0011-0011 (-3)</div> <div>0000 0011 0111, 0000 0010</div>	<div>Rozkodowujemy, tworząc uzupełnienie U9 otrzymanego wyniku, czyli powinien wyjść wynik dodatni, czyli moduł otrzymanej liczby – wyniku</div> <div>(zamiana zer na jedyne i jedynek na zera)</div> <div>Jest to dziesiętna wartość 37,02. Wynik jest ujemny, czyli wynosi $(-A)+(-B) = (-37,02)$.</div>

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład odejmowania - kod EXCESS-3 z U10 – wynik <0 EXCESS-3

Należy wyliczyć różnicę $A-B$ posługując się uzupełnieniami (U10)

Dane są dwie liczby:

$$A = 14,37$$

$$B = 22,65$$

Kroki postępowania

Dwie liczby	
0000 0001 0100,0011 0111 0011 0011 0011 0011 0111 (+3)	Liczba $A = 014,37$ przedstawiona w kodzie BCD
0011 0100 0111,0110 1010	Liczba $A = 014,37$ przedstawiona w EXCESS-3
0000 0010 0010,0110 0101 0011 0011 0011 0011 0111 (+3)	Liczba $B = 022,65$ przedstawiona w kodzie BCD
0011 0101 0101,1001 1000	Liczba $B = 022,65$ przedstawiona w EXCESS-3
Tworzenie uzupełnienia U10 liczby B	
1100 1010 1010,0110 1000	Uzupełnienie U10 liczby $B \rightarrow 977,35$ (zamiana zer na jedynki i jedynki na zera +1)
Sumowanie liczby A i U10 liczby B	
0011 0100 0111,0110 1010 1100 1010 1010,0110 1000	Liczba A Uzupełnienie U10 liczby B
1111 1111 0001,1101 0010	1) Sumowanie liczby A i U10 liczby B ; na czerwono zaznaczono przeniesienia
-0011-0011+0011-0011+0011 – korekty 1100 1100 0100,1010 0101	2) Korekta liczb po dodawaniu 3) uzyskany wynik $A-B$
Przy sumowaniu liczby dodatniej i uzupełnienia innej liczby dodatniej nie ma przeniesienia, stąd wniosek, że wynik jest uzupełnieniem (wynikiem jest liczba ujemna). Liczba 1100, czyli 9_{10} na początku wyniku mówi nam, że mamy do czynienia z uzupełnieniem. Wynik zatem jest ujemny.	
Rozkodowanie wyniku z U10	
1100 1100 0100,1010 0101	Wynik – otrzymano liczbę ujemną.
0011 0011 1011,0101 1011 -0011-0011-0011-0011-0011 (-3) 0000 0000 1000,0010 1000	1) zamieniamy zera na jedynki i jedynki na zera +1. 2) Rozkodowujemy, tworząc uzupełnienie U10 otrzymanego wyniku, czyli powinien wyjść wynik dodatni, czyli moduł otrzymanej liczby – wyniku Jest to dziesiętna wartość 8,28. Wynik jest ujemny, czyli wynosi $A-B = (-8,28)$.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład odejmowania - kod EXCESS-3 z U10 – wynik >0

Należy wyliczyć różnicę $B-A$ posługując się uzupełnieniami (U10)

Dane są dwie liczby:

$A = 14,37$

$B = 22,65$

Kroki postępowania

Dwie liczby	
0000 0010 0010,0110 0101 0011 0011 0011 0011 0011 (+3)	Liczba $B = 022,65$ przedstawiona w kodzie BCD
0011 0101 0101,1001 1000	Liczba $B = 022,65$ przedstawiona w EXCESS-3
0000 0001 0100,0011 0111 0011 0011 0011 0011 0011 (+3)	Liczba $A = 014,37$ przedstawiona w kodzie BCD
0011 0100 0111,0110 1010	Liczba $A = 014,37$ przedstawiona w EXCESS-3
Tworzenie uzupełnienia U10 liczby A	
1100 1011 1000,1001 0110	Uzupełnienie U10 liczby $A \rightarrow 985,63$ (zamiana zer na jedynki i jedynek na zera +1)
Sumowanie liczby B i U10 liczby A	
0011 0101 0101 1001 1000 1100 1011 1000 1001 0110 1 0000 0000 1110,0010 1110	Liczba B Uzupełnienie U10 liczby A
	1) Sumowanie liczby B i U10 liczby A ; na czerwono zaznaczono przeniesienia Przeniesienie z najbardziej znaczącej cyfry należy pominąć
+0011+0011-0011+0011-0011 – korekty 0011 0011 1011,0101 1011	2) Korekta liczb po dodawaniu 3) uzyskany wynik $B-A$
Przy sumowaniu liczby dodatniej i uzupełnienia innej liczby dodatniej jest przeniesienie (nadmiar, przepełnienie), stąd wniosek, że wynikiem jest liczba dodatnia. Liczba 0011, czyli 0_{10} na początku wyniku mówi nam, że mamy do czynienia z liczbą dodatnią	
Rozkodowanie wyniku	
0011 0011 1011,0101 1011 – wynik -0011-0011-0011-0011-0011 (-3) 0000 0000 1000,0010 1000	Rozkodowujemy wynik. Jest to dziesiętna wartość $B-A = 8,28$, czyli wynik prawidłowy

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykład działania - kod EXCESS-3 z U10 $\rightarrow (-A)+(-B)$

Należy wyliczyć sumę $(-A)+(-B)$ posługując się uzupełnieniami (U10)

Dane są dwie liczby:

$A = 14,37$

$B = 22,65$

Kroki postępowania

Dwie liczby	
0000 0001 0100,0011 0111 0011 0011 0011 0011 0011 (+3)	Liczba $A = 014,37$ przedstawiona w kodzie BCD
0011 0100 0111,0110 1010	Liczba $A = 014,37$ przedstawiona w EXCESS-3
Tworzenie uzupełnienia U10 liczby A	
1100 1011 1000,1001 0110	Uzupełnienie U10 liczby $A \rightarrow 985,63$ (zamiana zer na jedynki i jedynek na zera +1)
0000 0010 0010,0110 0101 0011 0011 0011 0011 0011 (+3)	Liczba $B = 022,65$ przedstawiona w kodzie BCD
0011 0101 0101,1001 1000	Liczba $B = 022,65$ przedstawiona w EXCESS-3
Tworzenie uzupełnienia U10 liczby B	
1100 1010 1010,0110 1000	Uzupełnienie U10 liczby $B \rightarrow 977,35$ (zamiana zer na jedynki i jedynek na zera +1)
Sumowanie uzupełnień U10 liczby B i U10 liczby A	
1100 1011 1000 1001 0110 1100 1010 1010 0110 1000 1 1001 0110 0010,1111 1110	Uzupełnienie U10 liczby A Uzupełnienie U10 liczby B 1) Sumowanie uzupełnień U10 liczby B i U10 liczby A ; na czerwono zaznaczono przeniesienia Przeniesienie z najbardziej znaczącej cyfry należy pominąć
+0011+0011+0011-0011-0011 – korekty 1100 1001 0101,1100 1011	2) Korekta liczb po dodawaniu 3) uzyskany wynik $(-A)+(-B)$
Liczba 1100, czyli 9_{10} na początku wyniku mówi nam, że mamy do czynienia z uzupełnieniem. Wynik zatem jest ujemny.	
Rozkodowanie wyniku	
1100 1001 0101,1100 1011 – wynik 0011 0110 1010,0011 0101 -0011-0011-0011-0011-0011 (-3) 0000 0011 0111, 0000 0010	Rozkodowujemy, tworząc uzupełnienie U10 otrzymanego wyniku, czyli powinien wyjść wynik dodatni, czyli moduł otrzymanej liczby – wyniku (zamieniamy zera na jedynki i jedynki na zera +1.) Jest to dziesiętna wartość 37,02. Wynik jest ujemny, czyli wynosi $(-A)+(-B) = (-37,02)$.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Mnożenie BCD

67 → 0110 0111_{BCD} konwertowanie binarnych na BCD

· 6 → 0110_{BCD}

42 101010₂ →

36 100100₂ → 0100 0010_{BCD}

302 + 0011 0110_{BCD}

3011 0011 1010 0010_{BCD}

0110 korekta

0011 0000 0010_{BCD}

1 0100 0000 0010_{BCD}

402₁₀

Dzielenie w kodzie BCD

1354₁₀ : 5₁₀

0101 < 0001 0011 0101 0101

Dzielnik mniejszy niż dzielna?

NIE

(jeżeli TAK, to zakończ)

Wybierz lewą tetradę

Q4	Q3	Q2	Q1
0000	0000	0000	0000
0001	0011	0101	0100
0101			

Wynik częściowy

Dzielna

Wyrównaj dzielnik z tetradą

(0101 ≤ 0001)?

Dzielnik mniejszy od tetrady?

NIE – zatem zostaw wartość 0000 w pierwszej tetradzie wyniku

Q4	Q3	Q2	Q1
0000	0000	0000	0000
0001	0011	0101	0100
	0101		

Wybierz kolejną tetradę

Wynik częściowy

Wyrównaj dzielnik z tetradą

(0101 ≤ 1 0011)?

Dzielnik mniejszy od tetrady i pozostałości?

TAK – zatem zwiększ drugą tetradę wyniku o 0001

Q4	Q3	Q2	Q1
0000	0001	0000	0000
0001	0011	0101	0100

Wynik częściowy

Dzielna

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

- 0101
1000

Pomniejsz tetradę i pozostałość o wartość dzielnika

Q4	Q3	Q2	Q1
0000	0010	0000	0000
0001	0011	0101	0100
-	0101		
	1000		

Wynik częściowy

Dzielnik

- 0101

1000

- 0101

0011

Dzielnik mniejszy od otrzymanej różnicy?
0101 < 0 1000)?
TAK – zatem zwiększ drugą tetradę wyniku o 0001
Pomniejsz otrzymaną różnicę o wartość dzielnika
Wykonuj tak długo, aż dzielnik będzie mniejszy od otrzymanego wyniku

Q4	Q3	Q2	Q1
0000	0010	0000	0000
0001	0011	0101	0100

Wybierz kolejną tetradę

Wynik częściowy

Dzielnik

...

0011

0101

Pozostałość poprzedniego odejmowania

Wyrównaj dzielnik z tetradą

(0101 ≤ 0011 0101)?

Dzielnik mniejszy od tetrady i pozostałości?
TAK – zatem zwiększ drugą tetradę wyniku o 0001
Wybierz kolejną tetradę

Q4	Q3	Q2	Q1
0000	0010	0001	0000
0001	0011	0101	0100

Wynik częściowy

Dzielnik

...

0011 0101

- 0101

0011 0000

- 0101

0010 0101

- 0101

itd.

Pozostałość poprzedniego odejmowania

Pomniejsz tetradę i pozostałość o wartość dzielnika

Przeniesienie z wcześniejszej tetrady

Pomniejsz tetradę i pozostałość o wartość dzielnika

Dzielnik mniejszy od tetrady i pozostałości? (0101 < 0010 0101)?
TAK – zatem zwiększ drugą tetradę wyniku o 0001
Wykonuj tak długo, aż dzielnik będzie mniejszy od otrzymanego wyniku

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Q4	Q3	Q2	Q1	Wybierz kolejną tetradę
0000	0010	0111	0000	Wynik cząstkowy
0001	0011	0101	0100	Dzielną
...				
0000				Pozostałość poprzedniego odejmowania
-				Pomniejsz tetradę i pozostałość o wartość dzielnika
				NIE – zatem zostaw wartość 0000 w pierwszej tetradzie wyniku
0100				Przepisz pozostałość z ostatniej tetrady – to reszta z dzielenia

Uzyskany wynik z dzielenia $1354_{10} : 5_{10} = 270_{10}$ reszta 4_{10}

Liczby zmiennopozycyjne

Wprowadzenie do reprezentacji zmiennopozycyjnej liczb

W systemach **stałoprzecinkowych** punkt oddzielający część całkowitą liczby od części ułamkowej jest jednoznacznie **na stałe określony**. Może również być przyjęte, że liczba zakodowana w bitach przedstawia liczbę dziesiętną, którą należy podzielić np. 4 ostatnie (najmniej znaczące) cyfry odpowiadają za część ułamkową, natomiast pozostałe za część całkowitą. Przecinek oddzielający część całkowitą od ułamkowej ma jednoznacznie określoną, stałą pozycję, stąd nazwa liczb – stałopozycyjne.

Reprezentacja zmiennopozycyjna wprowadzono, aby **zwiększyć dokładność obliczeń**.

Jeżeli mamy stałą długość słowa (czyli wykorzystujemy stale tyle samo bitów), to odległości pomiędzy kolejnymi/sąsiadującymi liczbami są stałe. Jest to największa dokładność odwzorowania liczb.

Najogólniej liczby zmiennopozycyjne to takie, w których będziemy prezentowali określoną liczbę cyfr znaczących, znak tej liczby oraz wykładnik, który będzie wykorzystywany do **skalowania** tej liczby.

W wyniku przyjętego sposobu prezentowania liczb można powiedzieć, że reprezentacja zmiennoprzecinkowa jest podobna w koncepcji do **zapisu naukowego**.

Liczby zmiennoprzecinkowe przedstawia się jako jedna cyfra przed przecinkiem, następnie część ułamkowa, a następnie cała ta liczba jest mnożona razy podstawa do potęgi równej wykładnikowi.

Przykładowo liczba 543,21 może być przedstawiona jako

$$543,21 = 5,4321 \cdot 10^2$$

lub

$$543,21 = 54321 \cdot 10^{-2}.$$

Zapis zmiennoprzecinkowy wymaga użycia dużo mniejszej liczby znaków do przedstawienia tych samych liczb o modułach bardzo dużych i bardzo małych

Przykład:

$$1.200.000.000.000 = 1,2 \cdot 10^{12}$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

$$0,000\,000\,005 = 5,0 \cdot 10^{-9}$$

$$4.010.000.000.000.000.000.000 = 4,01 \cdot 10^{21}$$

Reprezentacja liczby zmiennopozycyjnej

$$X = S \cdot M \cdot B^E$$

S (ang. *sign*) – znak liczby

M (ang. *mantissa*) – znormalizowana mantysa, rozumiana jako liczba ułamkowa

B (ang. *base*) – podstawa pozycyjnego systemu liczbowego

E (ang. *exponent*) – wykładnik (liczba całkowita)

Zapis zmiennopozycyjny (zmiennoprzecinkowy) przedstawia liczby za pomocą **trzech** słów binarnych:

- Jednobitowe słowo **znaku Z**, jest reprezentowany przez pojedynczy bit, który dla liczby dodatniej ma wartość 0, natomiast dla liczby ujemnej – wartość 1.
- *n*-bitowe słowo tzw. **mantysy S** (ang. *significant* lub *mantissa*);
- *m*-bitowe słowo **wykładnika E** (ang. *exponent*), Wykładnik jest określony przy danej podstawie lub danej bazie.

Baza, czyli **podstawa** (ang. *base*) dla całego systemu zmiennopozycyjnego jest taka sama, dlatego nie musi być przechowywana. To tak jak w systemie stałopozycyjnym dla każdej liczby miejsce oddzielenia cyfr dla liczby całkowitej od cyfr dla części ułamkowej jest stałe i nie musi być przechowywane w każdej zapisywanej liczbie. Jako baza wykładnika w systemach dziesiętnych jest stosowana liczba 10. W systemach binarnych jest ona równa 2.

Liczba dwójkowa 1001,1001 może być zapisana jako

- $0,10011001 \cdot 2^4$
- $1,0011001 \cdot 2^3$
- $10,011001 \cdot 2^2$
- $10011001 \cdot 2^{-4}$

To pokazuje, że warto byłoby określić, w sposób jednoznaczny, jaki zapis mantyzы jest właściwy.

Mantysa jest **znormalizowana** jeżeli ma wartość z przedziału $1 \leq M < B$ (*B* – to podstawa systemu liczbowego, *M* – mantysa). Tu jest to przedział jednostronnie otwarty. W przypadku dziesiętnego systemu liczbowego **znormalizowana mantysa** będzie mieścić się w granicach pomiędzy [1,0 , 10,0) – przedział jednostronnie otwarty. Jeżeli w skutek wykonywania jakichś działań mantysa nie mieści się w ww. przedziale np. jest większa → 203,5555 lub mniejsza → 0,000234 należy dokonać **normalizacji mantysy**, czyli sprowadzenia jej do wymaganej postaci poprzez zmianę wykładnika – w pierwszym przypadku jego zwiększenia, a w drugim – zmniejszenia.

Dziesiętna liczba zmiennoprzecinkowa

Warto przyjrzeć się, jak są przedstawiane dziesiętne liczby zmiennoprzecinkowe:

$$-5,127 \cdot 10^6$$

Mantysa wynosi 5,127, czyli każdą kolejną cyfrę liczby mnoży się razy kolejne potęgi liczby 10

$$5,127 = 5 \cdot 10^0 + 1 \cdot 10^{-1} + 2 \cdot 10^{-2} + 7 \cdot 10^{-3}$$

$$-5,127 \cdot 10^6 = -5 \underbrace{127 000}$$

Przesunięcie przecinka
o 6 pozycji w prawo (bo $\cdot 10^6$)

Binarna liczba zmiennoprzecinkowa

- Format binarnej liczby zmiennoprzecinkowej jest analogiczny. Do postaci liczby dziesiętnej:

$$-1,10101 \cdot 2^4 = -1,10101 \cdot 10_2^{100}$$

Mantysa wynosi 1,10101, czyli każdą kolejną cyfrę liczby mnoży się razy kolejne potęgi liczby 2

$$1,10101 = 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5}$$

$$-1,10101 \cdot 2^4 = -1 \underbrace{1010,1}$$

Przesunięcie przecinka
o 4 pozycji w prawo (bo $\cdot 2^4$)

Jeżeli mantysa jest taka sama, a **zmianie ulega wykładnik**, wtedy **zmienia się miejsce przecinka** – przesuwamy się. Stąd nazwa liczb – zmiennoprzecinkowe. Nazwa więc pochodzi stąd, że **przecinek** dziesiętny (kropka, w komputerach tzw. punkt binarny) **może zmieniać miejsce**, to znaczy może być przedstawiony w dowolnym miejscu względem cyfr znaczących.

UWAGA! Stosowanie w zapisie binarnych liczb zmiennoprzecinkowych cyfry 2 jako podstawy tego systemu i podnoszenie jej do określonej potęgi może być mylące. Niekonsekwencja wynika z tego, że cyfra 2 nie należy do binarnego systemu liczbowego. Zamiast 2^4 powinien być stosowany zapis $(10_2)^4$. Jednak dla zwiększenia czytelności przedstawiania liczb w niektórych częściach tego opracowania wykorzystywana jest cyfra 2.

Mantysa

Mantysa składa się z cyfr znaczących, które jednak mogą zarówno oznaczać liczbę całkowitą, jak również ułamek, w zależności od wartości wykładnika. Mantysa prezentuje najbardziej znaczące cyfry liczby. W zależności od wykładnika mogą one oznaczać liczbę całkowitą lub ułamek. W niektórych językach programowania (np. C, Fortran) wykorzystuje się mantysę w zapisie 0,54321, czyli jej wartość mieści się

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

pomiędzy 0,1 a 1,0. Wtedy
 $543,21 = 0,54321 \cdot 10^3$

Jeżeli występuje tzw. bit ukryty, to przekłada się to na pewne **ograniczenia**.
Przykładowo liczba zero nie może być reprezentowana w tym formacie.

Bit ukryty

Znormalizowany format mantysy w przypadku podstawy 2 wynosi 1,xxxxxxx, czyli wartość mantysy mieści się w granicach od 1,0 do 2,0, nie osiągając wartości 2,0. W znormalizowanej mantysie pierwszy bit (najbardziej znacząca cyfra mantysy) ma wartość niezerową – w systemach binarnych zawsze jest równy 1. Zatem nie musi być on zapamiętywany i wtedy jest nazywany **ukrytym bitem**.

Przykładowo liczba binarna:

1,0001000111111011

Będzie zapisywana w bitach mantysy jako

0001 0001 1111 1011 0000 000

Czyli z pominięciem pierwszej jedynek.

Zatem na bitach mantysy przechowywana jest tylko część ułamkowa liczby (po znormalizowaniu mantysy).

Dodatkowo brakujące liczby (aby zapełnić 23 bity mantysy) zostały zapisane zerami.

Wykładnik

Wykładnik, cecha – liczba całkowita, reprezentuje potęgę, do której należy podnieść podstawę i pomnożyć mantysę (i znak), aby otrzymać wartość liczby. Można powiedzieć, że wykładnik wyznacza mnożną, przez którą należy pomnożyć liczbę określoną przez mantysę.

Jeżeli zmienia się wartość wykładnika, przesuwa się przecinek – najprawdopodobniej stąd pochodzi nazwa tych liczb – zmiennoprzecinkowe. Dzięki zmianie wartości wykładnika można **dynamicznie zmieniać zakres** reprezentowanych liczb.

Różnica pomiędzy dwoma kolejnymi liczbami rośnie wraz ze wzrostem wartości wykładnika.

Chociaż wykładnik może być dodatni albo ujemny, w formatach binarnych jest przechowywany jako liczba bez znaku, do której dodano stałe „odchylenie” – przesunięcie binarne (ang. *bias*).

Wykładnik, który jest liczbą całkowitą, zapisywany jest w kodzie spolaryzowanym, czyli jest to wartość przesunięta o pewną stałą (ang. *biased exponent*). Właściwą wartość wykładnika wyznacza się odejmując od wykładnika zapisanego w kodzie wartość przesunięcia. Wartości wszystkich zer w polu wykładnika są zarezerwowane dla zer (ze względu na istnienie ukrytego bitu) i wartości szczególnych.

Zerowe wartości wykładnika zarezerwowane są dla przedstawiania wartości specjalnych.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Skończony zbiór kombinacji mantysy i wykładnika

Liczba cyfr mantysy i wykładnika (bitów) przeznaczonych na reprezentację wykładnika i mantysy **jest ustalona, skończona** i jednoznacznie określona. Dlatego też, można powiedzieć, że występuje tu **skończona liczba możliwych wykładników i mantys** prezentowanych liczb. Istnieje także skończona liczba reprezentacji liczb (skończony zbiór wartości), jak również liczby przedstawia się ze skończoną dokładnością. Może to powodować powstawanie błędów obliczeniowych i dalej ma wpływ na występowanie pewnych ograniczeń i właściwości działań arytmetycznych.

Można zatem powiedzieć, że istnieją skończone zbiory wartości reprezentujących wartości wykładnika i mantysy.

Mantysa (m – liczba cyfr przeznaczona na mantysę)

$$M_{min} = 1$$

$M_{max} = B - B^{-(m-1)}$ – czyli dla liczb binarnych 2 minus najmniejszy bit (same jedynki mantysy)

Wykładnik $\{n+1$ to liczba cyfr przeznaczonych na wykładnik (n cyfr dla wartości i 1 dla znaku wykładnika) $\}$

$$E_{min} = -B^n + 1$$

$$E_{max} = B^n - 1$$

Najmniejszy moduł liczby do przedstawienia:

$$x_{min} = M_{min} \cdot B^{E_{min}} = 1 \cdot B^{E_{min}}$$

Największy moduł liczby do przedstawienia:

$$x_{max} = M_{max} \cdot B^{E_{max}} = (B - B^{-(m-1)})B^{E_{max}} < B^{E_{max}+1}$$

Zakres liczb możliwych do przedstawienia w takim zapisie to:

$$[-x_{max}, -x_{min}] \cup \{0\} \cup [x_{min}, x_{max}]$$

Niedomiar, nadmiar

Jeżeli komputer miałby zapisać liczbę:

$$|x| < B^{E_{min}}$$

To traktowana jest ona jako niedomiar (ang. *underflow*)

Natomiast dla liczb

$$|x| > M_{max} \cdot B^{E_{max}}$$

traktowana jest ona jako tzw. nadmiar wykładniczy (ang. *overflow*).

Liczby szczególne, „ponadnormalne” (IEEE 754)

Można wyróżnić następujące przypadki liczb szczególnych:

- **+0** – bit znaku = 0, wszystkie bity wykładnika = 0, wszystkie bity mantysy = 0,
- **-0** – bit znaku = 1, wszystkie bity wykładnika = 0, wszystkie bity mantysy = 0,
- **$\pm\infty$** (nieskończoność) – **wszystkie bity wykładnika mają wartość 1**, natomiast **mantysa = 0**, bit znaku mówi, jaka nieskończoność występuje; taka wartość może pojawić się w przypadku wystąpienia nadmiaru (przepełnienia), choćby jako wynik dzielenia liczby przez 0,
- **NaN** (ang. *Not a number*) – bit znaku nie ma znaczenia – ma wartość 0 albo 1, **wszystkie bity wykładnika mają wartość 1**, natomiast **mantysa** ma wartość dowolną, ale różną od zera, żeby odróżnić ją od wartości plus nieskończoności oraz minus nieskończoności; Taka liczba może wystąpić jako wynik np. pierwiastkowania liczby ujemnej.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

- **Liczby bardzo małe, liczba nieznormalizowana, niedomiar zmiennoprzecinkowy** – występuje wtedy, kiedy zdarzy się niedomiar (ang. *underflow*), wykładnik równy -127 (czyli najmniejszy możliwy do przedstawienia); znak mówi o tym, czy jest to wartość dodatnia, czy ujemna, wszystkie **bity wykładnika mają wartości 0, mantysa różna od zera** (żeby odróżnić od reprezentacji wartości zero), natomiast w tej reprezentacji nie zakłada się istnienia niezerowego, ukrytego bitu, czyli reprezentuje liczbę w postaci 0,xxx...xxx.

Nie-liczba NaN

Nie-liczba (ang. *not a number*) – oznacza, że w wyniku wykonywania pewnych działań arytmetycznych uzyskano nie liczbę lub niereprezentowalną wartość. Tak jak w wyniku dzielenia przez zero można wyobrazić sobie jako wynik plus nieskończoność lub minus nieskończoność, to w wyniku wykonywania działania 0/0 nie uzyska się liczby, nie można wyobrazić sobie rzędu ani otoczenia takiej wartości. Analogicznie wartości 0 podniesionego do potęgi zerowej.

Analogicznie pierwiastek z liczby ujemnej lub logarytm liczby ujemnej jest wartością urojoną, jednak jako taka nie jest reprezentowana w prezentowanym na wykładzie zapisie zmiennoprzecinkowym. Dlatego też otrzymana wartość jest rozumiana jako nie-liczba.

Również arcus sinus lub arcus cosinus liczby mniejszej niż -1 lub większej niż 1.

Niedomiar zmiennoprzecinkowy – liczby zbyt małe

Liczby bardzo małe, liczba nieznormalizowana, niedomiar zmiennoprzecinkowy – występuje wtedy, kiedy zdarzy się niedomiar (ang. *underflow*). Liczby zbyt małe (niedomiar) to każda liczba niezerowa, której moduł wartości mieści się pomiędzy wartością zero, a najmniejszym modułem wartości możliwym do przedstawienia w danym kodzie zmiennopozycyjnym. Dzięki temu w wyniku działań np. nie dochodzi do dzielenia przez zero będące zaokrągleniem otrzymanego wyniku, tylko przez bardzo małą liczbę.

Jest to próba wypełnienia luki w systemach zmiennopozycyjnych. Wystąpienie takiego niedomiaru nie jest błędem krytycznym, zmniejsza jednak dokładność obliczeń. Najczęściej zwracana jest jako wartość zero lub jako wartość nieznormalizowana. W tym przypadku wykładnik musiałby być poniżej wartości minimalnej dopuszczalnej w danym systemie zmiennopozycyjnym.

Różne reprezentacje liczb a standard

Przez wiele lat w komputerach wykorzystywano różne reprezentacje liczb zmiennopozycyjnych. Od lat dziewięćdziesiątych najczęściej spotykaną reprezentacją jest taka, zdefiniowana w standardzie IEEE 754.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

reguła bitowa nie jest używana), a jego precyzja wynosi co najmniej 64 bity (około 19 cyfr dziesiętnych). Format spełniający minimalne wymagania (64-bitowa precyzja, 15-bitowy wykładnik, a więc dopasowany do 80 bitów) jest zapewniany przez architekturę x86.

Nazwa	Rodzaj	Znak [bit]	Wykładnik potęgowy [bity]	Mantysa [bity]	Szerokość słowa Łączna liczba bitów	Przesunięcie wykładnika BIAS	Bity precyzji (mantysa + ukryty)	Liczba cyfr dziesiętnych
binary16	Półowa (IEEE 754-2008)	1	5	10	16	15	11	~ 3,3
binary32	Pojedyncza precyzja	1	8	23	32	127	24	~ 7,2
binary64	Podwójna precyzja	1	11	52	64	1023	53	~ 15,9
	Rozszerzona precyzja x86	1	15	64	80	16383	64	~ 19,2
binary128	Poczwórna precyzja	1	15	112	128	16383	113	~ 34,0
binary256	Ośmiokrotna precyzja	1	19	236	256	262143	237	

Standard IEEE 754 – podwójna precyzja

Liczby podwójnej precyzji (ang. *double precision*) w tym standardzie zapisywane są na 64 bitach (8 bajtach). Pierwszy bit to **1 bit znaku S** (ang. *sign*), dla liczby dodatniej jest on równy 0, natomiast dla ujemnej ma wartość 1. Kolejne **11 bitów jest to wykładnik**/cecha, jest ona kodowana w kodzie spolaryzowanym (BIAS = 1023), dzięki czemu otrzymuje się zakres wykładników od -1023 (wszystkie bity wykładnika są równe 0) do 1024 (każdy bit wykładnika = 1). Dodatkowo najmniejsza i największa wartość wykładników mają wartość specjalną. Dalej występują **52 bity mantysy**.

Zakłada się, że liczba jest zapisywana na 53 bitach, przy czym jeden z nich (ten, który powinien być najstarszy), niezerowy, czyli ten, który ma zawsze wartość 1, tzw. **bit ukryty** jest pomijany. Otrzymuje się zatem ok. 7-8 dziesiętnych miejsc znaczących liczby.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”**Arytmetyka zmiennopozycyjna**

Arytmetyka zmiennopozycyjna jest pewnego rodzaju kompromisem przy reprezentowaniu liczb rzeczywistych, pomiędzy liczbą prezentowanych bitów, zakresem a precyzją działań. Tego typu rozwiązania są tak jakby dedykowane przy wykonywaniu działań na bardzo małych lub bardzo dużych liczbach, przy jednoczesnym zachowaniu krótkiego czasu przetwarzania tych liczb.

Przykładowo za pomocą takiego samego zapisu liczb można dokonywać działań arytmetycznych na liczbach z zupełnie innego zakresu wartości np. opisujących średnicę jądra atomowego i średnicę gwiazd lub odległość między nimi.

Dodawanie liczb zmiennopozycyjnych

Podczas dodawania liczb zmiennopozycyjnych w pierwszym kroku należy wyrównać wykładniki dwóch liczb. Mantysa liczby o mniejszym wykładniku (mniejszej liczby) jest zmniejszana (przesuwana w prawo na mniej znaczące pozycje o tyle pozycji, ile wynika z konieczności wyrównania wykładników) = zwiększany jest wykładnik, aby wyrównać go z wykładnikiem liczby większej.

Podczas przesuwania mantysy nowa (przesunięta mantysa) reprezentowana jest przez tyle samo bitów. Zatem istnieje ryzyko gubienia bitów, które po przesunięciu mantysy „w prawo” nie będą miały miejsca, gdzie mogłyby być reprezentowane.

Liczbę można zamieniać kolejnością oraz znakiem liczb wraz z rodzajem wykonywanego działania

Możliwe są różne kombinacje:

$(+x_1) + (+x_2); (+x_1) - (-x_2); (+x_2) + (+x_1); (+x_2) - (-x_1); \rightarrow (x_1 + x_2);$
 $(+x_1) + (-x_2); (+x_1) - (+x_2); (-x_2) + (+x_1); (-x_2) - (-x_1); \rightarrow (x_1 - x_2);$
 $(-x_1) + (-x_2); (-x_1) - (+x_2); (-x_2) + (-x_1); (-x_2) - (+x_1); \rightarrow -(x_1 + x_2);$
 $(-x_1) + (+x_2); (-x_1) - (-x_2); (+x_2) + (-x_1); (+x_2) - (+x_1); \rightarrow -(x_1 - x_2);$

Dodawanie i odejmowanie liczb zmiennoprzecinkowych w ogólności można byłoby zapisać wzorem:

$$x_1 \pm x_2 = (M_1 \pm M_2 \cdot B^{(E_2 - E_1)}) \cdot B^{E_1}$$

Jeżeli liczby mają różne wykładniki, to mantysa jednej z liczb (tej o mniejszym wykładniku) musi zostać zdenormalizowana, poprzez pomnożenie razy $B^{(E_2 - E_1)}$. Jeżeli wykładniki są takie same, to w wyniku odejmowania uzyska się: $E_2 - E_1 = 0$.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykłady dodawania dwóch liczb

Działanie na liczbach pojedynczej precyzji

$$X = 1,011001101010 \cdot 2^{10}$$

$$Y = 1,011011011 \cdot 2^0$$

$$X + Y = 1,011001101010 \cdot 2^{10} + 1,011011011 \cdot 2^0 = ?$$

1) Zapis liczb w mantysie

Bity ukryte, nie zapisywane w mantysie

Bity mantysy:

X	Bity liczby													Uzupełnienie zerami															
1	0	1	1	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Y	Bity liczby										Uzupełnienie zerami																
1	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2) Wyrównywanie wykładników

Konieczność przesunięcia cyfr w mantysie drugiej liczby o 10 pozycji

$$1,011001101010 \cdot 2^{10} + 0,0000000001011011011 \cdot 2^{10} = ?$$

Przesunięty bit ukryty

Zapis liczb w mantysie

Bity mantysy:

X	Bity liczby												Uzupełnienie zerami															
1	0	1	1	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

mantysa przesunięta o 10 pozycji aby wyrównać wykładniki liczb

Y	Uzupełnienie zerami										Bity liczby – przesunięte cyfry										zera					
0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	1	1	0	0	0	0	0	

+

1	0	1	1	0	0	1	1	0	1	1	1	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Otrzymany wynik

$$1,011001101010 \cdot 2^{10} + 1,011011011 \cdot 2^0 = 1,011001101111011011 \cdot 2^{10}$$

Inaczej ten przykład można przedstawić w następujący sposób:

$$X = 1,011001101010 \cdot 2^{10}$$

$$Y = 1,011011011 \cdot 2^0$$

Mantysa liczby $X = 1,011001101010$

Wykładnik liczby $X = 10001001$

$$X_{IEEE} = 0 \ 10001001 \ (1) \ 0110 \ 0110 \ 1010 \ 0000 \ 0000 \ 000$$

Mantysa liczby $Y = 1,011011011$

Wykładnik liczby $Y = 01111111$

$$Y_{IEEE} = 0 \ 01111111 \ (1) \ 0110 \ 1101 \ 1000 \ 0000 \ 0000 \ 000$$

$$Y_{IEEE} = 0 \ 10001001 \ (0) \ 0000 \ 0000 \ 0101 \ 1011 \ 0110 \ 000 - \text{wyrównanie wykładników}$$

$$X_{IEEE} = 0 \ 10001001 \ (1) \ 0110 \ 0110 \ 1010 \ 0000 \ 0000 \ 000$$

$$X+Y = 0 \ 10001001 \ (1) \ 0110 \ 0110 \ 1111 \ 1011 \ 0110 \ 000 - \text{suma}$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Inny przykład, bardziej skomplikowany

$$X = 65535_{10} = 1111111111111111_2 = 1,111111111111111_2 \cdot 10^{1111} (\rightarrow 2^{15})$$

$$Y = 2 \frac{3}{512} = 2,0058593750_{10} = 10,000000011_2 = 1,0000000011_2 \cdot 10^1 (\rightarrow 2^1)$$

$$X + Y = 6,5535 \cdot 10^4 + 2,0058593750 \cdot 10^0 = 65535 + 2 \frac{3}{512} = ?$$

X	Bity znaczące															zera									
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0

Y	Bity znaczące										zera														
1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Wyrównywanie wykładników
stracony

Ten bit nie zmieścił się w mantysie, więc będzie

	Bity znaczące															zera									
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0

+																Przesunięte bity znaczące									
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1

Uzyskany wynik dodawania
1 – bit do przeniesienia

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Normowanie mantysy, czyli przesunięcie bitów o 1 pole w lewo

	bity znaczące																								
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

Ten bit nie zmieścił się w mantysie po jej normowaniu, więc będzie stracony

Otrzymany wynik (część ułamkowa została stracona)

$$X + Y = 6,5565 \cdot 10^4 + 2,0058593750 \cdot 10^0 = 65565 + 2 \frac{3}{512} = 65537$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Inaczej ten przykład można przedstawić w następujący sposób:

$$X = 65535_{10} = 111111111111111_2 = 1,11111111111111_2 \cdot 10^{1111} (\rightarrow 2^{15})$$

$$Y = 2,0058593750_{10} = 10,0000000011_2 = 1,00000000011_2 \cdot 10^1 (\rightarrow 2^1)$$

$$\text{Mantysa liczby } X = 1,11111111111111_2 \cdot 2^{15}$$

$$\text{Wykładnik liczby } X = 10001110$$

$$X_{IEEE} = 0 \ 10001110 \ (1) \ 1111 \ 1111 \ 1111 \ 1110 \ 0000 \ 000$$

$$\text{Mantysa liczby } Y = 1,000000011$$

$$\text{Wykładnik liczby } Y = 10000000$$

$$Y_{IEEE} = 0 \ 10000000 \ (1) \ 0000 \ 0000 \ 1100 \ 0000 \ 0000 \ 000$$

$$Y_{IEEE} = 0 \ 10001110 \ (0) \ 0000 \ 0000 \ 0000 \ 0100 \ 0000 \ 0011 \ 1 - \text{wyrównanie wykładników}$$

$$X_{IEEE} = 0 \ 10001110 \ (1) \ 1111 \ 1111 \ 1111 \ 1110 \ 0000 \ 000$$

$$X+Y = 0 \ 10001110 \ 1(0) \ 0000 \ 0000 \ 0000 \ 0010 \ 0000 \ 001 - \text{konieczność normalizowania mantysy bo jest przeniesienie}$$

$$X+Y = 0 \ 10001111 \ (1) \ 0000 \ 0000 \ 0000 \ 0001 \ 0000 \ 0001 - \text{wynik końcowy}$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przykłady odejmowania dwóch liczb z użyciem uzupełnienia U9

Działanie na liczbach pojedynczej precyzji

$$X = 1,01010011100 \cdot 2^8 = 339,5_{10}$$

$$Y = 1,011011011 \cdot 2^2 = 5,7109375_{10}$$

$$X - Y = X + \bar{Y} = 1,01010011100 \cdot 2^8 - 0,100100100 \cdot 2^2 = ?$$

1) Zapis mantys liczb X i Y: Bity ukryte, nie zapisywane w mantysie

Bity mantysy:

X	Bity liczby												Uzupełnienie zerami											
1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Y	Bity liczby												Uzupełnienie zerami											
1	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2) Wyrównywanie wykładników

Konieczność przesunięcia cyfr w mantysie drugiej liczby o 6 pozycji

$$X - Y = 1,01010011100 \cdot 2^8 - 0,000001011011011 \cdot 2^8 = ?$$

$$X + \bar{Y} = 1,01010011100 \cdot 2^8 + 1,111110100100100 \cdot 2^8 = ?$$

Zapis liczb w mantysie

Bity mantysy:

X	Bity liczby												Uzupełnienie zerami											
1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Y	zera						Bity liczby – przesunięte cyfry												Uzupełnienie zerami					
0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0

Tworzenie uzupełnienia U9 liczby Y

X	Bity liczby												Uzupełnienie zerami											
1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

\bar{Y}	zera						Bity liczby – przesunięte cyfry												Uzupełnienie zerami					
1	1	1	1	1	1	1	0	1	0	0	1	0	0	1	0	0	1	1	1	1	1	1	1	1

+																								
1	1	1	1	1	1	1	0	1	0	0	1	0	0	1	0	0	1	1	1	1	1	1	1	1
	1	0	1	0	0	1	1	0	1	1	1	0	0	1	0	0	1	1	1	1	1	1	1	1
	1	0	1	0	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0

Obecność przeniesienia pokazuje, że wynik jest dodatni.

Tutaj dodajemy do najmniej znaczącej cyfry (bitu) wyniku, ponieważ jest to uzupełnienie U9, czyli uzupełnienie (p-1)-sze.

Otrzymany wynik:

$$X - Y = X + \bar{Y} = 1,01010011100 \cdot 2^8 - 0,100100100 \cdot 2^2 = 1,010011011100101 \cdot 2^8$$

$$X - Y = 339,5 - 5,7109375 = 333,7890625_{10}$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Inaczej ten przykład można przedstawić w następujący sposób. **Zamiast odejmować X uwzględnimy, że to dodawanie liczby ujemnej**, więc przekonwertujemy liczbę ujemną na uzupełnienie U9:

$$X = 1,01010011100 \cdot 2^8$$

$$Y = (-1,011011011) \cdot 2^2$$

Mantysa liczby $X = 1,01010011100$

Wykładnik liczby $X = 10000111$

$$X_{IEEE} = 0 \ 10000111 \ (1) \ 0101 \ 0011 \ 1000 \ 0000 \ 0000 \ 000$$

Mantysa liczby $Y = 1,011011011$

Wykładnik liczby $Y = 10000001$

$$Y_{IEEE} = 1 \ 01111111 \ (1) \ 0110 \ 1101 \ 1000 \ 0000 \ 0000 \ 000$$

$$Y_{IEEE} = 1 \ 10000111 \ (0) \ 0000 \ 0101 \ 1011 \ 0110 \ 0000 \ 000 - \text{wyrównanie wykładników}$$

↓ Zmiana znaku z minusa (1) dla liczby ujemnej, na plus (0) dla uzupełnienia

$$\bar{Y}_{IEEE} = 0 \ 10000111 \ (1) \ 1111 \ 1010 \ 0100 \ 1001 \ 1111 \ 111 - \text{uzupełnienie U9 liczby (mantysy) } Y (1 \neq 0)$$

$$X_{IEEE} = 0 \ 10000111 \ (1) \ 0101 \ 0011 \ 1000 \ 0000 \ 0000 \ 000$$

1 – przeniesienie poza mantysę (oznacza, że wynik jest dodatni)

$$X + \bar{Y} = 0 \ 10000111 \ (1) \ 0100 \ 1101 \ 1100 \ 1001 \ 1111 \ 111 - \text{suma}$$

+ 1 – przeniesienie poza mantysę

$$X + \bar{Y} = 0 \ 10000111 \ (1) \ 0100 \ 1101 \ 1100 \ 1010 \ 0000 \ 000 - \text{suma}$$

$$X - Y = X + \bar{Y} = 101001101,1100101$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”**Mnożenie i dzielenie liczb zmiennopozycyjnych**

Wykonywanie działań mnożenia i dzielenia liczb zmiennopozycyjnych nie wymaga wykonywania działań wyrównywania wykładników. Działania wykonuje się na mantysach i na wykładnikach. Po wykonaniu działań należy dokonać normalizacji mantysy, czyli sprowadzenia jej do wymaganej postaci.

Dodawanie i odejmowanie wykładników realizowane jest w kodzie spolaryzowanym, dlatego wymagane jest wprowadzenie wymaganych korekt wyniku. Działania takie można wykonać w kodzie NKB, a następnie odjąć współczynnik polaryzacji,

Algorytm wykonywania mnożenia

- Pomnożyć mantysy,
- Zaokrąglić otrzymany wynik i dokonać normalizacji mantysy wynikowej,
- Dodać do siebie wykładniki
- Dokonać korekty otrzymanego, sumarycznego wykładnika o składniki korekcyjne wynikające z normalizacji mantysy.

Algorytm wykonywania dzielenia

- Podzielić mantysy,
- Zaokrąglić otrzymany wynik i dokonać normalizacji mantysy wynikowej (ilorazu),
- Odjąć wykładniki
- Dokonać korekty otrzymanego wykładnika o składniki korekcyjne wynikające z normalizacji mantysy.

Mając dane dwie liczby zmiennopozycyjne

$$x = S_1 M_1 B^{E_1}$$

$$y = S_2 M_2 B^{E_2}$$

gdzie:

S_1, S_2 – znak, odpowiednio liczby x, y ; $S_1, S_2 \in \{-1, 1\}$

M_1, M_2 – mantysa, odpowiednio liczby x, y

B – podstawa pozycyjnego systemu liczbowego

E_1, E_2 – wykładnik, odpowiednio liczby x, y

Mnożenie dwóch liczb zmiennopozycyjnych

$$x \cdot y = (S_1 \cdot S_2) \cdot (M_1 \cdot M_2) \cdot B^{(E_1 + E_2)}$$

$$x / y = (S_1 \cdot S_2) \cdot (M_1 / M_2) \cdot B^{(E_1 - E_2)}$$

Mnożenie liczb rzeczywistych

Przykład

$$X = 9,125_{10} = 1001,001_2$$

$$Y = 60,25_{10} = 111100,01_2$$

$$X \cdot Y = ?$$

Krok 1 – zapisujemy poszczególne liczby w formacie najmniejsza możliwa liczba całkowita pomnożona przez podstawę systemu liczbowego podniesionego do określonej potęgi:

$$X = 9,125_{10} = 1001,001_2 = 1001001_2 \cdot 2^{-3}$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

$$Y = 60,25_{10} = 111100,01_2 = 11110001_2 \cdot 2^{-2}$$

$$X \cdot Y = 1001001_2 \cdot 2^{-3} \cdot 11110001_2 \cdot 2^{-2} = 1001001_2 \cdot 11110001_2 \cdot 2^{-2} \cdot 2^{-3} =$$

$$= 1001001_2 \cdot 11110001_2 \cdot 2^{-5}$$

Krok 2 – mnożymy te liczby całkowite:

$$\begin{array}{r} Y = 11110001_2 \\ \cdot X = 1001001_2 \\ \hline 11110001_2 \\ 11110001xxx_2 \\ + 11110001xxxxxx_2 \\ \hline 100010010111001_2 \end{array}$$

I dodajemy wykładniki

$$2^{-3} + 2^{-2} = 2^{-5}$$

Zatem wynik mnożenia jest następujący:

$$X \cdot Y = 100010010111001_2 \cdot 2^{-5} = 1000100101,11001_2 = 549,78125_{10}$$

Czyli analogicznie jak w przypadku mnożenia liczb dziesiętnych, liczba miejsc po przecinku wyniku jest sumą liczb po przecinku poszczególnych czynników mnożenia.

Normalizujemy mantysę

$$X \cdot Y = 1,00010010111001_2 \cdot 2^9$$

Następnie kodujemy wykładnik:

$$W = 127 + 9 = 136 = 10001000_2$$

I zapisujemy tę liczbę w notacji pojedynczej precyzji IEEE 754

$$(X \cdot Y)_{IEEE} = 0 \ 10001000 \ 0001001011100100000000$$



Bit znaku | wykładnik | mantysa bez ukrytej 1 | zera dodatkowe

Właściwości arytmetyki zmiennoprzecinkowej

Kolejność wykonywania działań wpływa na wynik końcowy.

Arytmetyka taka nie jest przemienna ani łączna:

$$(x + y) + z \neq x + (y + z)$$

$$(x \cdot y) \cdot z \neq x \cdot (y \cdot z)$$

Również nie jest rozdzielna:

$$(x + y) \cdot z \neq x \cdot z + y \cdot z$$

Podczas wykonywania działań na liczbach zmiennopozycyjnych występują również:

- Nieprawidłowe działania,
- Zaokrąglenia,
- Przepelnienia,
- Nedomiar.

Algebra Boole’a

Wprowadzenie

Istnieje gałąź matematyki nazywana algebrą Boole’a, która jest wykorzystywana podczas projektowania i analizowania komputerowych układów cyfrowych. Nazwa pochodzi od matematyka George’a Boole’a, który to w 1854 roku zaproponował podstawowe zasady tej algebry.

Algebra Boole’a zakłada możliwość występowania tylko dwóch stanów (Prawdy i Fałszu) jednak przy założeniu, że w jednym miejscu w tym samym czasie nie występują one na raz.

W algebrze tej zakłada się, że wyrażenie może mieć nieskończoną liczbę zmiennych, każda z nich jednak oznaczana jest indywidualnie, aby reprezentować zmienne wejściowe do wyrażenia. Przykładowo mogą to być litery: A , B , C itd. Jest to opracowanie zasad wykonywania działań logicznych.

Wykonywanie działań jedynie na wartościach przyjmujących dwa stany:

- Prawda (T) albo fałsz (F),
- 1 albo 0, inaczej można powiedzieć, że jeżeli $x \neq 0$ wtedy $x = 1$; natomiast jeżeli $x \neq 1$ wtedy $x = 0$;
- Włączony albo wyłączony,

Mimo tego, że w algebrze Boole’a posługujemy się zmiennymi mogącymi mieć dwa stany, czasem nazywane jako 0 i 1, nie należy ich mylić ani utożsamiać z miarami ilości/liczby np. elementów, systemem binarnym oraz działaniami wykonywanymi tam.

Wyrażenia algebry Boole’a powstają wskutek połączenia takich dwuwartościowych zmiennych oraz realizacji działań na nich.

Algebra Boole’a jest prostym sposobem przedstawiania akcji przełączania bramek logicznych w zależności od zmiany wartości argumentów (danych wejściowych). Wykorzystywane są trzy **podstawowe** bramki: **negacja NOT**, **suma logiczna OR** i **iloczyn logiczny AND**. Istnieją jednak i są również stosowane bramki pochodne.

Przykładowo w algebrze Boole’a $A + A = A$, a nie $2A$, tak jak w klasycznej algebrze liczb rzeczywistych. Algebra ta bazuje na zmiennych logicznych. Prawda + Prawda nie dają dwóch Prawd, tylko Prawdę. Ale „suma” logiczna Prawda i Fałsz dają Prawdę.

Tablica prawdy

Tablica prawdy (ang. *Truth table*) jest to matematyczna tabela zero-jedynkowa, stosowana w logice. Ma ona związek z Algebrą Boole’a, rachunkiem zdań, funkcjami logicznymi. Przedstawia potrzebne/analizowane kombinacje wartości argumentów $\{0,1\}$ i wartość zwracaną przez dane wyrażenie lub funkcję $\{0,1\}$. W szczególności tablica prawdy służy do pokazania, dla jakich wartości wejściowych poszczególnych zmiennych dane wyrażenie jest prawdziwe.

Dla każdej zmiennej wejściowej (dla każdego argumentu) tablica ma jedną **kolumnę**. Najczęściej tablica przedstawia wszystkie możliwe kombinacje wejściowe. Inne przypadki to takie, że przedstawione są wszystkie możliwe kombinacje lub wszystkie interesujące dla rozważenia danego zagadnienia.

Tablica taka ma również jedną **kolumnę końcową** pokazującą wszystkie możliwe **wyniki** działań/wyrażenia/funkcji logicznej reprezentowanej przez tablicę.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Każdy **wiersz** tablicy prawdy zawiera jedną możliwą konfigurację zmiennych wejściowych (na przykład $a = \text{prawda}$, $b = \text{fałsz}$), oraz końcowy wynik działania/wyrażenia przy założeniu danej kombinacji danych wejściowych.

Podsumowując można powiedzieć, że tablica prawdy, tablica prawdy jest to graficzne przedstawienie w tablicy kombinacji stanów logicznych zmiennych wejściowych, wyjściowych, a czasami również obliczeń cząstkowych dla danego wyrażenia/zdania/funkcji logicznej.

Tablica prawdy dla alternatywy – funkcji OR

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Skrócona tablica prawdy

Wiedząc, że pewne funkcje logiczne wymagają dwóch zmiennych oraz przyjmując pewną konwencję zapisu tych zmiennych tak, jakby kolejnym wierszom, przypisywać kolejne wartości binarne czasami wykorzystuje się skrócony sposób prezentacji tablicy prawdy.

Funkcja	Działanie	Skrócona tablica prawdy
OR	$x + y$	(0111)
AND	$x \cdot y$	(0001)
XOR	$x \oplus y$	(0110)
NAND	$x \uparrow y$	(1110)
NOR	$x \downarrow y$	(1000)
XNOR	$x \odot y$	(1001)

Diagram Venna

Diagram Venna jest to graficzny sposób przedstawienia wszystkich możliwych relacji logicznych między skończonymi zbiorami różnych elementów.

Diagramy te przedstawiają elementy jako punkty na płaszczyźnie i tworzą obszary wewnątrz zamkniętych krzywych.

Diagram Venna składa się z wielu nachodzących na siebie zamkniętych krzywych, zwykle okręgów, z których każdy reprezentuje zbiór. Punkty wewnątrz krzywej oznaczonej A reprezentują elementy zbioru A , natomiast punkty poza granicą tej krzywej reprezentują elementy spoza zbioru A .

Diagramy Venna powstały około 1880 r. Zostały wymyślone przez Johna Venna. Służą do nauczania elementarnej teorii zbiorów, a także ilustrują proste zależności między sobą w prawdopodobieństwie, logice, statystyce, językoznawstwie i informatyce.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Diagramy Venna nie mogą służyć do udowadniania twierdzeń. Obrazują jednak wiele zasad i zależności.

Biały obszar pokazuje, gdzie stwierdzenie jest fałszywe

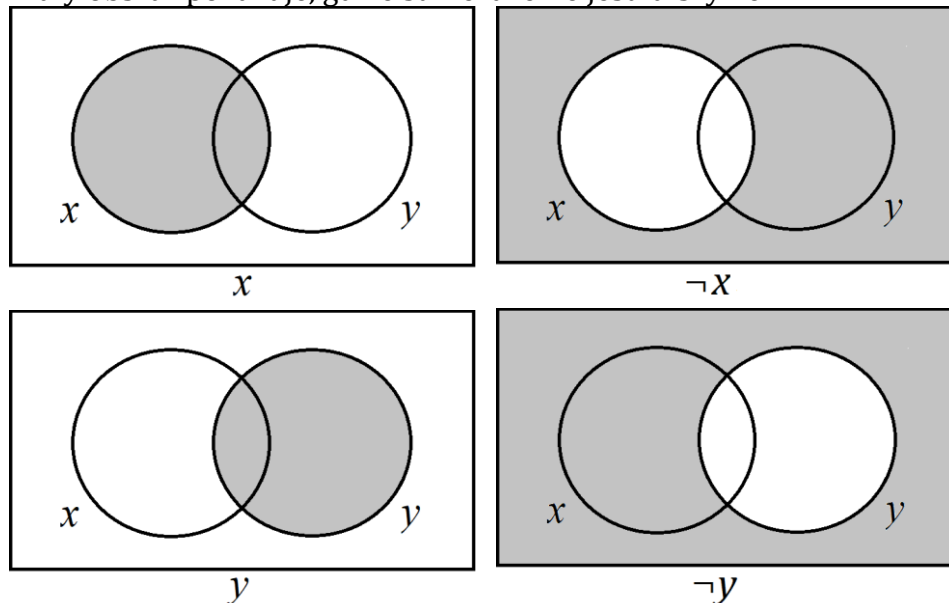
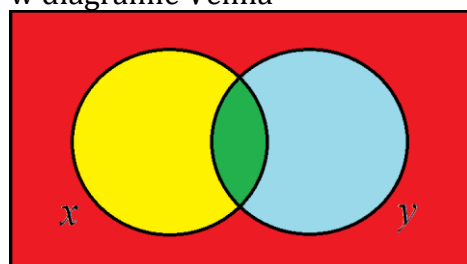


Diagram Venna a tablica prawdy

Kolory pokazują, które wiersze tablicy prawdy odpowiadają odpowiednim obszarom w diagramie Venna



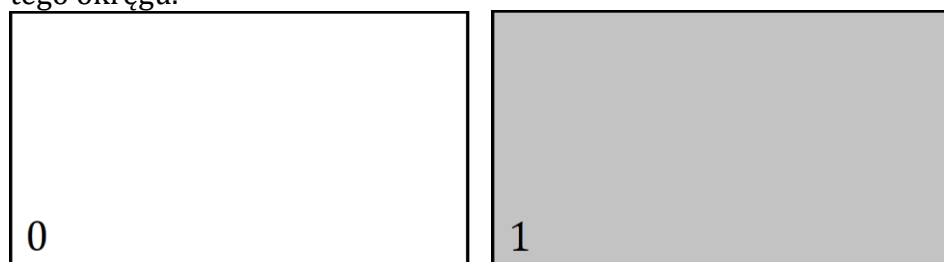
x	y
0	0
0	1
1	0
1	1

Diagram Venna a stałe logiczne

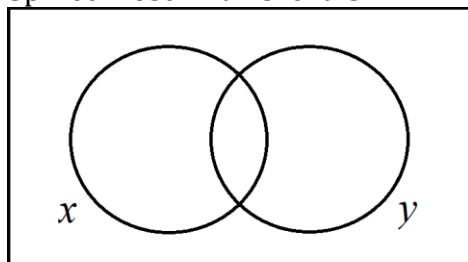
Diagram Venna dla stałej logicznej 0, czyli dla fałszu, będzie trywialny – będzie to prostokąt biały w środku.

Dla prawdy, czyli dla logicznej 1, również diagram jest trywialny, będąc ciemnym pudełkiem.

W żadnym z tych przypadków diagram Venna nie będzie zawierał żadnego okręgu w środku. Chyba, że byłby to okrąg dla zmiennej x (o takiej samej wartości logicznej jak reszta diagramu), która będzie takiego samego koloru, co tło na zewnątrz tego okręgu.

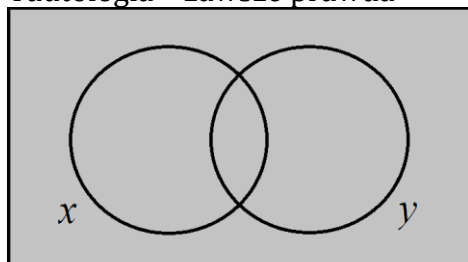


Sprzeczność – zawsze fałsz



x	y	\mathbf{I}
0	0	0
0	1	0
1	0	0
1	1	0

Tautologia – zawsze prawda



x	y	\mathbf{T}
0	0	1
0	1	1
1	0	1
1	1	1

Pusta prawda

Pusta prawda (ang. *Vacuous truth*) – w matematyce i logice jest to stwierdzenie, które twierdzi, że wszyscy członkowie zbioru pustego mają pewną właściwość.

Inaczej rzecz ujmując: pusty zestaw obiektów może mieć jakąkolwiek właściwość.

Przykładowo zdanie "wszystkie telefony komórkowe w pokoju są wyłączone" będzie prawdziwe, jeżeli w pokoju nie będzie żadnych telefonów komórkowych.

Przykładem może być wnioskowanie – implikacja ze zdania, które jest fałszywe.

W potocznym rozumieniu jest to bezsensowne sformułowanie i wnioskowanie, ponieważ fakt, że poprzednik jest fałszywy, uniemożliwia użycie danego stwierdzenia do wnioskowania o prawdziwości następnika.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Podstawowe działania

Negacja – Funkcja logiczna NOT

Negacja, inaczej zaprzeczenie, jest działanie wymagające tylko jednego argumentu.

Zwraca wartość przeciwną do argumentu. Jeżeli na wejściu jest 0 to zwraca 1, a jeżeli na wejściu jest 1 to zwraca 0. Ponieważ odwracają one stan wejściowy, czasami są nazywane inwerterami.

Tablica prawdy dla negacji

A	\bar{A}
0	1
1	0

Negacja każdemu zdaniu/stanowi x przyporządkowuje zdanie NIE x . Inne interpretacje to: *nieprawda, że x* lub *nie jest tak, że x* .

Ogólny sposób działania negacji

$\neg x = 1 - x$ – **to tylko obrazowe** przedstawienie działania negacji (w algebrze Boole’a nie ma odejmowania)

zatem

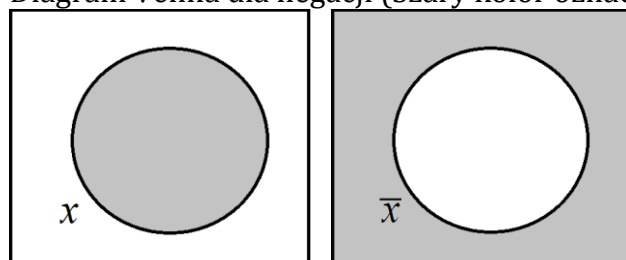
$$\neg 0 = 1$$

$$\neg 1 = 0$$

Negację można również nazwać **dopełnieniem logicznym**.

Diagram przedstawia dopełnienie $\neg x$ zaznaczone przez zacienienie obszaru nie znajdującego się wewnątrz okręgu.

Diagram Venna dla negacji (Szary kolor oznacza prawdę)



Suma logiczna OR

Alternatywa, inaczej **suma logiczna**, jest działanie dwuargumentowe. Zwraca prawdę (1), jeżeli choć jeden argument jest prawdziwy. Zwraca nieprawdę-fałsz jedynie w przypadku, kiedy obydwa argumenty są fałszywe (0).

Stosuje się tu czasem spójnik LUB (ang. OR), który pokazuje, że prawda jest zwracana, jeżeli jeden argument LUB drugi są prawdziwe.

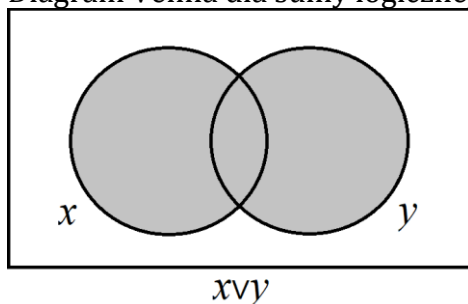
Przez pewne podobieństwo nazwano tę funkcję sumą logiczną.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Tablica prawdy dla alternatywy – zmienne x i y można nazwać składnikami alternatywy.

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Diagram Venna dla sumy logicznej



Suma logiczna, alternatywa – wynik (zestaw argumentów) jest prawdziwy wtedy i tylko wtedy, gdy **jeden lub więcej** z jego argumentów jest prawdziwy.

$x \vee y$ jest prawdziwe, jeżeli x jest prawdą **lub** jeżeli y jest prawdą, **lub** jeżeli jedno i drugie x i y są prawdziwe.

$$x + y = x \vee y = \max(x, y)$$

Zgodnie z teorią „pustej prawdy” (ang. *Vacuous truth*) przyjmuje się, że suma logiczna z zera elementów jest równa 0.

Im więcej elementów wchodzi w skład sumy logicznej, czyli im większa jest alternatywa, tym większe prawdopodobieństwo, że w wyniku otrzyma się wartość jeden 1. Z drugiej strony im mniej elementów, tym większe prawdopodobieństwo uzyskania w wyniku wartości zero 0.

Lub jest zwykle wyrażany przez operatora infiks:

- w matematyce i logice \vee ;
- w elektronice $+$;
- oraz w większości języków programowania, $|$, $||$ lub **OR** .

Określenia pokrewne:

- W języku naturalnym spójnik LUB,
- W teorii mnogości – unia, suma zbiorów,

W algebrze Boole’a alternatywa, czyli funkcja OR działa zgodnie z prawem przemienności. Nie jest istotna kolejność zmieniania stanów wejściowych. Istotny jest aktualny stan tych wejść. Jeżeli choć jedno ma status Prawda (1, wł) to operator zwraca wartość Prawda (1). Jedynie w przypadku, kiedy obydwa wejścia mają status Fałsz, zwracana jest wartość: Fałsz (0).

Suma logiczna jest równa 1, jeżeli choć jedna zmienna wejściowa jest równa 1:

$$0 + 0 + 0 + \dots + 0 + 0 + 0 = 0; \quad 0 + 1 + 0 + \dots + 0 + 0 + 0 = 1;$$

Ponieważ czasami wykorzystywana jest tzw. Alternatywa wykluczająca (ang. *Exclusive-OR*), to tę alternatywę dla odróżnienia nazywa się alternatywą zwykłą, łączną lub nierozłączną (ang. *Inclusive OR*).

$$(x \vee y) \Leftrightarrow (\neg x \Rightarrow y)$$

Iloczyn logiczny AND

Koniunkcja, inaczej **iloczyn logiczny**, jest działanie dwuargumentowe. Zwraca prawdę (1) tylko w przypadku, kiedy obydwa argumenty są prawdziwe. Zatem zwraca prawdę, jeżeli dwa lub więcej argumentów jest prawdziwych jednocześnie, czyli w tym samym czasie, aby zwracana była wartość: Prawda (1). W każdym innym przypadku zwracany jest: Fałsz (0).

Kolejność, w której argumenty osiągnęły stan prawdy jest nieistotna, ponieważ nie wpływa na wynik końcowy. Istotne jest tylko to, czy wszystkie argumenty mają status: Prawda, czy może choć jeden ma status: Fałsz.

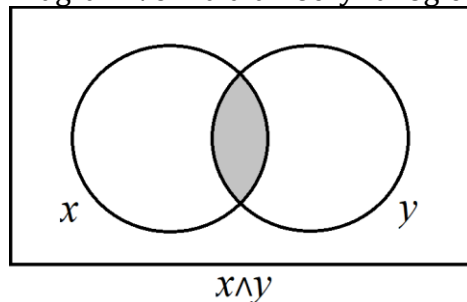
Jeżeli choć jeden argument jest fałszywy (0), wtedy zwraca nieprawdę 0. Stosuje się tu czasem spójnik I (ang. AND), który pokazuje, że prawda jest zwracana, jeżeli jeden argument I drugi są prawdziwe.

Przez podobieństwo nazwaną ją iloczynem logicznym.

tablica prawdy dla koniunkcji:

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

Diagram Venna dla iloczynu logicznego



W algebrze Boole’a operator AND działa zgodnie z Prawem przemienności, czyli nie ma znaczenia kolejność, w której rozpatruje się elementy wejściowe.

Zasadniczo jest dwuargumentowy, ale łatwo można rozpisać tablice prawdy dla większej liczby wejść.

Iloczyn logiczny jest równy 1, jeżeli wszystkie wejścia są równe 1:

$$1 \cdot 1 \cdot 1 \cdot 1 \cdot \dots \cdot 1 \cdot 1 \cdot 1 = 1; \quad 1 \cdot 0 \cdot 1 \cdot 1 \cdot \dots \cdot 1 \cdot 1 \cdot 1 = 0;$$

Można zatem powiedzieć, że mnożenie logiczne niczym nie różni się od algebraicznego.

Zgodnie z teorią „pustej prawdy” (ang. *Vacuous truth*) przyjmuje się, że iloczyn logiczny z zera elementów jest równy 1.

Im więcej elementów wchodzi w skład iloczynu logicznego, czyli im większa jest koniunkcja, tym większe prawdopodobieństwo, że w wyniku otrzyma się wartość zero

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

0. Z drugiej strony im mniej elementów, tym większe prawdopodobieństwo uzyskania w wyniku wartości jeden 1.

Koniunkcja, iloczyn logiczny – wynik (zestaw argumentów) jest prawdziwy wtedy i tylko wtedy, gdy **wszystkie** jego składniki są prawdziwe.

$$x \cdot y = x \wedge y = \min(x, y)$$

Określenia pokrewne:

W języku naturalnym spójnik I,

W teorii mnogości – przecięcie, przekrój zbiorów.

Różnica symetryczna XOR

Alternatywa rozłączna, wykluczająca, różnica symetryczna, suma modulo dwa (ang. *exclusive or*), XOR, ALBO – w logice bardzo często wykorzystuje się takie działanie logiczne. Czasami wykorzystuje się tu znak \oplus , $\underline{\vee}$, $\dot{\vee}$.

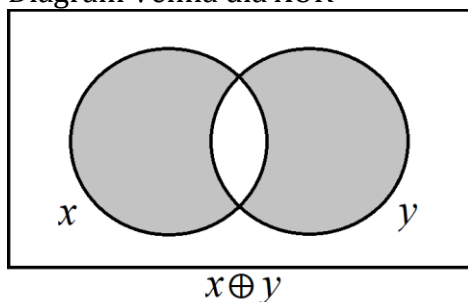
W języku potocznym rozumiana jest jako „nie jednocześnie.”

Zwraca ona wartość prawdy, jeżeli jedynie jeden argument jest prawdziwy. Innymi słowy zwraca ona prawdę jedynie w przypadku, gdy argumenty (zmienne wejściowe) mają różne wartości/stany. Alternatywa rozłączna to działanie dwuargumentowe.

tablica prawdy dla różnicy symetrycznej

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Diagram Venna dla XOR



Alternatywa wykluczająca w zdaniu $(1+1 = 3) \oplus (2+2=2)$ (czyli jedno ALBO drugie) jest fałszywa, ponieważ obydwa zdania są fałszywe.

Alternatywa wykluczająca w zdaniu $(1+1 = 2) \oplus (2+2=4)$ (czyli jedno ALBO drugie) jest fałszywa, ponieważ obydwa zdania są prawdziwe, a do prawdziwości alternatywy rozłącznej konieczne jest, żeby tylko jedno zdanie było prawdziwe.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Tabela. Właściwości alternatywy wykluczającej

$A \oplus B = B \oplus A$	Alternatywa wykluczająca jest przemienne
$A \oplus (B \oplus C) = (A \oplus B) \oplus C$	Alternatywa wykluczająca jest łączna
$A \oplus 0 = A$	Istnieje element neutralny
$A \oplus A = 0$	
$A \oplus 1 = \neg A$	
$A \oplus B = (\neg A \cdot B) + (A \cdot \neg B)$	
$\neg(A \oplus B) = (\neg A \cdot \neg B) + (A \cdot B)$	

Równoważność \Leftrightarrow

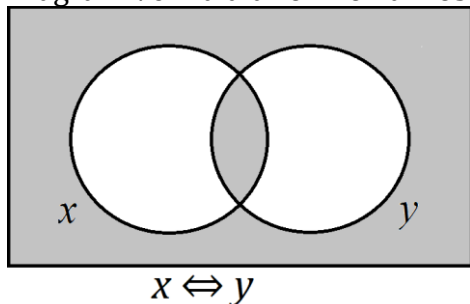
Zapisywana jest za pomocą sformułowania: ...*wtedy i tylko wtedy* (wtw lub wtt), lub „*obydwa albo żadne*”

Równoważność jest prawdziwa, jeżeli dwa zdania (x i y) równocześnie są prawdziwe albo równocześnie są fałszywe.

Tablica prawdy dla równoważności

x	y	$x \Leftrightarrow y$
0	0	1
0	1	0
1	0	0
1	1	1

Diagram Venna dla równoważności



Implikacja materialna \Rightarrow

Implikacja, implikacja materialna, implikacja prosta, związanie jest to zdanie logiczne lub funkcja zdaniowa powstałe przez połączenie dwóch zdań x (poprzednik implikacji) i y (następnik implikacji) spójnikiem implikacji $x \Rightarrow y$.

Implikacja powstaje w wyniku połączenia dwóch elementów: poprzednika (pierwszego zdania) z następnikiem (drugim zdaniem).

Znak implikacji skierowany jest od poprzednika do następnika. Oprócz symbolu \Rightarrow czasami używany jest znak \rightarrow .

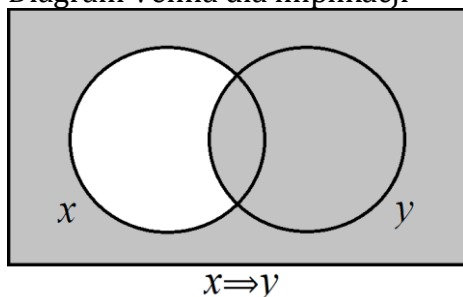
Jeżeli.... to... zatem.... lub Skoro.... to...

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

tablica prawdy dla implikacji

x	y	$x \Rightarrow y$
0	0	1
0	1	1
1	0	0
1	1	1

Diagram Venna dla implikacji



Implikacja zwraca fałsz jedynie w przypadku, kiedy poprzednik jest prawdziwy, a następnik – fałszywy.

Implikacja nie jest przemieniana, czyli poprzednika i następnika nie można zamienić miejscami ze sobą.

Potocznie czasami mówi się, że implikacja przypomina obietnicę i tu wprowadza się trochę zamętu. Przykład: mama powiedziała do córki: *jeżeli odrobisz lekcję to będziesz mogła pójść na imprezę*.

• **Przypadki:**

- 1) córka odrobiła lekcje (poprzednik) i poszła na imprezę (następnik) – mama mówiła prawdę
- 2) córka nie odrobiła lekcji i nie poszła na imprezę – mama mówiła prawdę
- 3) córka odrobiła lekcje i nie została wypuszczona na imprezę – mama ją okłamała,
- 4) córka nie odrobiła lekcji i poszła na imprezę – mama mówiła prawdę, bo mama jedynie obiecała, że jak córka odrobi lekcje to będzie miała zagwarantowane wyjście.

Implikacja - z fałszu wynika prawda

W czystej matematyce, przypadkowa prawidłowość stwierdzenia nie są zazwyczaj interesująca sama w sobie, ale często wykorzystywana jest jako podstawowy przypadek dowodzenia przez indukcję matematyczną. To pojęcie to ma znaczenie w czystej matematyce, a także w każdej innej dziedzinie, która wykorzystuje klasyczną logikę.

Poza matematyką stwierdzenia, które można scharakteryzować nieoficjalnie jako bezsensowne, mogą wprowadzać w błąd. Takie stwierdzenia zawierają rozsądne twierdzenia o obiektach, które w rzeczywistości nie istnieją. Na przykład dziecko może powiedzieć rodzicom: *"Zjadłem wszystkie warzywa na talerzu"*, kiedy na talerzu dziecka nie było warzyw.

Czasami w logice języka inaczej podchodzi się do takich stwierdzeń.

- Zdanie: *Zjadłem wszystkie warzywa na talerzu*

Jest złożeniem dwóch zdań:

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

- *na talerzu były warzywa ORAZ*
- *zjadłem wszystkie warzywa na talerzu*

W tym jednak przypadku wnioskowanie jest inne:

- *Zdanie: Zjadłem wszystkie warzywa na talerzu*

Jest złożeniem dwóch zdań

- *JEŻELI na talerzu były warzywa TO*
- *zjadłem wszystkie warzywa na talerzu*

Własności implikacji

Implikacja jest fałszywa wtedy i tylko wtedy, kiedy poprzednik x jest prawdziwy, a następnik y jest fałszywy. W każdym innym przypadku implikacja jest prawdziwa.

Z fałszu może wynikać cokolwiek: prawda albo fałsz.

Z prawdy musi wynikać prawda.

Zasada kontrapozycji, która jest podstawą dowodu nie wprost:

- $(x \Rightarrow y) \Leftrightarrow (\neg x \Rightarrow \neg y)$

Możliwość zastąpienia implikacji

- $(x \Rightarrow y) \Leftrightarrow (\neg x \vee y)$

Paradoksy implikacji

Paradoksy implikacji (ang. *Paradoxes of material implication*) są logicznymi stwierdzeniami, które są prawdziwe, ale których prawda jest intuicyjnie zaskakująca dla osób, które ich nie znają.

Są to prawdy logiki klasycznego, które jednak są problematyczne intuicyjnie.

Oto kilka przykładów paradoksów implikacji: p, q, r – zmienne logiczne

1. $(\neg p \wedge p) \Rightarrow q$ - p i jego negacja (czyli zawsze fałsz) implikują q ,
2. $p \Rightarrow (q \Rightarrow p)$ - jeżeli p jest prawdziwe to dalej jest implikowane przez każde q .
3. $\neg p \Rightarrow (p \Rightarrow q)$ - jeżeli p jest fałszywe to dalej q jest implikowane przez fałszywe p .
4. $p \Rightarrow (q \vee \neg q)$ - dla każdego q lub jego negacji otrzymuje się wartość prawdy, więc to może być implikowane przez dowolne p .

Paradoksy wynikają z definicji funkcjonalnej odnośnie do prawdziwości implikacji materialnej, która mówi się, że implikacja jest prawdziwa wtedy, gdy poprzednik jest fałszywy, a następnik/konsekwencja jest prawdziwa.

Jak najlepiej znany z paradoksów:

Pada deszcz

I

Nie pada

Zatem z tego wynika, że

George Washington składa się z grabi.

Z tego płynie wniosek, że niespójne przesłanki zawsze czynią argumentację ważną/prawdziwą; to znaczy niespójne przesłanki sugerują jakiegokolwiek wnioski. Wydaje się to paradoksalne, ponieważ chociaż powyższe jest logicznie uzasadnionym argumentem, nie jest to dedukcja logiczna (nie wszystkie jego przesłanki są prawdziwe).

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

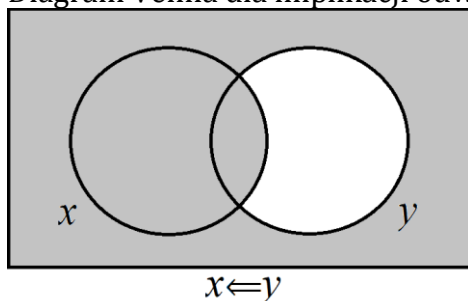
Odwrotna implikacja \Leftarrow

Odwrotna implikacja (ang. *Converse implication*) – używane symbole: \Leftarrow , \Leftarrowarrow
Pierwsze zdanie jest następnikiem, a drugie poprzednikiem.
W języku naturalnym: *x jeżeli y* lub *Nie x bez y*.

tablica prawdy dla implikacji odwrotnej

x	y	$x \Leftarrow y$
0	0	1
0	1	0
1	0	1
1	1	1

Diagram Venna dla implikacji odwrotnej



Zaprzeczona implikacja

Zaprzeczona implikacja materialna (ang. *Material nonimplication*)

Dla dowolnych dwóch zdań x i y zaprzeczona implikacja materialna jest prawdziwa, wtedy i tylko wtedy, gdy negacja implikacji materialnej z x do y jest prawdziwa.

Wykorzystywany jest symbol to najczęściej prostu przekreślony symbol implikacji materialnej: \nrightarrow lub \nRightarrow

W potocznym rozumieniu: ... *ale nie* ...

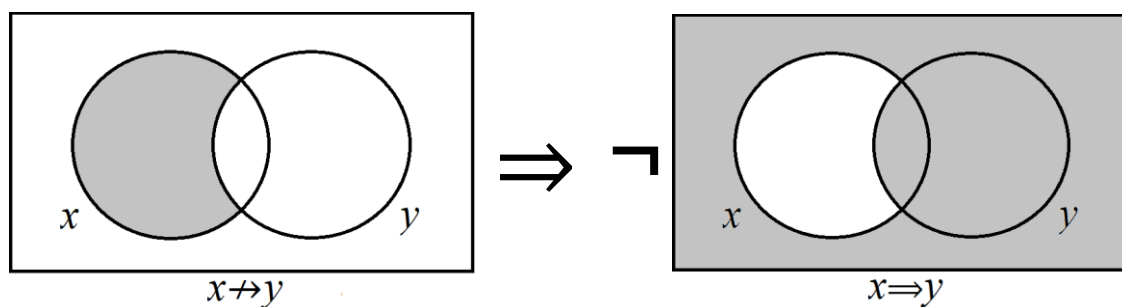
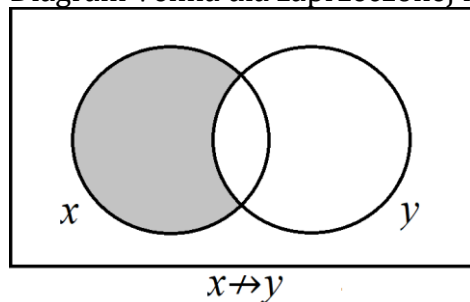
$$x \nrightarrow y = x \cdot \neg y$$

tablica prawdy dla zaprzeczonej implikacji

x	y	$x \Rightarrow y$	$x \nrightarrow y$
0	0	1	0
0	1	1	0
1	0	0	1
1	1	1	0

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Diagram Venna dla zaprzeczonej implikacji



Zaprzeczona implikacja odwrotna

Zaprzeczona implikacja odwrotna (ang. *Converse nonimplication*)

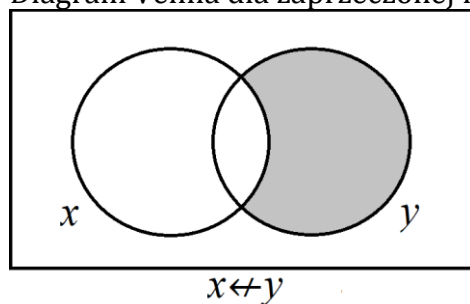
Dla dowolnych dwóch zdań x i y zaprzeczona implikacja odwrotna jest prawdziwa, wtedy i tylko wtedy, gdy negacja implikacji odwrotnej z x do y jest prawdziwa.

Wykorzystywany jest symbol to najczęściej prostu przekreślony symbol implikacji odwrotnej: \nLeftarrow lub \nRightarrow

tablica prawdy dla zaprzeczonej implikacji odwrotnej

x	y	$x \nLeftarrow y$	$x \nRightarrow y$
0	0	1	0
0	1	0	1
1	0	1	0
1	1	1	0

Diagram Venna dla zaprzeczonej implikacji odwrotnej



„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Podstawowe zestawy używanych symboli

	Technika cyfrowa	Teoria mnogości	Rachunek zdań	Język C itp.	Angielskie oznaczenie	Polskie oznaczenie	Inne oznaczenia
Negacja zaprzeczenie	– albo \bar{a}	\sim	\neg	!	NOT	NIE	a'
Suma logiczna, alternatywa	+	\cup	\vee		OR	LUB	
Iloczyn logiczny, koniunkcja	·	\cap	\wedge	&&	AND	I	

Symbole wykorzystywane do przedstawienia określonego rodzaju funkcji

	Angielskie oznaczenie	Polskie oznaczenie				
Alternatywa wykluczająca	XOR	ALBO	\oplus	$\underline{\vee}$	$\dot{\vee}$	\perp
Binegacja	NOR	NIE LUB	\downarrow	$\bar{\vee}$		
Dysjunkcja	NAND	NIE AND	\uparrow		/	
	XNOR		\odot			

Podstawowe prawa algebry Boole’a

jeżeli $x \neq 0$ wtedy $x = 1$; jeżeli $x \neq 1$ wtedy $x = 0$;
Jeżeli $a = 0$, to $\bar{a} = 1$ – element odwrotny Jeżeli $a = 1$, to $\bar{a} = 0$ – element odwrotny

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

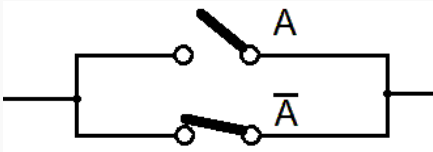
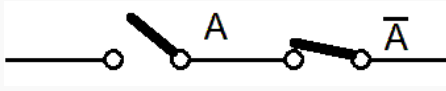
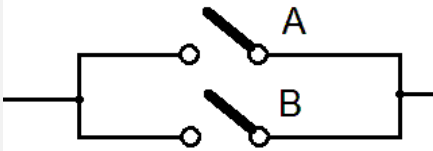
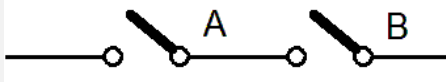
$\overline{(\bar{a})} = a$	Inwolucja, podwójna negacja jest zawsze równa zmiennej
$a \Rightarrow a$	Prawo tożsamości. Każde zdanie implikuje samo siebie
$a + \bar{a} = 1$	Prawo wyłącznego środka – z dwóch zdań (zdania i jego zaprzeczenia) zawsze jedno jest prawdziwe – zdanie jest prawdziwe albo jego zaprzeczenie jest prawdziwe – odpowiednik reguły łacińskiej <i>tertium non datur</i> (łac. trzeciej możliwości nie ma)
$\overline{(\bar{a} + a)} = 0$ $a \cdot \bar{a} = 0$	Prawo niesprzeczności – nie może być jednocześnie prawdziwe zdanie i jego zaprzeczenie

Podstawowe tożsamości algebry Boole’a

$a + 1 = 1$ A równoległe z zamkniętym Zawsze = 1	$a \cdot 0 = 0$ A szeregowo z otwartym Zawsze = 0	Unieważnienie (ang. <i>Annulment</i>)
$a + 0 = a$ A równoległe z otwartym	$a \cdot 1 = a$ A szeregowo z zamkniętym	Prawa tożsamości (ang. <i>Identity</i>) Element identycznościowy
$a + a = a$ Prawo idempotentności koniunkcji A równoległe z A (z samym sobą) równa się sobie samej (zmiennej wejściowej)	$a \cdot a = a$ Prawo idempotentności alternatywy A szeregowo z A (z samym sobą) równa się sobie samej (zmiennej wejściowej)	Prawa tautologii – idempotentność (ang. <i>Idempotent</i>)

Idempotentność jest to właściwość pewnych działań, która pozwala na wielokrotne stosowanie ich bez zmiany wyniku końcowego. Np. moduł z modułu, mnożenie razy 1, dodawanie zera itp.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

$a + \bar{a} = 1$  <p>A równoległe z NIE A Zawsze jeden zamknięty = 1</p>	$a \cdot \bar{a} = 0$  <p>A szeregowo z NIE A Zawsze jeden otwarty = 0</p>	<p>Prawa odwrotności – pochłanianie (ang. <i>Complement</i>)</p>
$a + b = b + a$  <p>Przemienność alternatywy A równoległe z B Kolejność w dodawaniu nie ma znaczenia.</p>	$a \cdot b = b \cdot a$  <p>Przemienność koniunkcji A szeregowo z B Kolejność w mnożeniu nie ma znaczenia.</p>	<p>Prawa przemienności (ang. <i>commutativity</i>) Pozwalają na zamianę pozycji (kolejności zmiennych) podczas wykonywania działań dodawania i mnożenia</p>

$a + (b + c) = (a + b) + c$ Prawo łączności alternatywy	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$ Prawo łączności koniunkcji	Prawo łączenia (ang. <i>associativity</i>)
Umożliwia usunięcie nawiasów i przegrupowania zmiennych podczas wykonywania działań mnożenia i dodawania		
$a + (a \cdot b) = a$	$a \cdot (a + b) = a$	Absorbacja (ang. <i>absorption</i>)
Umożliwia upraszczanie skomplikowanych wyrażeń i pochłanianie niektórych wyrażeń.		
$a + (b \cdot c) = (a + b) \cdot (a + c)$ koniunkcja jest rozdzielna względem alternatywy	$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ alternatywa jest rozdzielna względem koniunkcji	Prawa rozdzielności (ang. <i>distributivity</i>)



W algebrze liczb rzeczywistych mnożenie NIE jest rozdzielne względem dodawania!

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Czy rzeczywiście: $a + (b \cdot c) = (a + b) \cdot (a + c)$?

a	b	c	$(b \cdot c)$	$a + (b \cdot c)$	$(a + b)$	$(a + c)$	$(a + b) \cdot (a + c)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

$a + (a \cdot b) = a \cdot (1 + b) = a \cdot 1 = a$	$a \cdot (a + b) = a + a \cdot b = a$	
$a + (\bar{a} \cdot b) = (a + \bar{a}) \cdot (a + b) = a + b$	$a \cdot (\bar{a} + b) = a \cdot b$	
$\overline{(a + b)} = \bar{a} \cdot \bar{b}$ Rozbij czynniki i zmień OR na AND	$\overline{(a \cdot b)} = \bar{a} + \bar{b}$ Rozbij czynniki i zmień AND na OR	Twierdzenie De Morgana
$\overline{(a + b + c)} = \bar{a} \cdot \bar{b} \cdot \bar{c}$	$\overline{(a \cdot b \cdot c)} = \bar{a} + \bar{b} + \bar{c}$	jw.

Zebrańe tożsamości algebry Boole’a

Tożsamości logiczne

Unieważnienie

Annulment

$$A \wedge 0 = 0$$

$$A \vee 1 = 1$$

Dopełnienie

Complement

$$A \wedge \neg A = 0$$

$$A \vee \neg A = 1$$

Prawo łączenia

Associative

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

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

Tożsamość

Identity

$$A \wedge 1 = 1$$

$$A \vee 0 = A$$

Podwójna negacja

Double negation

$$\neg(\neg A) = A$$

Przemienność

Commutative

$$A \wedge B = B \wedge A$$

$$A \vee B = B \vee A$$

Idempotentność

Idempotent

$$A \wedge A = A$$

$$A \vee A = A$$

Rozdzielność

Distributive

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

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

Absorbacja

Absorptive

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

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

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Symbole boolowskie

Unieważnienie
Annulment

$$A \cdot 0 = 0$$

$$A + 1 = 1$$

Dopełnienie
Complement

$$A \cdot \neg A = 0$$

$$A + \neg A = 1$$

Prawo łączenia
Associative

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

$$(A + B) + C = A + (B + C)$$

Tożsamość
Identity

$$A \cdot 1 = 1$$

$$A + 0 = A$$

Podwójna negacja
Double negation

$$\neg(\neg A) = A$$

Przemienność
Commutative

$$A \cdot B = B \cdot A$$

$$A + B = B + A$$

Idempotentność
Idempotent

$$A \cdot A = A$$

$$A + A = A$$

Rozdzielność
Distributive

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

Absorbacja

Absorptive

$$A + (A \cdot B) = A$$

$$A \cdot (A + B) = A$$

Kilka przykładów

$a + a \cdot b = a \cdot (1 + b) = a \cdot 1 = a$ – wyciągamy a przed nawias

$$(a + b) \cdot (a + c) = a \cdot a + a \cdot b + a \cdot c + b \cdot c =$$

$$= a + a \cdot b + a \cdot c + b \cdot c = a \cdot (1 + b) + a \cdot c + b \cdot c =$$

$$= a + a \cdot c + b \cdot c = a \cdot (1 + c) + b \cdot c = a + b \cdot c$$

Możemy wyeliminować żółty element.

Prawo algebry Boole’a:

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

więc

$$a + (\bar{a} \cdot b) = (a + \bar{a}) \cdot (a + b) = 1 \cdot (a + b) = (a + b)$$

$$\bar{a} + (a \cdot b) = (\bar{a} + a) \cdot (\bar{a} + b) = 1 \cdot (\bar{a} + b) = (\bar{a} + b)$$

Możemy wyeliminować żółty element.

Specyfika algebry Boole’a

Każde z praw logiki w algebrze Boole’a ma jedną lub dwie zmienne, ale prawo nie ogranicza się tylko do takiej liczby zmiennych. Może istnieć nieskończenie wiele zmiennych danych wejściowych do danego wyrażenia np. podczas dodawania lub mnożenia.

Prawa Boole’a mogą zostać wykorzystane do udowodnienia dowolnego np. bardziej skomplikowanego wyrażenia lub do uproszczenia bardziej skomplikowanych układów cyfrowych. Uproszczenie takie może polegać na identyfikacji niepotrzebnych bramek logicznych w danym projektowanym układzie cyfrowym, realizującym określone funkcje logiczne. Dzięki temu można zredukować liczbę wymaganych bramek, a skutkiem tego uzyskuje się uproszczenie układu, oszczędności finansowe i związane ze zużyciem energii.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Kolejność wykonywania działań

W algebrze Boole’a przyjęto następującą kolejność wykonywania działań:

1. negacja (negowanie) – NOT
2. iloczyn logiczny (koniunkcja) – AND
3. suma logiczna (alternatywa) – OR
4. Implikacja \Rightarrow
5. równoważność \Leftrightarrow

Nawiasy w algebrze Boole’a odgrywają rolę analogiczną do roli w arytmetyce – wskazują kolejność wykonywania działań, która może różnić się od kolejności wynikającej z kolejności zmiennych.

Bramki cyfrowe

Definicja

Brama logika jest:

- wyidealizowanym teoretycznym modelem – idealna bramka logiczna to taka, który ma zerowy czas narastania, działanie bezzwłoczne, możliwość nieskończonej liczby odpowiedzi na występujące zmiany stanów wejściowych w bardzo krótkim czasie, lub
- nieidealnym urządzeniem fizycznym realizującym określoną funkcję logiczną (z listy podstawowych funkcji).

Każda bramka cyfrowa realizuje funkcję logiczną. Na poszczególnych wejściach ma podane określone wartości i zwraca na pojedynczym wyjściu stan wyjściowy.

Bramka to pewna abstrakcja odpowiadająca operatorowi logicznemu.

Bramka logiczna to układ np. cyfrowy realizujący fizycznie wybraną prostą funkcję logiczną, której zmienne logiczne (argumenty), oraz wynikowo sama funkcja mogą przybierać jedną z dwóch wartości logicznych: 0 albo 1.

Bramki logiczne negacji, sumy logicznej – alternatywy i iloczynu logicznego – koniunkcji są podstawowymi elementami logicznymi, powszechnie wykorzystywanymi w budowie cyfrowych układów logicznych.

Bufor cyfrowy

Bufor cyfrowy (lub bufor napięciowy) to element obwodu elektronicznego, który służy do izolowania wejścia od wyjścia, zapewniając na wyjściu brak napięcia lub napięcie równe napięciu wejściowemu (poziom L lub H).

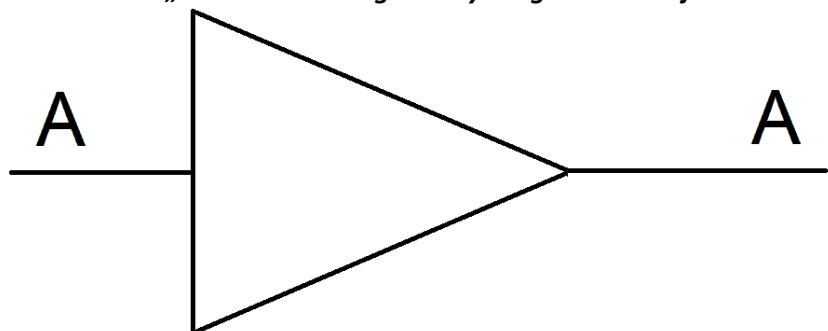
Zużywa stosunkowo niewiele energii i nie zakłóca pierwotnego obwodu. Jest również nazywany buforem wzmocnienia jedności, ponieważ zapewnia wzmocnienie równe 1, co oznacza, że dostarcza co najwyżej takie samo napięcie jak napięcie wejściowe, nie pełniąc funkcji wzmocnienia.

Tabela prawdy bufora cyfrowego

A	A
0	0
1	1

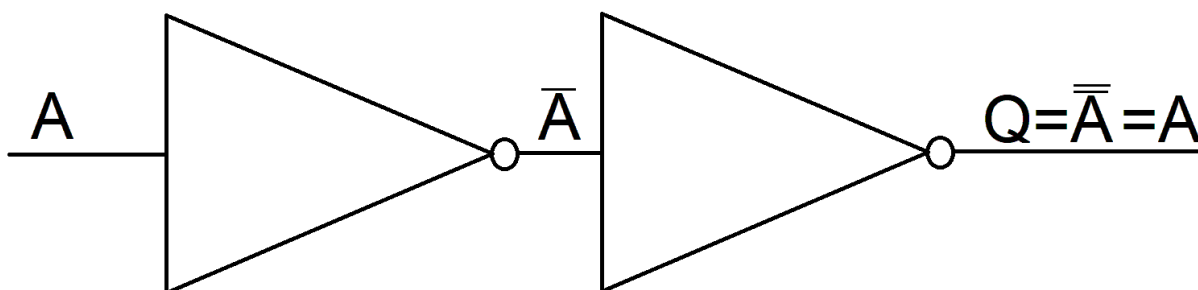
Symbol bufora cyfrowego

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”



Bufor cyfrowy może zostać wykonany w wyniku połączenia ze sobą dwóch bramek NOT, jak pokazano poniżej.

Pierwsza „odwróci” sygnał wejściowy A , a druga „odwróci” go z powrotem do pierwotnego poziomu, wykonując podwójną inwersję sygnału wejściowego.



Negacja – bramka NOT

Negator jest bramką cyfrową realizującą funkcję negacji, inaczej zaprzeczenie.

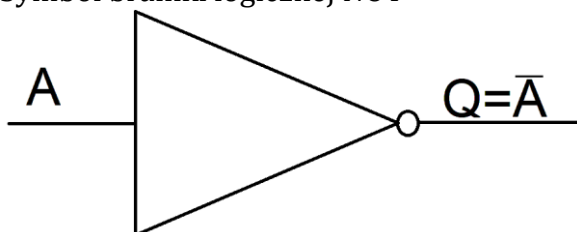
Zwraca wartość odwrotną do argumentu. Jeżeli na wejściu jest 0 to zwraca 1, a jeżeli na wejściu jest 1 to zwraca 0. Ponieważ odwracają one stan wejściowy, czasami są nazywane **inwerterami**.

Czasami mówi się, że negacja jest operatorem bąbelkowym, ponieważ negacja wyjścia z danego elementu zaznaczana jest za pomocą **koła**. Negację tu można też nazwać dopełnieniem logicznym.

tablica prawdy dla negacji

A	\bar{A}
0	1
1	0

Symbol bramki logicznej NOT



„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Negacja każdemu zdaniu/stanowi **A** przyporządkowuje zdanie **NIE A**. Inne interpretacje to: *nieprawda, że A* lub *nie jest tak, że A*.

Stan wyjściowy bramki NOT jest uzupełnieniem, przeciwieństwem lub odwrotnością sygnału wejściowego.

Ogólny sposób działania negacji:

$\bar{A} = \neg A = 1 - A$ – to tylko obrazowe przedstawienie działania negacji

zatem

$$\neg 0 = 1$$

$$\neg 1 = 0$$

Wyrażenie logiczne dla NOT: $Q = \neg A$

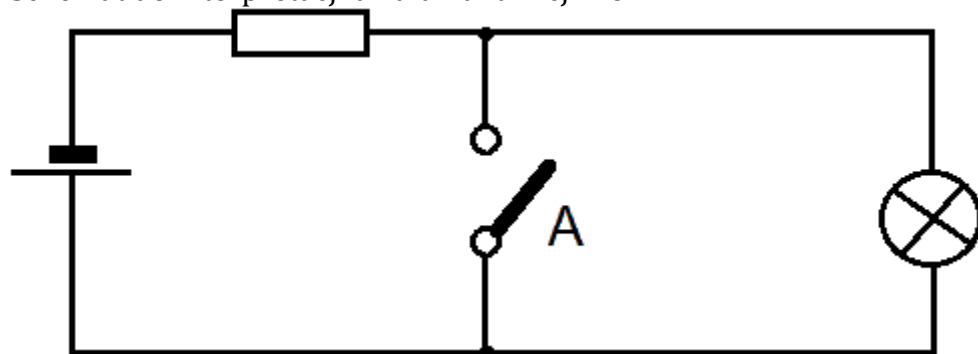
Analityczne równanie funkcji NOT to: $f(a) = 1 - a$

$$f(0) = 1 - 0 = 1$$

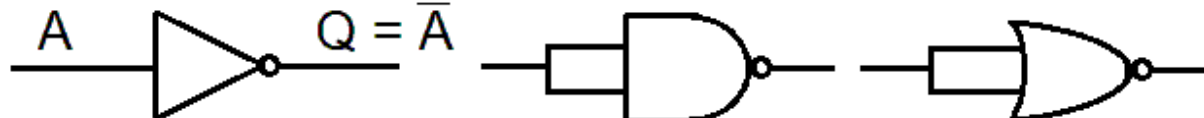
$$f(1) = 1 - 1 = 0$$

- Negacja może być przedstawiona jako wyłącznik zwierający – mostkujący. Jeżeli wyłącznik A jest zamknięty, to prąd nie płynie przez żarówkę. Jeżeli jest rozwarty – to żarówka się świeci.
- Bramki NOT, nazywane inwerterami, są powszechnie łączone z bramkami OR i AND, aby otrzymać odpowiednio NOR i NAND.

Schemat do interpretacji działania funkcji NOT



Funkcje równoważne negacji uzyskane dzięki użyciu innych bramek cyfrowych:



Bramka OR

Alternatywa, inaczej **suma logiczna**, zasadniczo jest operacją dwuargumentową. Zwraca prawdę (1), jeżeli choć jeden argument jest prawdziwy. Zwraca nieprawdę-fałsz jedynie w przypadku, kiedy obydwa/wszystkie argumenty są fałszywe (0).

Stosuje się tu czasem spójnik LUB (ang. OR), który pokazuje, że prawda jest zwracana, jeżeli jeden argument LUB drugi są prawdziwe.

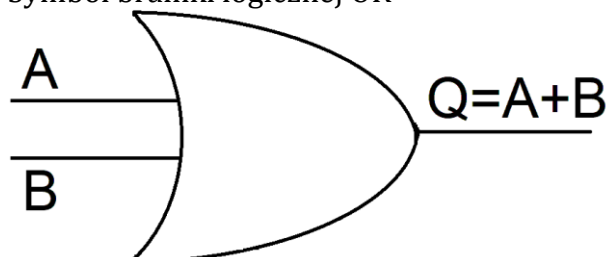
Przez pewne podobieństwo nazwano tę funkcję sumą logiczną.

tablica prawdy dla alternatywy – zmienne A i B można nazwać składnikami alternatywy.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

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

Symbol bramki logicznej OR



Wyrażenie logiczne dla OR: $Q = A + B$

Analityczne równanie funkcji OR to: $f(a,b) = a + b - a \cdot b$

$$f(0,0) = 0 + 0 - 0 \cdot 0 = 0$$

$$f(0,1) = 0 + 1 - 0 \cdot 1 = 1$$

$$f(1,0) = 1 + 0 - 1 \cdot 0 = 1$$

$$f(1,1) = 1 + 1 - 1 \cdot 1 = 2 - 1 = 1$$

Jeżeli przynajmniej jedno A lub B są prawdziwe, Q jest prawdziwe

W algebrze Boole'a alternatywa, czyli funkcja OR działa zgodnie z prawem przemienności. Nie jest istotna kolejność zmieniania stanów wejściowych. Istotny jest aktualny stan tych wejść. Jeżeli choć jedno ma status Prawda (1, wł) to operator zwraca wartość Prawda (1). Jedynie w przypadku, kiedy obydwa wejścia mają status Fałsz, zwracana jest wartość: Fałsz (0).

Ponieważ czasami wykorzystywana jest tzw. Alternatywa wykluczająca (ang. *Exclusive-OR*), to tę alternatywę dla odróżnienia nazywa się alternatywą zwykłą, łączną lub nierozłączną (ang. *Inclusive OR*).

Suma logiczna jest równa 1, jeżeli choć jedna zmienna wejściowa jest równa 1:
 $0 + 0 + 0 + \dots + 0 + 0 + 0 = 0$; $0 + 1 + 0 + \dots + 0 + 0 + 0 = 1$;

bramka OR **podaje/zwraca wartość 1** na wyjściu wtedy i tylko wtedy, kiedy **dowolne wejście ma wartość 1**, lub wtedy, kiedy stan choć jednego wejścia nie jest równy 0.

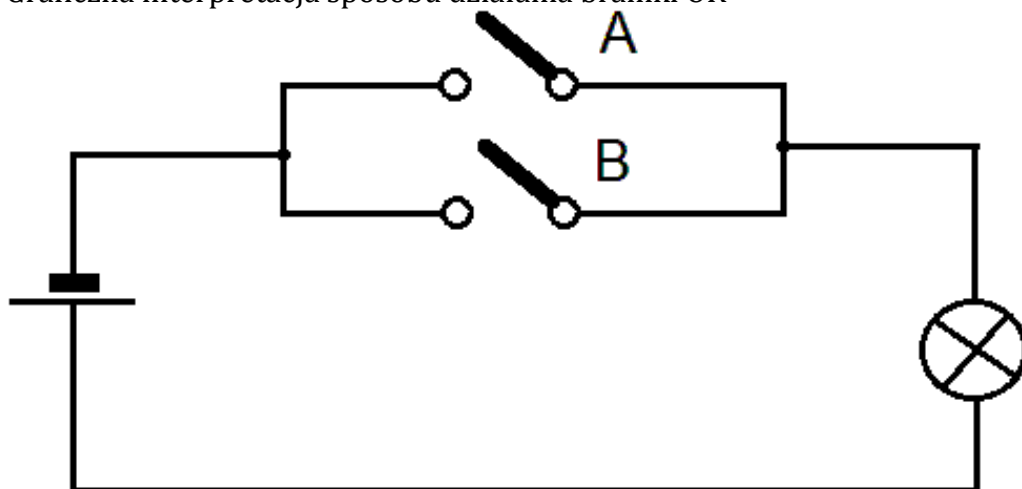
Jeszcze inaczej można powiedzieć, że bramka OR realizuje funkcję, która na wyjściu podaje **największą wartość** występującą **na dowolnym wejściu**.
 $x \text{ OR } y \text{ OR } z = \max(x, y, z)$

Operator OR może być przedstawiony jako dwa wyłączniki połączone równolegle. Wystarczy, że tylko jeden, dowolny wyłącznik zostanie włączony, a żarówka zapali się. W takiej sytuacji stan drugiego wyłącznika nie ma żadnego znaczenia, podobnie jego załączenie lub wyłączenie.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Żarówka zgaśnie jedynie w przypadku, kiedy obydwa wyłączniki będą rozłączone

Graficzna interpretacja sposobu działania bramki OR



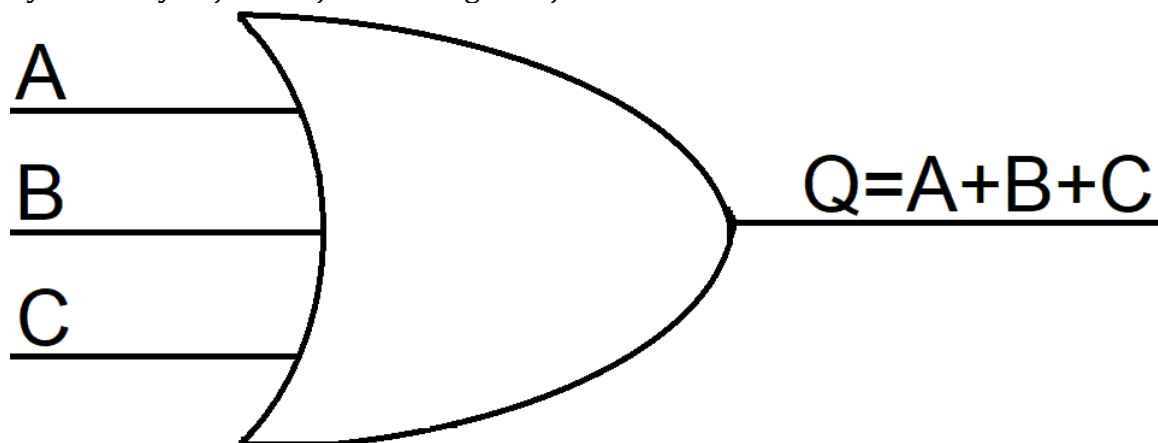
3-wejściowa bramka logiczna OR

Wyrażenie logiczne: $Q = A + B + C$

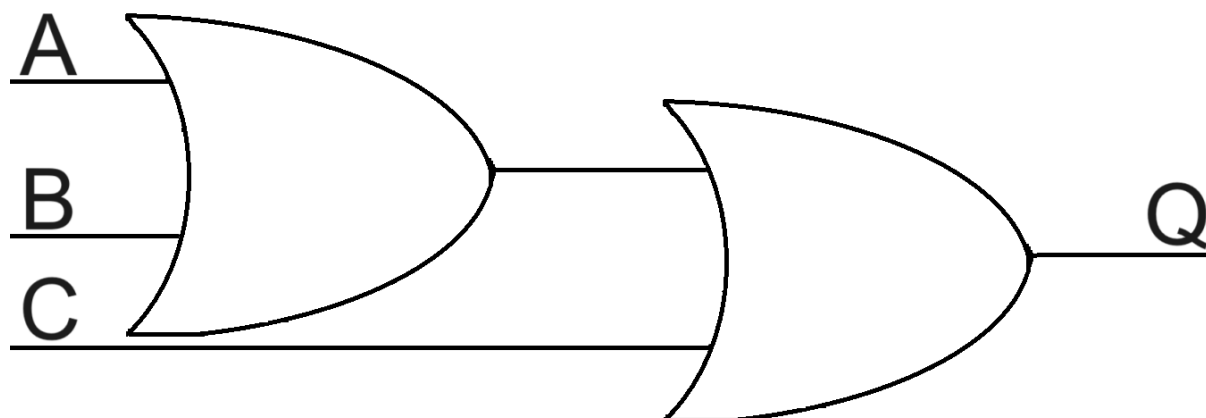
Tablica prawdy dla trzywejściowej bramki logicznej OR

A	B	C	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Symbol trzywejściowej bramki logicznej OR



Trzywejściowa bramka OR zachowuje się w taki sposób, jak szeregowe połączenie dwóch bramek dwuwejściowych OR.



$$(A + B) + C = A + (B + C)$$

Bramka AND

Koniunkcja, inaczej **iloczyn logiczny**, jest operacją dwuargumentową. Zwraca prawdę (1) tylko w przypadku, kiedy obydwa argumenty są prawdziwe. Zatem zwraca prawdę, jeżeli dwa lub więcej argumentów jest prawdziwych jednocześnie, czyli w tym samym czasie, aby zwracana była wartość: Prawda (1). W każdym innym przypadku zwracany jest: Fałsz (0).

Kolejność, w której argumenty osiągnęły stan prawdy jest nieistotna, ponieważ nie wpływa na wynik końcowy. Istotne jest tylko to, czy wszystkie argumenty mają status: Prawda, czy może choć jeden ma status: Fałsz.

Jeżeli choć jeden argument jest fałszywy (0), wtedy zwraca nieprawdę 0.

Stosuje się tu czasem spójnik I (ang. *AND*), który pokazuje, że prawda jest zwracana, jeżeli jeden argument I drugi są prawdziwe. Przez podobieństwo nazywaną ją iloczynem logicznym.

W algebrze Boole’a operator AND działa zgodnie z Prawem przemienności, czyli nie ma znaczenia kolejność, w której rozpatruje się elementy wejściowe.

Zasadniczo jest dwuargumentowy, ale łatwo można rozpisać tablice prawdy dla większej liczby wejść.

Wyrażenie logiczne: $Q = A \cdot B$

Analityczne równanie funkcji AND to: $f(a,b) = a \cdot b$

$$f(0,0) = 0 \cdot 0 = 0$$

$$f(0,1) = 0 \cdot 1 = 0$$

$$f(1,0) = 1 \cdot 0 = 0$$

$$f(1,1) = 1 \cdot 1 = 1$$

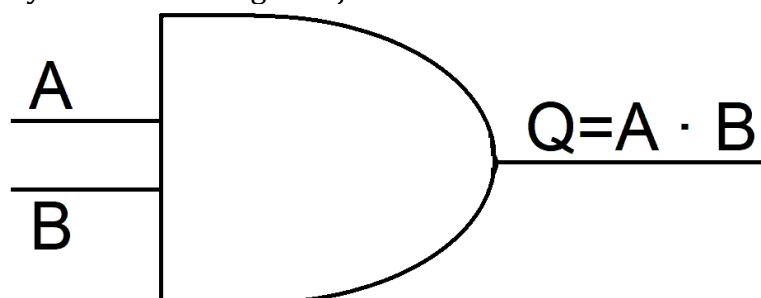
Jeżeli zarówno A, jak i B są prawdziwe, Q jest prawdziwe

tablica prawdy dla koniunkcji:

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0

1	1	1
---	---	---

Symbol bramki logicznej AND



Iloczyn logiczny jest równy 1, jeżeli wszystkie wejścia są równe 1:
 $1 \cdot 1 \cdot 1 \cdot 1 \cdot \dots \cdot 1 \cdot 1 \cdot 1 = 1$; $1 \cdot 0 \cdot 1 \cdot 1 \cdot \dots \cdot 1 \cdot 1 \cdot 1 = 0$;

Można zatem powiedzieć, że mnożenie logiczne niczym nie różni się od algebraicznego.

bramka AND **podaje/zwraca wartość 1** na wyjściu wtedy i tylko wtedy, kiedy **wszystkie wejścia mają wartość 1**.

Jeszcze inaczej można powiedzieć, że bramka and realizuje funkcję, która na wyjściu podaje **najmniejszą wartość** występującą **na dowolnym wejściu**.

$$x \text{ AND } y \text{ AND } z = \min(x, y, z)$$

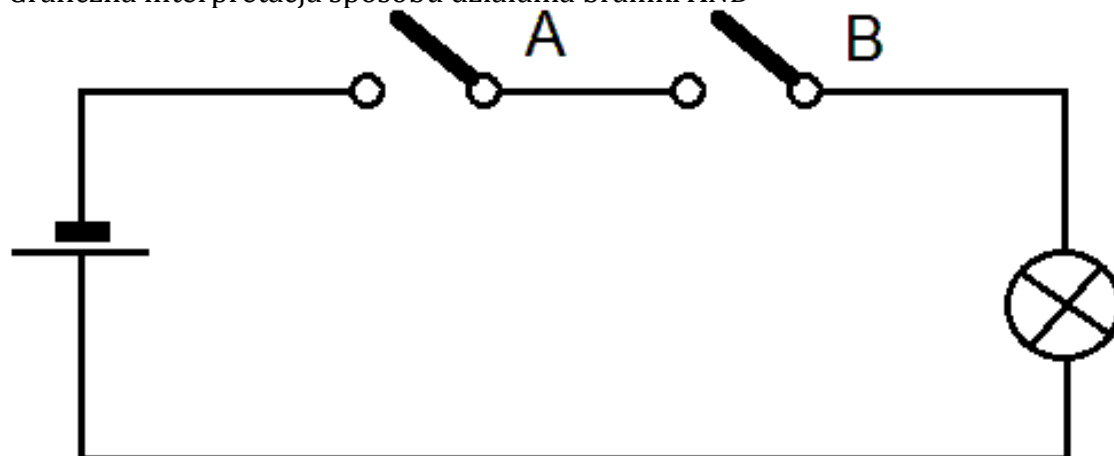
Stan wyjściowy bramki logicznej AND zwraca poziom „NISKI” tylko wtedy, gdy DOWOLNE z jej wejść ma poziom logiczny „NISKI”.

Innymi słowy, dla bramki logicznej AND, zmiana dowolnego stanu wejścia na 0 da na wyjściu 0.

Operator AND można graficznie przedstawić jako dwa wyłączniki połączone szeregowo. Każdy wyłącznik ma tylko dwa możliwe stany: włączony (wł) lub wyłączony (wył). Zatem istnieją możliwe 4 różne kombinacje wyłączników.

Żarówka zaświeci się jedynie w przypadku, kiedy obydwa wyłączniki zostaną załączone (stan: wł/ON). Kolejność ich załączenia oraz ewentualne opóźnienia są nieistotne. Jeżeli choć jeden wyłącznik jest wyłączony (stan: wył/OFF) to żarówka nie zaświeci się. Wtedy też nie ma znaczenia stan drugiego wyłącznika.

Graficzna interpretacja sposobu działania bramki AND



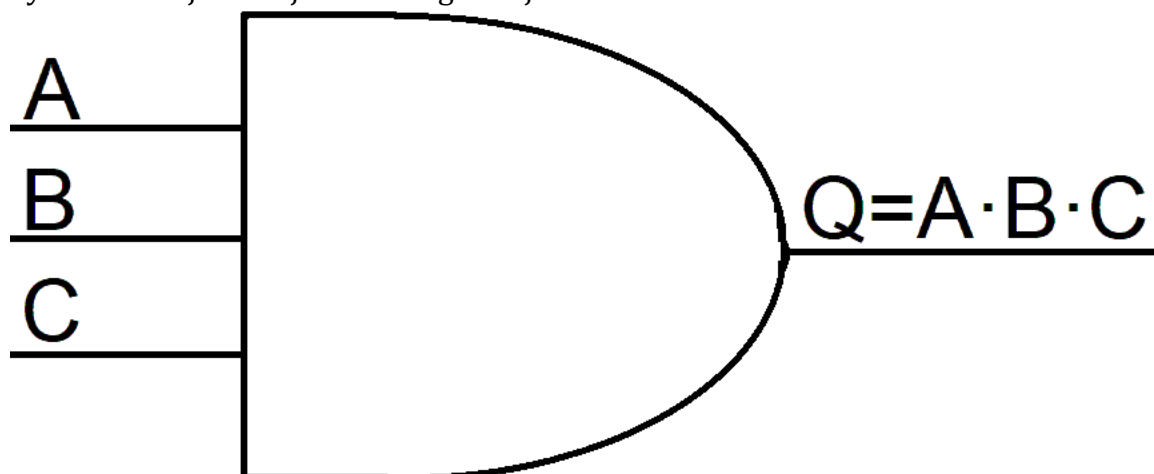
3-wejściowa bramka logiczna AND

Wyrażenie logiczne: $Q = A \cdot B \cdot C$

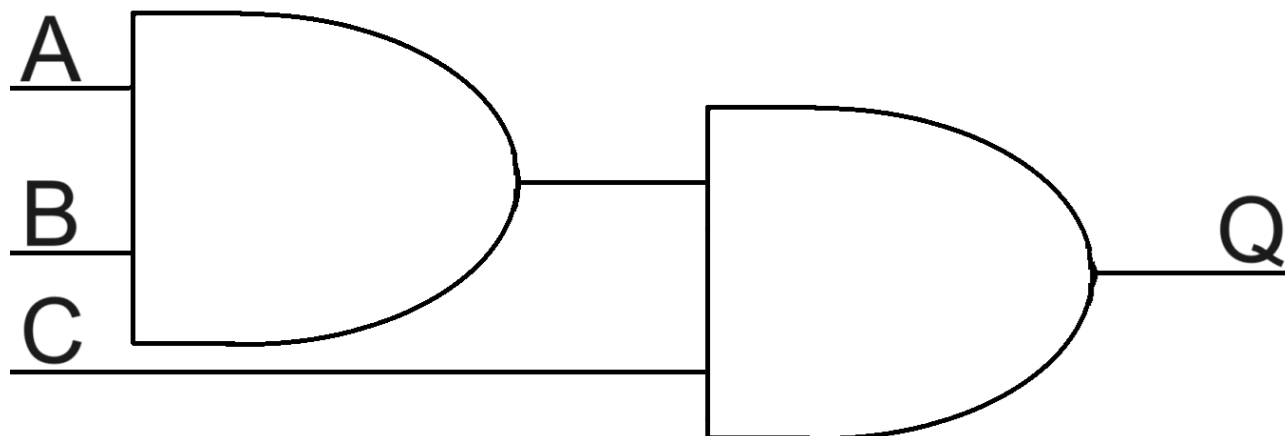
Tablica prawdy 3-wejściowej bramki logicznej AND

A	C	B	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Symbol 3-wejściowej bramki logicznej AND



Trzywejściowa bramka AND zachowuje się w taki sposób, jak szeregowe połączenie dwóch bramek dwuwejściowych AND.



$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$



„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Pusta koniunkcja – bramka zerowejściowa AND

Bramka AND zachowuje się jak mnożnik logiczny zmiennych wejściowych.

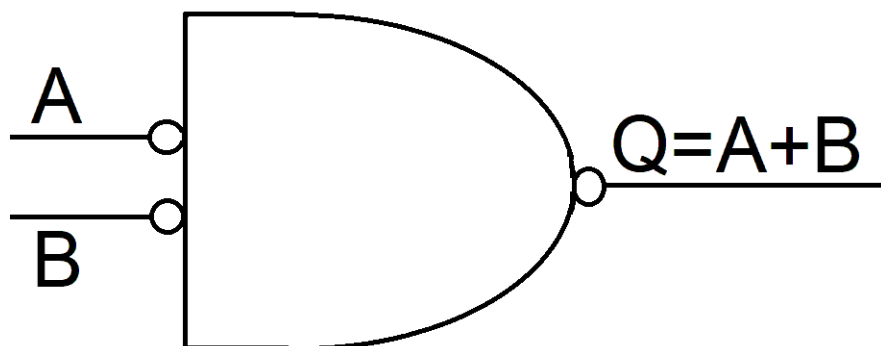
W przypadku 0 wejść do bramki AND mamy pustą koniunkcję, która jest identycznie równa prawdzie (ang. *true*).

Prawa de Morgana

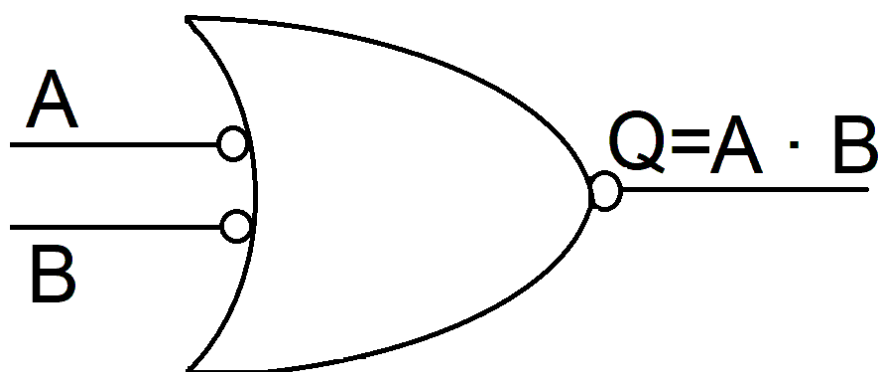
Prawa de Morgana

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

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



A	B	$\neg A$	$\neg B$	$(\neg A \wedge \neg B)$	$\neg(\neg A \wedge \neg B)$	$A \vee B$
0	0	1	1	1	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	0	1	1



A	B	$\neg A$	$\neg B$	$(\neg A \vee \neg B)$	$\neg(\neg A \vee \neg B)$	$A \wedge B$
0	0	1	1	1	0	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	0	1	1

Bramki podstawowe

Bramki NOT, AND i OR tworzą pełny funkcjonalnie zestaw elementów. Można z nich zbudować każdy układ logiczny.

Za pomocą wyłącznie bramek dysjunkcji – NAND albo wyłącznie bramek binegacji – NOR można także zbudować każdy układ logiczny, również realizujący funkcje podstawowe.

Można zatem powiedzieć, że bramki NOR, bramki NAND stanowią oddzielnie funkcjonalnie minimalny zestaw funkcji.

Bramka XOR

Alternatywa rozłączna, wykluczająca, różnica symetryczna, suma modulo dwa (ang. *exclusive or*), XOR, ALBO – w logice bardzo często wykorzystuje się taką operację logiczną. Czasami wykorzystuje się tu znak \oplus , \vee , $\dot{\vee}$.

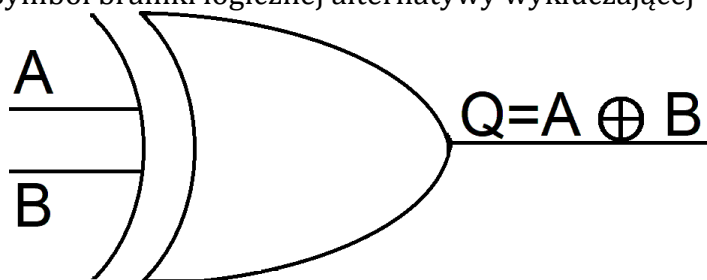
W języku potocznym rozumiana jest jako „nie jednocześnie.”

Zwraca ona wartość prawdy jeżeli jedynie jeden argument jest prawdziwy. Innymi słowy zwraca ona prawdę jedynie w przypadku, gdy argumenty (zmienne wejściowe) mają różne wartości/stany. Alternatywa rozłączna to działanie dwuargumentowe.

tablica prawdy dla alternatywy wykluczającej

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

Symbol bramki logicznej alternatywy wykluczającej – XOR



Wyrażenie logiczne XOR: $Q = A \oplus B = \bar{A} \cdot B + A \cdot \bar{B}$

Analityczne równania funkcji XOR to: $f(a,b) = a + b - 2 \cdot a \cdot b$ lub $f(a,b) = |a - b|$

$$f(0,0) = 0 + 0 - 2 \cdot 0 \cdot 0 = 0$$

$$f(0,1) = 0 + 1 - 2 \cdot 0 \cdot 1 = 1$$

$$f(1,0) = 1 + 0 - 2 \cdot 1 \cdot 0 = 1$$

$$f(1,1) = 1 + 1 - 2 \cdot 1 \cdot 1 = 2 - 2 = 0$$

Wyjście bramki XOR przyjmuje stan „WYSOKI” **TYLKO**, gdy jej dwa zaciski wejściowe są na „**RÓŻNYCH**” poziomach logicznych względem siebie.

Nieparzysta liczba logicznych „jedynek” na jego wejściach daje logikę „1” na wyjściu. Te dwa wejścia mogą być na poziomie logicznym „1” lub na poziomie logicznym „0”.

Funkcja bramki XOR jest uzyskiwana przez połączenie standardowych bramek logicznych w celu utworzenia bardziej złożonych funkcji bramek.

Bramka XOR z dwoma wejściami jest w zasadzie sumatorem modulo dwa, ponieważ daje sumę dwóch liczb binarnych i w rezultacie ma bardziej złożoną konstrukcję niż inne podstawowe typy bramek logicznych.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Trzywejściowa bramka XOR

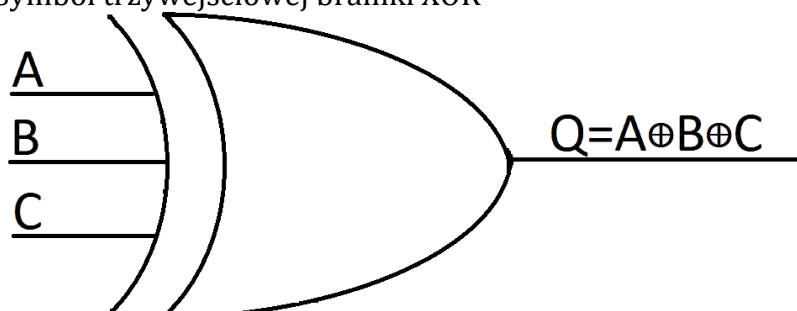
Wyrażenie boolowskie: $Q = A \oplus B \oplus C = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$

Bramka XOR z więcej niż dwoma wejściami jest nazywana funkcją nieparzystości lub sumą modulo-2 (Mod-2-SUM), a nie funkcją XOR. Jeżeli zanegujemy wyjście trzywejściowej bramki XOR nie otrzymamy bramki równoważności!

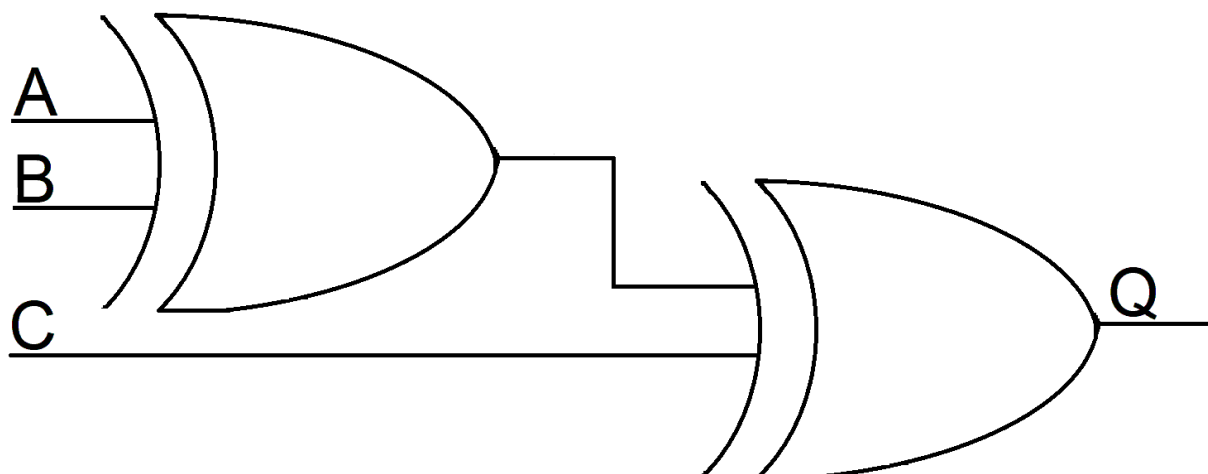
Tablica prawdy trzywejściowej bramki XOR

A	B	C	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Symbol trzywejściowej bramki XOR



Trzywejściowa bramka XOR zachowuje się w taki sposób, jak szeregowe połączenie dwóch bramek dwuwejściowych XOR.



$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Bramka NOR

Binegacja jest kombinacją dwóch innych, oddzielnych funkcji: NOT i OR. To połączenie szeregowo. Funkcja NOT odwraca/neguje to, co znajduje się na wyjściu bramki OR, czyli znajduje się za bramką OR.

W języku potocznym: *ani... ani...*

Bramka zwraca prawdę jedynie w przypadku, kiedy na obydwu wejściach jest stan: Fałsz.

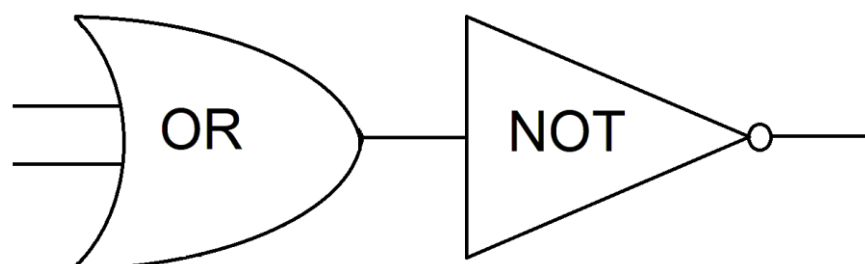
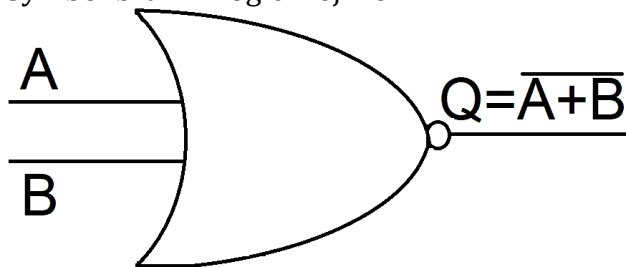
Binegacja, NOR, jest to negacja alternatywy, a także koniunkcją negacji.

Binegacja jest to dwuargumentowa funkcja Boolowska, która realizuje zaprzeczoną sumę logiczną NOT OR. Funkcja odpowiada sformułowaniu „*ani... nie jest, ani... nie jest*”. Czasem binegacja jest zastępowana negacją alternatywy.

tablica prawdy dla binegacji

A	B	$A + B$	$A \downarrow B$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Symbol bramki logicznej NOR



Bramka NOR

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

$$A \downarrow B = \neg A \wedge \neg B$$

Poniższy schemat elektryczny obrazuje działanie funkcji NOR. Włączenie dowolnego wyłącznika A lub B powoduje zwarcie/zmostkowanie żarówki, która przestaje się wtedy świecić. Stan drugiego wyłącznika w takiej sytuacji nie ma znaczenia, jak również jego załączanie i wyłączanie w takiej sytuacji.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Graficzna interpretacja bramki NOR

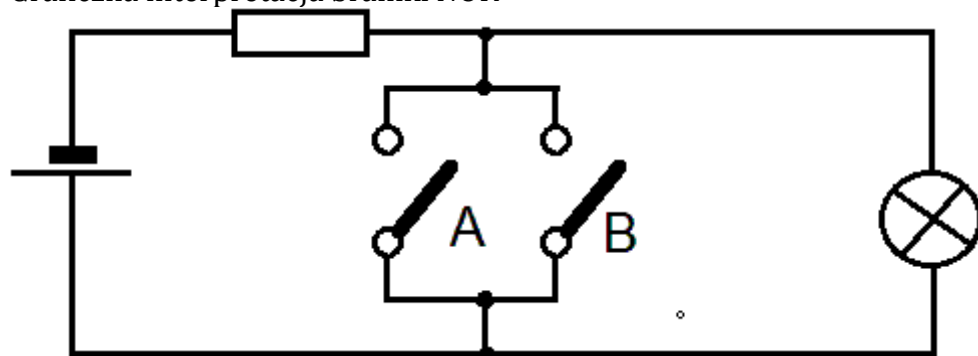
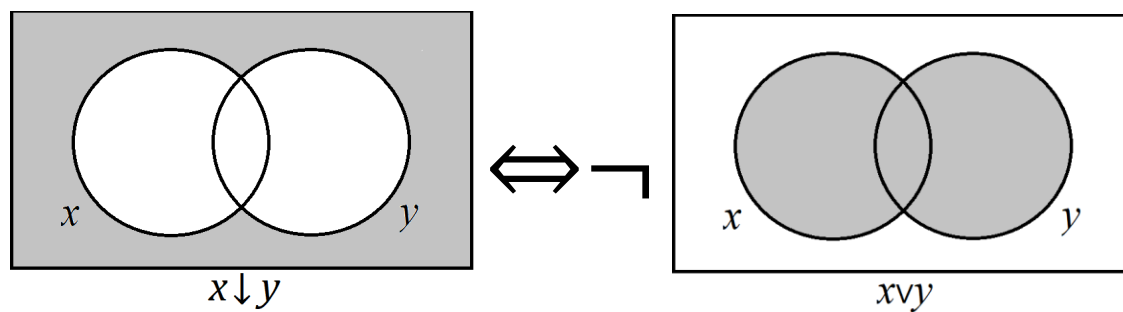
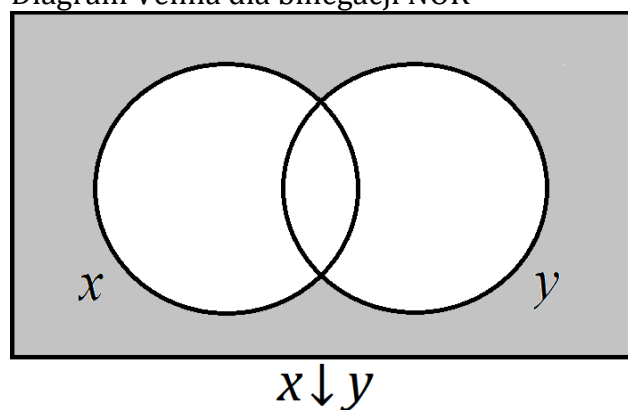


Diagram Venna dla binegacji NOR



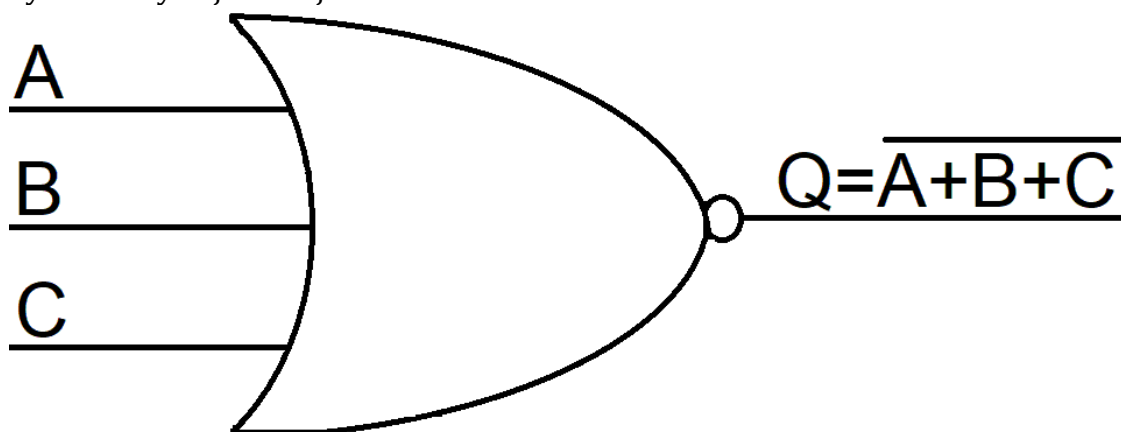
3-wejściowa bramka logiczna NOR

Wyrażenie logiczne: $Q = \overline{A + B + C}$

Tablica prawdy

A	C	B	Q
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Symbol trzywejściowej bramki NOR



Bramka NAND

Dysjunkcja, dysjunkcja/dyzjunkcja Sheffera, funkcja Sheffera, NAND, **niewspółzachodzenie**,

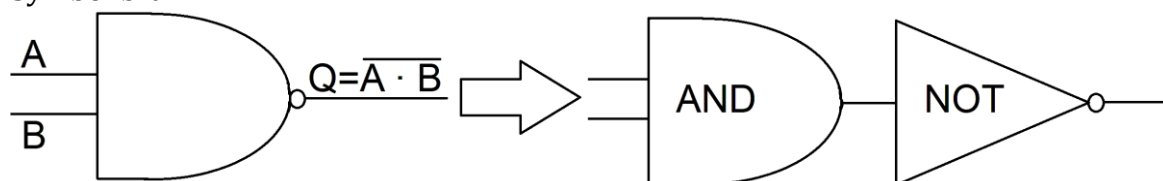
Dysjunkcja jest niejako połączeniem szeregowym dwóch osobnych operatorów AND i NOT. Dysjunkcja jest przedstawiana jako $/$, $|$ lub \uparrow . Bramka NAND neguje wynik bramki AND.

Funkcję tę opisuje zdanie: „Jeżeli zarówno A, jak i B są prawdziwe, Q NIE jest prawdziwe”

tablica prawdy dla dysjunkcji

A	B	$A \cdot B$	$A \uparrow B$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Symbol bramki NAND



Bramka/funkcja NAND zwraca prawdę wyłącznie wtedy, kiedy na przynajmniej jednym wejściu jest Fałsz. Wtedy też nie ma znaczenie stan drugiego wejścia.

Funktor zwraca Fałsz wyłącznie wtedy, kiedy obydwa wejścia mają stan: Prawda.

Poniżej znajduje się graficzna interpretacja funktora NAND. Żarówka będzie się świecić tak długo, jak przynajmniej jeden wyłącznik będzie wyłączony.

Jeżeli obydwa wyłączniki zostaną załączone (Prawda) spowoduje to zmostkowanie żarówki i zgaśnie ona.

Graficzna interpretacja sposobu działania bramki NAND

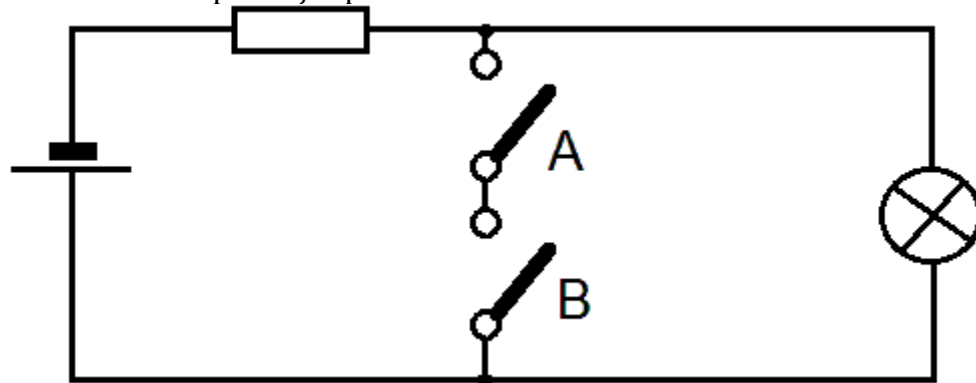
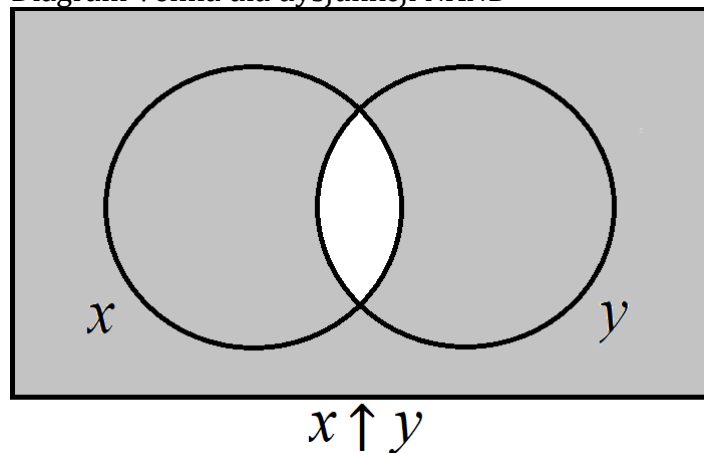
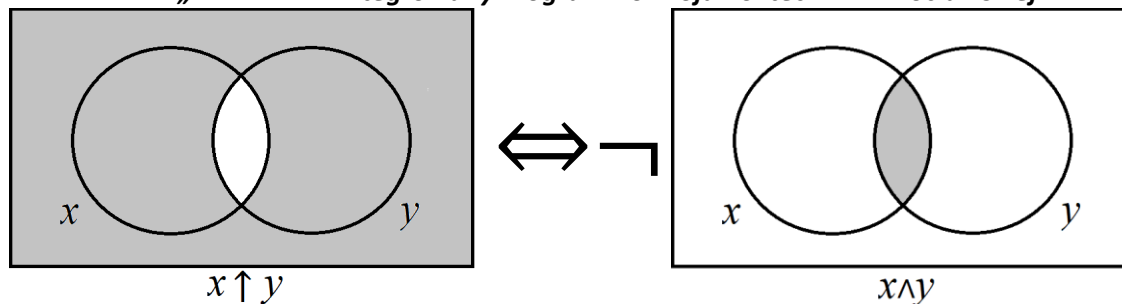


Diagram Venna dla dysjunkcji NAND



„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”



Bramka NAND

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

$$A \uparrow B = \neg A \vee \neg B$$

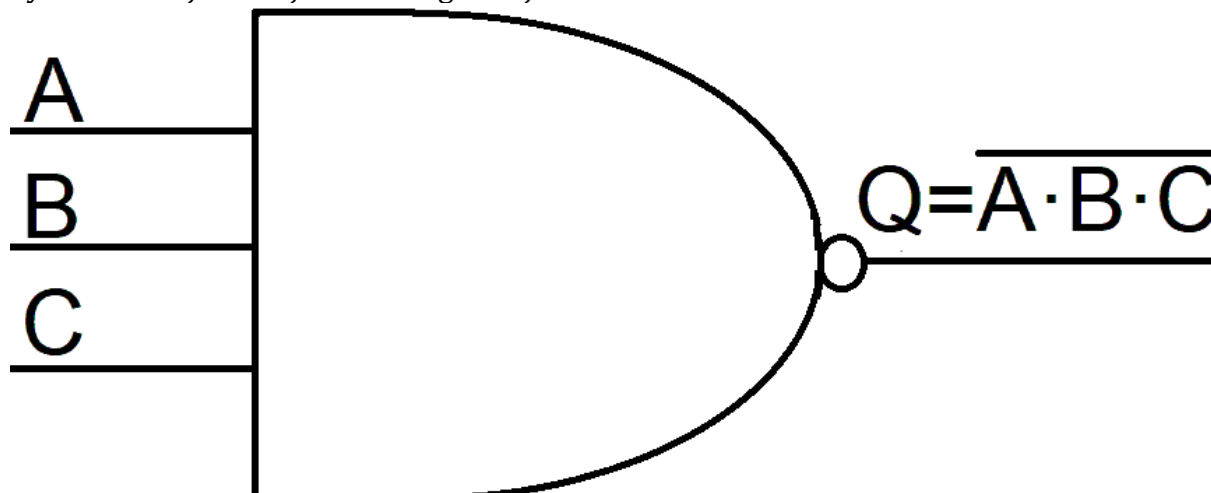
3-wejściowa bramka logiczna NAND

Wyrażenie logiczne: $Q = \overline{A \cdot B \cdot C}$

Tablica prawdy

A	B	C	Q
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Symbol 3-wejściowej bramki logicznej NAND



Zalety bramek NOR i NAND

Funktor dysjunkcji NAND posiada bardzo interesujące cechy, np. za jego pomocą można przedstawić wszystkie inne funkcje. Jedynym innym takim funktorem jest binegacja NOR. Dlatego też mówi się, że bramki NOR i NAND są bramkami uniwersalnymi lub że są one kluczowe dla nowoczesnej elektroniki cyfrowej.

Możliwość przedstawienia wszystkich możliwych funkcji za pomocą danego zbioru elementów, w przypadku funkcji NOR i NAND są to zbiory jednoelementowe, nazywa się systemami funkcjonalnie pełnymi.

Brama równoważności XNOR

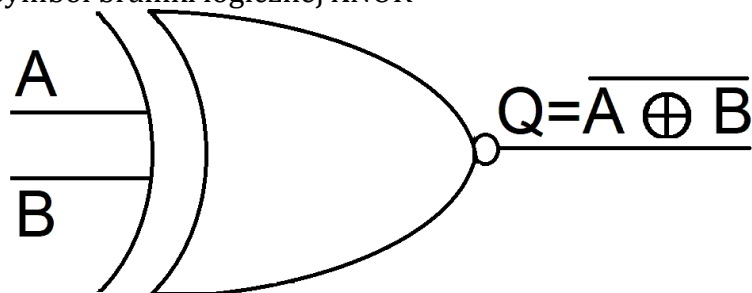
Negacja alternatywy wykluczającej (ang. *Exclusive NOR*). Bramka neguje wynik alternatywy wykluczającej. Jest nazywana **bramą równoważności**, ponieważ zwraca prawdę jedynie w przypadku, kiedy wartości zmiennych są sobie równe.

Występuje ona w niektórych opracowaniach dotyczących logiki lub bramek logicznych.

tablica prawdy dla funkcji XNOR

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

Symbol bramki logicznej XNOR



XNOR Gate – Na wyjściu jest stan “1” jeżeli obydwa wejścia jednocześnie mają ten sam stan logiczny “1” albo “0”.

Wyrażenie logiczne $Q = \overline{A \oplus B}$ – Należy czytać *Jeżeli A i B są takie same, Q = 1*

Realizacja funkcji XNOR z funkcji podstawowych: $Q = \bar{A} \cdot \bar{B} + A \cdot B$

Brama implikacji

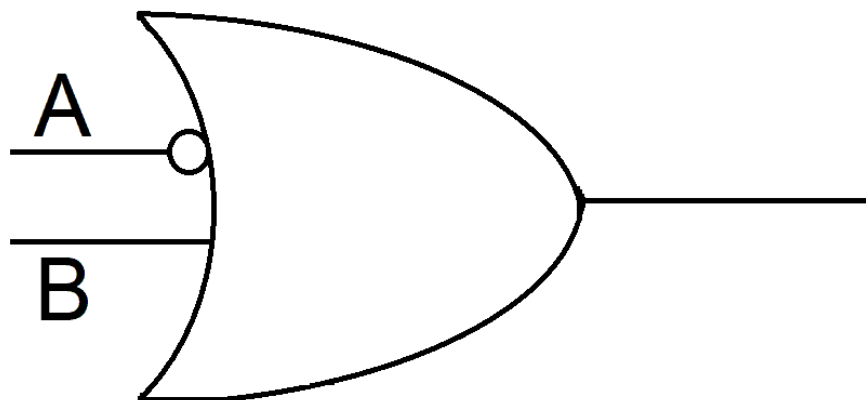
Brama implikacji (ang. *IMPLY gate*) jest to cyfrowa bramka logiczna, która implementuje funkcję implikacji.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

tablica prawdy dla bramki implikacji

A	B	$A \Rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

Symbol bramki logicznej implikacji



Dane wejściowe a wszystkie możliwe odpowiedzi

Porównanie stanu wyjść różnych bramek logicznych

Wejścia		Tablice prawdy dla porównania stanu wyjścia różnych bramek logicznych					
A	B	AND	NAND	OR	NOR	XOR	XNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Dla jednej zmiennej boolowskiej istnieją 4 możliwe wyrażenia jednoargumentowe określające rodzaje odpowiedzi w zależności od stanu zmiennej wejściowej (jeden argument wejściowy, jedna zwracana zmienna). Są one następujące:

- **Zawsze prawda** – niezależnie od argumentu zwraca prawdę,
- **Nigdy nie prawda** – niezależnie od argumentu zwraca fałsz,
- **Logiczna tożsamość** – jako wartość wyjściową zwraca wartość argumentu,
- **Negacja** – zwraca wartość przeciwną do wprowadzonego argumentu.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Wszystkie możliwe kombinacje odpowiedzi (różnego działania funkcji) w zależności od kombinacji danych wejściowych jednej zmiennej boolowskiej.

p	Zawsze prawda	Nigdy nie prawda	Logiczna tożsamość	Negacja
0	1	0	0	1
1	1	0	1	0

Wszystkie możliwe kombinacje odpowiedzi (różnego działania funkcji) w zależności od kombinacji danych wejściowych dwóch zmiennych boolowskich.

Wyrażenia dwuargumentowe

a	b	0 – ZERO – Zawsze fałsz	$a \text{ NOR } b$	$\neg(a \Leftarrow b)$; $\neg a \text{ AND } b$	$\neg(a \Rightarrow b)$	$a \text{ AND } b$	$\neg a$	$\neg b$	$a \text{ XNOR } b$; $a \Leftrightarrow b$	$a \text{ XOR } b$	b	a	$a \text{ NAND } b$	$a \Rightarrow b$	$a \Leftarrow b$; $a \text{ OR } \neg b$	$a \text{ OR } b$	1 – Zawsze prawda
0	0	0	1	0	0	0	1	1	1	0	0	0	1	1	1	0	1
0	1	0	0	1	0	0	1	0	0	1	1	0	1	1	0	1	1
1	0	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1	1
1	1	0	0	0	0	1	0	0	1	0	1	1	0	1	1	1	1

Systemy funkcjonalnie pełne

System funkcjonalnie pełny (ang. *Functional completeness*) to zbiór funkcji boolowskich, za pomocą których można wyrazić dowolną inną funkcję.

Zbiór taki nazywa się **minimalnym**, jeżeli po usunięciu dowolnego elementu z tego zbioru przestanie on być zbiorem funkcjonalnie pełnym.

System funkcjonalnie pełny stanowią trzy operanty: **negacji**, **sumy logicznej** (alternatywy) i **iloczynu logicznego** (koniunkcji). Taki zestaw nie jest zestawem minimalnym, ponieważ po usunięciu alternatywy albo koniunkcji nadal będzie zbiorem funkcjonalnie pełnym (brakującą funkcję można wyprowadzić za pomocą praw De Morgana).

Jednoelementowe

- Funkcja Sheffera (NAND) – $\{\uparrow\}$
- Funkcja Peirce'a (NOR) – $\{\downarrow\}$

Dwuelementowe

- Negacja i iloczyn logiczny $\{\neg, \wedge\}$ (sumę logiczną można wyprowadzić dzięki prawu De Morgana)

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

- Negacja i suma logiczna $\{\neg, \vee\}$ (iloczyn logiczny można wyprowadzić dzięki prawu De Morgana)
- Implikacja i negacja $\{\Rightarrow, \neg\}$
- Implikacja i zero logiczne/fałsz $\{\Rightarrow, \perp\}$
- I wiele innych

Funkcja Peirce'a (NOR)

Sposób przedstawienia podstawowych funkcji za pomocą wykonywania jedynie działań binegacji:

- **Negacja** $\neg p \Leftrightarrow p \downarrow p$ – sposób reprezentowania negacji przez binegację
- **Koniunkcja** $p \cdot q \Leftrightarrow (p \downarrow p) \downarrow (q \downarrow q)$ – sposób reprezentowania alternatywy przez binegację
- **Alternatywa** $p + q \Leftrightarrow \neg(p \downarrow q) \Leftrightarrow (p \downarrow q) \downarrow (p \downarrow q)$ – sposób reprezentowania koniunkcji przez binegację

Funkcja Sheffera (NAND)

Sposób przedstawienia podstawowych funkcji za pomocą wykonywania jedynie działań dysjunkcji:

- **Negacja** $\neg p \Leftrightarrow p \uparrow p$ – sposób reprezentowania negacji przez dysjunkcję
- **Koniunkcja** $p \cdot q \Leftrightarrow \neg\neg(p \cdot q) \Leftrightarrow \neg(p \uparrow q) \Leftrightarrow (p \uparrow q) \uparrow (p \uparrow q)$ – sposób reprezentowania koniunkcji przez dysjunkcję
- **Alternatywa** $p + q \Leftrightarrow \neg\neg(p + q) \Leftrightarrow \neg(\neg(p) + \neg(q)) \Leftrightarrow (p \uparrow p) \uparrow (q \uparrow q)$ – sposób reprezentowania alternatywy przez dysjunkcję
- **Implikacja** $p \Rightarrow q \Leftrightarrow p \uparrow (q \uparrow q) \Leftrightarrow p \uparrow (p \uparrow q)$ – sposób reprezentowania implikacji przez dysjunkcję

Implikacja i zero logiczne

Sposób przedstawienia podstawowych funkcji za pomocą wykonywania jedynie działań implikacji i zera logicznego:

- **Negacja** $\neg p \Leftrightarrow p \Rightarrow 0$ – sposób reprezentowania negacji przez implikację i zero logiczne
- **Koniunkcja** $p \cdot q \Leftrightarrow \neg(p \Rightarrow \neg q) \Leftrightarrow [p \Rightarrow (q \Rightarrow 0)] \Rightarrow 0$ – sposób reprezentowania koniunkcji przez implikację i zero logiczne
- **Alternatywa** $p + q \Leftrightarrow \neg p \Rightarrow q \Leftrightarrow (p \Rightarrow 0) \Rightarrow q$ – sposób reprezentowania alternatywy przez implikację i zero logiczne

Funkcje boolowskie

Definicja funkcji boolowskiej

W literaturze można spotkać 2 różne podejścia:

- A. funkcja boolowska może zwrócić wartość **dokładnie jednej zmiennej** – wtedy mówimy, że jest to funkcja przełączająca, ponieważ jako wynik końcowy zwraca ona tylko Prawdę (1) lub Fałsz (0).
- B. funkcja boolowska może zwrócić **wartości kilku zmiennych**, wtedy mówi się, że funkcja jest to odwzorowanie dwóch przestrzeni logicznych
 $f: \{0,1\}^m \rightarrow \{0,1\}^n$, gdzie m – liczba zmiennych wejściowych,
 n – liczba zmiennych wyjściowych

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Funkcja boolowska (ang. *Boolean function*) – funkcja logiczna jest to dowolne odwzorowanie

$f: A^x \rightarrow A$, gdzie

- $A = \{0,1\}$,
- x – liczba zmiennych wejściowych;
- wartości $\{0,1\}$ nie należy traktować jako liczb naturalnych, tylko jako sposób przedstawienia dwóch stanów logicznych „Prawdy” i „Fałszu”.

Funkcja boolowska o m wejściach i 1 wyjściu to funkcja

- $f: \{0,1\}^m \rightarrow \{0,1\}$
- Za pomocą funkcji boolowskiej opisuje się cyfrowe układy logiczne, w których zmienne wejściowe i zmienne wyjściowe przybierają tylko jeden z dwóch możliwych stanów

Funkcja boolowska może mieć wiele argumentów, natomiast na wyjściu zwraca wartości $y \in \{0, 1\}$, czyli taką, która może mieć tylko jeden z dwóch stanów: prawda (1) albo fałsz (0).

Z tego też powodu czasem nazywana jest funkcją przełączającą.

Funkcja służy do uzyskania oceny danej kombinacji wejściowych zmiennych logicznych.

- Funkcja boolowska składa się z
- Co najmniej jednej zmiennej boolowskiej,
- Co najmniej jednego operatora boolowskiego,
- Co najmniej jednego argumentu ze zbioru $\{0,1\}$

Aby zobaczyć, jak zachowuje się funkcja boolowska, przygotowuje się tablice prawdy. Przedstawia się w niej wszystkie kombinacje danych wejściowych i odpowiadający każdej kombinacji zwracany wynik.

W przypadku skomplikowanych funkcji tablice rozbija się, dodając kolumny pokazujący częściowe wyniki, czyli wyniki zwracane po wykonaniu kolejnych kroków.

Funkcja logiczna może mieć tzw. nieokreśloności, tzn. nigdy niewystępujące kombinacje zmiennych wejściowych. Taką funkcję zapisujemy w następujący sposób:

$f: \{0,1\}^m \rightarrow \{0,1, -\}$

Zapisy funkcji boolowskich

Typy zapisów

Funkcje boolowskie mogą być zapisywane na kilka różnych sposobów:

- **opis słowny**, czyli dla jeżeli wszystkie wejścia mają wartość 1, na wyjściu podaj wartość 0 itd.
- **tablice prawdy** (ang. *truth table*) – czyli pokazanie w tablicy jakie wartości otrzymają zmienne wyjściowe dla poszczególnych kombinacji zmiennych wejściowych,
- **tablice Karnaugh'a** (ang. *Karnaugh table*) inaczej nazywane **mapami Karnaugh'a** lub **siatkami Karnaugh'a** – w tych tablicach poszczególnym wierszom i kolumnom przypisane są odpowiednie kombinacje wartości zmiennych wejściowych.
- **Algebraiczny kanoniczny zapis funkcji:**
 - **Dysjunkcyjna/sumacyjna postać kanoniczna DNF (alternatywna)** – funkcja boolowska jest sumą iloczynów zmiennych wejściowych,
 - **Koniunkcyjna postać normalna CNF (koniunkcyjna)** – funkcja boolowska jest iloczynem sum zmiennych wejściowych,

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Funkcje boolowskie zupełne zwracają określone, zawsze takie same wartości dla każdej z możliwych kombinacji wartości zmiennych.

Funkcje niezupełne to takie, dla których dla pewnych kombinacji wartości zmiennych funkcja jest nieokreślona. W zapisach funkcji należy wyszczególnić te wartości argumentów, dla których funkcja jest nieokreślona.

Termy

Do zapisu funkcji używa się termów (definicje termów pochodzą z Wikipedii).

Term jest to wyrażenie składające się ze zmiennych oraz symboli funkcyjnych o dowolnej argumentowości z pewnego ustalonego zbioru.

Minterm – term składający się z literałów połączonych logicznym symbolem **koniunkcji**, który dla dokładnie jednej kombinacji wejść danej funkcji przyjmuje wartość 1. Minterm może być nazywany jako term iloczynowy.

Można zapamiętać to w następujący sposób: funkcja zwraca wartość minimalną ze zbioru literałów w tym termie.

$\min(\text{zbiór_literałów})$

$$X \cdot Y \cdot Z = \min(X, Y, Z)$$

Jeżeli choć jeden literał jest równy zero, minterm jest równy zero.

Minterm zawiera wszystkie literały danej funkcji.

Maksterm (maxterm) – term składający się z literałów połączonych logicznym symbolem **alternatywy**, który dla dokładnie jednej kombinacji wejść danej funkcji przyjmuje wartość 0. Maksterm może być nazywany jako term sumacyjny.

Można zapamiętać to w następujący sposób: funkcja zwraca wartość maksymalną ze zbioru literałów w tym termie.

$\max(\text{zbiór_literałów})$

$$X + Y + Z = \max(X, Y, Z)$$

Jeżeli choć jeden literał jest równy jeden, maksterm jest równy jeden.

Maksterm zawiera wszystkie literały danej funkcji.

Każdą funkcję boolowską można przedstawić w postaci sumy logicznej iloczynów (mintermów), dla których wartość funkcji wynosi 1.

Analogicznie każdą funkcję boolowską można przedstawić w postaci iloczynu logicznego sumy (makstermów), dla których wartość funkcji wynosi 0.

Dysjunkcyjna/sumacyjna postać normalna

Dysjunkcyjna/sumacyjna postać normalna/kanoniczna DNF (ang. *disjunctive normal form*) w skrócie nazywana jako postać sumacyjna (alternatywna), czasem nazywana sumacyjną postacią kanoniczną, jest to funkcja boolowska, która jest **sumą iloczynów (mintermów)** zmiennych wejściowych.

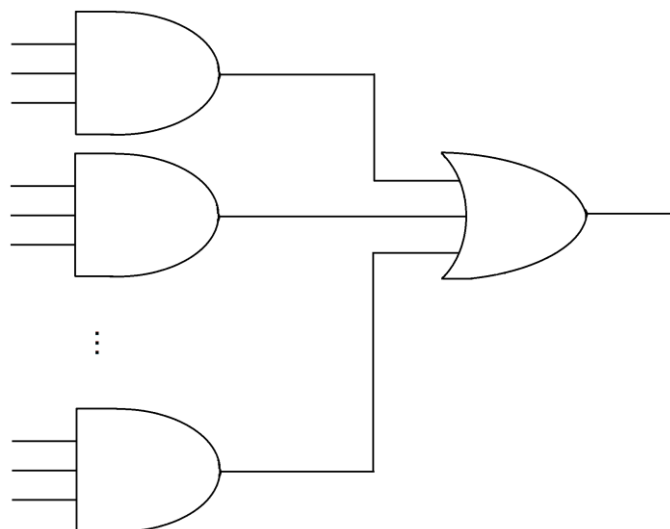
Taka postać formuły logicznej oznacza, że jest ona zapisana w postaci dysjunkcji (alternatywy) klauzul dualnych.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Klauzula dualna (ang. *dual clause*) jest to koniunkcja skończonego zbioru literałów. Jest ona prawdziwa wtedy i tylko wtedy, gdy każdy z literałów jest prawdziwy. Klauzulę dualną pustą uznaje się za zawsze prawdziwą.

Postać ogólna:

$$DNF = f(\dots) = (\dots \cdot \dots \cdot \dots) + (\dots \cdot \dots \cdot \dots) + (\dots \cdot \dots \cdot \dots) + \dots$$



Przykładowa sumacyjna postać normalna funkcji:

$$y = x_0 \bar{x}_1 \bar{x}_2 + \bar{x}_0 x_1 \bar{x}_2 + x_0 \bar{x}_1 x_2 + \bar{x}_0 x_1 x_2 + x_0 x_1 x_2$$

Jak na podstawie tablicy prawdy stworzyć sumacyjną postać normalną?

Mając tabelę prawdy poszukuje się wierszy, w których uzyskuje się wartość na wyjściu 1, następnie tworzy się zapis sumy tych kombinacji zmiennych wejściowych, które na wyjściu dają wartość 1. Dzięki temu możemy zbudować poszczególne mintermy, czyli termy. Dla każdego minterma danej funkcji przyjmuje wartość 1 dla dokładnie jednej kombinacji wejść. Sumacyjna postać normalna jest sumą iloczynów.

A	B	C _i	S - suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	0	0
0	0	1	1
0	1	1	0
1	0	1	0
1	1	1	1

$$y = f(A, B, C_i) = \bar{A} \cdot B \cdot \bar{C} + A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot C + A \cdot B \cdot C$$

Ta suma będzie równa jeden, jeżeli choć jeden składnik sumy, czyli choć jedno wyrażenie w nawiasie będzie równe jeden. Jeżeli nie zdarzy się żaden taki przypadek, to funkcja będzie zwracała wartość 0.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Zapis funkcji

Aby określić wartości argumentów, dla których funkcja $f()$ zwraca wartość 1 zazwyczaj używa się np. następującego zapisu:

$$f(x_1, x_2, \dots, x_n) = \Sigma(d_1, d_2, \dots, d_n)$$

gdzie:

Σ – oznacza, że chodzi tu o sumę, czyli o sumacyjną postać normalną,

x_i – argumenty funkcji, dla $(0 < i < n)$,

d_j – liczby dziesiętne, które odpowiadają odpowiedniej kombinacji wartości argumentów x_1, x_2, \dots, x_n dla których funkcja zwraca wartość $f() = 1$; $(0 < j < n)$.

$$f(a, b, c) = \Sigma(1, 3, 6)_{abc} = \Sigma(001, 011, 110)_{abc} = (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c)$$

Jeżeli funkcja jest niezupełna, czyli dla pewnych kombinacji argumentów zwracana przez nią wartość jest nieokreślona (nie zależy nam na wyniku, wynik jest nieznany), wtedy takie argumenty podajemy w kolejnym nawiasie:

$$f(a, b, c) = \Sigma(2, 4, 6(1, 3))_{abc} = \Sigma(010, 100, 110, (001, 011))_{abc}$$

można się również spotkać z innym zapisem tej samej funkcji:

$$f(a, b, c) = \Sigma m(2, 4, 6) + d(1, 3) = \Sigma m(010, 100, 110) + d(001, 011)$$

m – oznacza tu minitermy (ang. *minterms*)

d – oznacza wartości nieokreślone (ang. *don't care*)

W przypadku kombinacji wartości zmiennych:

$a=0, b=0, c=1$ oraz $a=0, b=1, c=1$ funkcja $f(a, b, c)$ jest nieokreślona.

Przykłady funkcji w postaci DNF

$$(\neg A \cdot B \cdot \neg C) + (D \cdot \neg E \cdot \neg F)$$

$$(A \cdot \neg B) + \neg D$$

$$\neg A \cdot C$$

$$D$$

Przykłady funkcji, które nie są w postaci DNF:

$$\neg(A + C)$$

$$\neg(A \cdot C) + D$$

Koniunkcyjna/iloczynowa postać normalna

Koniunkcyjna/iloczynowa postać normalna CNF (ang. *conjunctive normal form*)

w skrócie nazywana jako postać iloczynowa (koniunkcyjna), czasem nazywana iloczynową postacią kanoniczną, danej formuły logicznej jest to funkcja boolowska, która jest **iloczyn sum (makstermów)** zmiennych wejściowych.

Taka postać formuły logicznej oznacza, że jest ona zapisana w postaci koniunkcji klauzul.

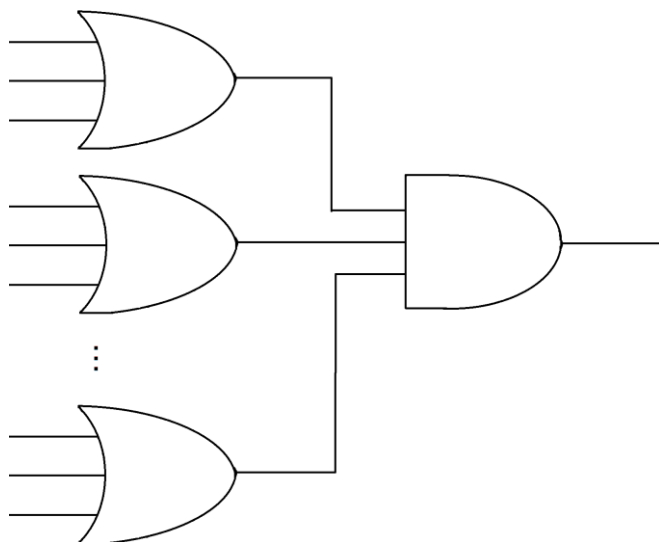
„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Klauzula (ang. *clause*) – zbiór formuł logicznych. Klauzulę nazywamy prawdziwą wtedy i tylko wtedy, gdy alternatywa jej formuł logicznych jest prawdziwa. Klauzula pusta jest zawsze fałszywa.

Taka postać formuły logicznej oznacza, że jest ona zapisana w postaci iloczynu sum zmiennych wejściowych.

Wzór ogólny

$$CNF = f(\dots) = (\dots + \dots + \dots) \cdot (\dots + \dots + \dots) \cdot (\dots + \dots + \dots) \cdot \dots$$



Przykładowa iloczynowa postać normalna funkcji:

$$y = (x_0 + x_1 + x_2) \cdot (\bar{x}_0 + \bar{x}_1 + x_2) \cdot (x_0 + x_1 + \bar{x}_2)$$

Jak na podstawie tablicy prawdy iloczynową postać normalna?

Mając tabelę prawdy poszukuje się wierszy, w których uzyskuje się wartość na wyjściu 0, następnie tworzy się zapis sumy tych kombinacji zmiennych wejściowych, które na wyjściu dają wartość 0. Dzięki temu możemy zbudować poszczególne makstermy. Dla każdego maksterma danej funkcji przyjmuje wartość 0 dla dokładnie jednej kombinacji wejść. Iloczynowa postać normalna jest iloczynem sum. Każde jedno wyrażenie w nawiasie (...) odpowiada jednemu wierszowi, w którym kombinacja zmiennych zwraca wartość 0.

A	B	C _i	S – suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	0	0
0	0	1	1
0	1	1	0
1	0	1	0
1	1	1	1

$$y = f(A, B, C_i) = (A + B + C) \cdot (\bar{A} + \bar{B} + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + \bar{C})$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Iloczyn będzie równy zero, jeżeli choć jeden składnik iloczynu, czyli choć jedno wyrażenie w nawiasie będzie równe zero. Jeżeli nie zdarzy się taki przypadek, to w funkcja będzie zwracała wartość 1.

Zapis funkcji

Aby określić wartości argumentów, dla których funkcja $f()$ zwraca wartość 0 zazwyczaj używa się np. następującego zapisu:

$$f(x_1, x_2, \dots, x_n) = \Pi(d_1, d_2, \dots, d_n)$$

gdzie:

Π – oznacza, że chodzi tu o iloczyn, czyli o iloczynową postać normalną,

x_i – argumenty funkcji, dla $(0 < i < n)$,

d_j – liczby dziesiętne, które odpowiadają odpowiedniej kombinacji wartości argumentów x_1, x_2, \dots, x_n dla których funkcja zwraca wartość $f() = 0$; $(0 < j < n)$.

$$f(a, b, c) = \Pi(1, 3, 6)_{abc} = \Pi(001, 011, 110)_{abc} = (a \vee b \vee \neg c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$$

Jeżeli funkcja jest niezupełna, czyli dla pewnych kombinacji argumentów zwracana przez nią wartość jest nieokreślona (nie zależy nam na wyniku, wynik jest nieznany), wtedy takie argumenty podajemy w kolejnym nawiasie:

$$f(a, b, c) = \Pi(1, 3, 6(0, 2))_{abc} = \Pi(001, 011, 110(000, 010))_{abc} = (a \vee b \vee \neg c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$$

można się również spotkać z innym zapisem tej samej funkcji:

$$f(a, b, c) = \Pi m(1, 3, 6) + d(0, 2) = \Pi m(001, 011, 110) + d(000, 010)$$

m – oznacza tu makstermy (ang. *maxterms*)

d – oznacza wartości nieokreślone (ang. *don't care*)

W przypadku kombinacji wartości zmiennych:

$a=0, b=0, c=0$ oraz $a=0, b=1, c=0$ funkcja $f(a, b, c)$ jest nieokreślona.

Przykłady funkcji w postaci CNF

$$(\neg A + B + \neg C) \cdot (D + \neg E + \neg F)$$

$$(A + \neg B) \cdot \neg D$$

$$\neg A + C$$

$$\neg A \cdot C$$

$$D$$

Przykłady funkcji, które nie są w postaci CNF:

$$\neg(A + C)$$

$$(A \cdot C) + D$$

$$\neg(A \cdot C)$$

			Klauzule dualne. Składniki iloczynu dla postaci sumacyjnej (muszą się równać jeden, dlatego negujemy zera)	Klauzule. Składniki sumy dla postaci iloczynowej (muszą się równać zero, dlatego negujemy jedynki)
X	Y	Z		
0	0	0	$\bar{X} \cdot \bar{Y} \cdot \bar{Z} = \min(\bar{X}, \bar{Y}, \bar{Z})$	$X + Y + Z = \max(X, Y, Z)$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

0	0	1	$\bar{X} \cdot \bar{Y} \cdot Z = \min(\bar{X}, \bar{Y}, Z)$	$X + Y + \bar{Z} = \max(X, Y, \bar{Z})$
0	1	0	$\bar{X} \cdot Y \cdot \bar{Z} = \min(\bar{X}, Y, \bar{Z})$	$X + \bar{Y} + Z = \max(X, \bar{Y}, Z)$
0	1	1	$\bar{X} \cdot Y \cdot Z = \min(\bar{X}, Y, Z)$	$X + \bar{Y} + \bar{Z} = \max(X, \bar{Y}, \bar{Z})$
1	0	0	$X \cdot \bar{Y} \cdot \bar{Z} = \min(X, \bar{Y}, \bar{Z})$	$\bar{X} + Y + Z = \max(\bar{X}, Y, Z)$
1	0	1	$X \cdot \bar{Y} \cdot Z = \min(X, \bar{Y}, Z)$	$\bar{X} + Y + \bar{Z} = \max(\bar{X}, Y, \bar{Z})$
1	1	0	$X \cdot Y \cdot \bar{Z} = \min(X, Y, \bar{Z})$	$\bar{X} + \bar{Y} + Z = \max(\bar{X}, \bar{Y}, Z)$
1	1	1	$X \cdot Y \cdot Z = \min(X, Y, Z)$	$\bar{X} + \bar{Y} + \bar{Z} = \max(\bar{X}, \bar{Y}, \bar{Z})$

Tablice Karnaugh'a

Tworzenie mapy Karnaugh'a

W takich tablicach część zmiennych jest przypisywanych do wierszy, a część do kolumn. Jeżeli jest parzysta liczba zmiennych, to taka sama zmiennych przypisana jest do wierszy, jak i do kolumn. Wiersze i kolumny numerowane są przy wykorzystaniu kodu Grey'a. Dla danej kolumny wektor danych wejściowych, czyli kombinacja zmiennych wejściowych, powstaje po zestawieniu binarnego kodu wiersza z binarnym kodem kolumny.

Jakim kombinacjom czterech zmiennych odpowiadają poszczególne pola mapy Karnaugh'a dla funkcji $F(A,B,C,D)$

		AB			
		00	01	11	10
CD	00	$F(0,0,0,0)$	$F(0,1,0,0)$	$F(1,1,0,0)$	$F(1,0,0,0)$
	01	$F(0,0,0,1)$	$F(0,1,0,1)$	$F(1,1,0,1)$	$F(1,0,0,1)$
	11	$F(0,0,1,1)$	$F(0,1,1,1)$	$F(1,1,1,1)$	$F(1,0,1,1)$
	10	$F(0,0,1,0)$	$F(0,1,1,0)$	$F(1,1,1,0)$	$F(1,0,1,0)$

Dzięki wykorzystaniu kodu Grey'a możliwe jest znalezienie, w sposób wizualny, pól, które mają taką samą wartość logiczną.

Przykładowe sąsiedztwa logiczne dla funkcji o 6 zmiennych:

		ABC							
		000	001	011	010	110	111	101	100
DEF	000								
	001								
	011								
	010								
	110								
	111								
	101								
	100								

Przykładowe tablice Karnaugh'a dla sumatora 1 bitowego

S	AB			
	00	01	11	10
C_i	0	0	1	0
	1	1	0	1

$$S = \bar{A} \cdot \bar{B} \cdot C_i + \bar{A} \cdot B \cdot \bar{C}_i + A \cdot B \cdot C_i + A \cdot \bar{B} \cdot \bar{C}_i$$

C_{out}	AB			
	00	01	11	10
C_i	0	0	0	1
	1	0	1	1

A	B	C _i	S - suma	C _{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$C_{out} = C \cdot B + C \cdot A + A \cdot B$$

Algorytm postępowaniu przy minimalizacji

- Należy stworzyć mapę,
- Wypełnić ją wartościami funkcji {0, 1} zwracanymi dla danej kombinacji stanów argumentów wejściowych,
- Grupuje się sąsiadujące pola o tych samych wartościach (jeżeli chce się otrzymać normalną postać sumacyjną, należy grupować jedynki, natomiast jeżeli iloczynową – wtedy zera),
- Kolejne grupy muszą mieć kształt prostokąta – mogą one wychodzić poza krawędź prostokąta i kończyć się z jego drugiej strony,
- Wymiary (długość/szerokość) prostokąta muszą być potęgami liczby 2^n , gdzie $n \in \mathbb{N}$ (czyli 1,2,4,8),
- Dąży się do tego, aby uzyskać jak najmniej grup. Jedno pole może należeć do kilku grup.
- Aby uzyskać jak największy stopień minimalizacji należy dążyć do tego, aby te prostokąty były jak największe, czyli żeby obejmowały jak najwięcej pól.
- Następnie na podstawie uzyskanych prostokątów zapisuje się funkcję w postaci uproszczonej. Przy tworzeniu minimalnej sumacyjnej postaci normalnej należy zapewnić, że w jej zapisie również znajdą się kombinacje zmiennych, dla których funkcja zwraca wartość 1, a które nie zostały objęte żadnym prostokątem (nie wchodzą w skład żadnej grupy).
- Przy tworzeniu minimalnej iloczynowej postaci normalnej należy zwracać uwagę, żeby zapis funkcji uwzględniał wszystkie prostokąty i wszystkie pola z wartościami równymi 0, które nie wchodzą w skład żadnej z grup.

Numerowanie kolumn, wierszy i pól w tablicach Karnaugh'a

2 zmienne

Y	A	
	00	01
B	0	0
	1	1

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

3 zmienne

		<i>Y</i>			
		<i>AB</i>			
		00	01	11	10
<i>C</i>	0	0	2	6	4
	1	1	3	7	5

		<i>Y</i>			
		<i>AB</i>			
		00	01	11	10
<i>C</i>	0	000	010	110	100
	1	001	011	111	101

4 zmienne

		<i>Y</i>			
		<i>AB</i>			
		00	01	11	10
<i>CD</i>	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

5 zmiennych

		<i>Y</i>							
		<i>ABC</i>							
		000	001	011	010	110	111	101	100
<i>DE</i>	00	0	4	12	8	24	28	20	16
	01	1	5	13	9	25	29	21	17
	11	3	7	15	11	27	31	23	19
	10	2	6	14	10	26	30	22	18

6 zmiennych

		<i>Y</i>							
		<i>ABC</i>							
		000	001	011	010	110	111	101	100
<i>DEF</i>	000	0	8	24	16	48	56	40	32
	001	1	9	25	17	49	57	41	33
	011	3	11	27	19	51	59	43	35
	010	2	10	26	18	50	58	42	34
	110	6	14	30	22	54	62	46	38
	111	7	15	31	23	55	63	47	39
	101	5	13	29	21	53	61	45	37
	100	4	12	28	20	52	60	44	36

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Sąsiedztwa – niektóre przykłady

3 zmienne

		<i>AB</i>			
		00	01	11	10
<i>C</i>	0	0	2	6	4
	1	1	3	7	5

		<i>AB</i>			
		00	01	11	10
<i>C</i>	0	0	2	6	4
	1	1	3	7	5

		<i>AB</i>			
		00	01	11	10
<i>C</i>	0	0	2	6	4
	1	1	3	7	5

		<i>AB</i>			
		00	01	11	10
<i>C</i>	0	0	2	6	4
	1	1	3	7	5

4 zmienne

		<i>AB</i>			
		00	01	11	10
<i>CD</i>	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

		<i>AB</i>			
		00	01	11	10
<i>CD</i>	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

		<i>AB</i>			
		00	01	11	10
<i>CD</i>	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

		AB			
		00	01	11	10
CD	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

		AB			
		00	01	11	10
CD	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

		AB			
		00	01	11	10
CD	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

Wykorzystanie tablic Karnaugh'a do upraszczania funkcji

W algorytmicznej metodzie map/tablic/siatek Karnaugh'a stosuje się tzw. regułę sklejania. Tablice Karnaugh'a umożliwiają „sklejanie” ze sobą nadmiernych zmiennych.

$(A + B) \cdot (A + \bar{B}) = A$ – sposoby upraszczania dla postaci iloczynu sum

$(A \cdot B) + (A \cdot \bar{B}) = A$ – sposoby upraszczania dla postaci sumy iloczynów

A, B – dowolna zmienna lub funkcja boolowska

Sumacyjna postać normalna (DNF)

		AB			
		00	01	11	10
C_i	0	0	1	0	1
	1	1	0	1	0

$$S = \bar{A} \cdot \bar{B} \cdot C_i + \bar{A} \cdot B \cdot \bar{C}_i + A \cdot B \cdot C_i + A \cdot \bar{B} \cdot \bar{C}_i$$

		AB			
		00	01	11	10
C_i	0	0	0	1	0
	1	0	1	1	1

$$C_{out} = C \cdot B + C \cdot A + A \cdot B$$

Iloczynowa postać normalna (CNF)

		AB			
		00	01	11	10
C_i	0	0	1	0	1
	1	1	0	1	0

$$S = (\bar{A} + \bar{B} + C_i) \cdot (\bar{A} + B + \bar{C}_i) \cdot (A + B + C_i) \cdot (A + \bar{B} + \bar{C}_i)$$

		AB			
		00	01	11	10
C_i	0	0	0	1	0
	1	0	1	1	1

$$C_{out} = (B + C)(A + C)(A + B)$$

Jeżeli wykorzystywana funkcja nie jest określona dla jakiejś kombinacji danych wejściowych, czyli nie jest istotna wartość, jaką funkcja zwraca w takim przypadku, wtedy takie pole może być dodane do jakiegoś prostokąta (do jakiejś grupy) w celu powiększenia go.

Przykład

		AB			
		00	01	11	10
CD	00	1	1	1	1
	01	1	0	1	1
	11	1	0	0	0
	10	0	0	0	0

$$F = \bar{C}\bar{D} + A\bar{C} + \bar{A}\bar{B}D$$

$$F = (\bar{C} + D) \cdot (\bar{A} + \bar{C}) \cdot (A + \bar{B} + \bar{D})$$

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

		AB			
		00	01	11	10
CD	00	1	1	1	1
	01	1	x	1	1
	11	1	0	0	0
	10	0	0	0	0

x – stan nieokreślony dla funkcji

$$F = \bar{C} + \bar{A}\bar{B}D$$

$$F = (\bar{C} + D) \cdot (\bar{A} + \bar{C}) \cdot (\bar{B} + \bar{C})$$

UWAGA!!!!

		AB			
		00	01	11	10
CD	00	0	1	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	0	1	1	1

Suma iloczynów (DNF)

$$F = (A) + (B) + (D) = A + B + D$$

Iloczyn sum (CNF)

$$F = (A + B + D) = A + B + D$$

		AB			
		00	01	11	10
CD	00	0	0	0	0
	01	0	0	0	0
	11	0	0	0	1
	10	0	0	0	1

Suma iloczynów (DNF)

$$F = (A \cdot \bar{B} \cdot C) = A \cdot \bar{B} \cdot C$$

Iloczyn sum (CNF)

$$F = (A) \cdot (\bar{B}) \cdot (C) = A \cdot \bar{B} \cdot C$$

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Można zapytać, jaka będzie wartość funkcji, jeżeli $D=1$? Przecież tej wartości nie ma w równaniu funkcji. Wartość funkcji jest wyliczana dla całej kombinacji zmiennych, a nie tylko dla wartości jednej ze zmiennych. Jeżeli $D=1$, to wartość funkcji będzie zależała od tego, ile równają się zmienne A , B i C . A każda z tych zmiennych musi mieć jakąś wartość 0 albo 1.

Metoda implikantów prostych

Metoda implikantów prostych (ang. *method of prime implicants*) nazywana jest również metodą Quine’a – McCluskeya (ang. *algorithm Quine – McCluskey*). Jest ona stosowana dla funkcji o wielu zmiennych. Im więcej tych zmiennych, tym zwiększa się jej efektywność w porównaniu do innych metod minimalizacyjnych.

Jest to metoda w zasadzie bazująca na tym samym podejściu (taki sam mechanizm upraszczania), co metoda siatek Karnaugh’a tyle, że metoda Karnaugh’a jest „graficzna” natomiast implikantów prostych – „tekstowa”.

W metodzie Karnaugh’a należy pilnować, aby grupy zakreślały wszystkie wymagane pola (wszystkie jedynki albo wszystkie zera – w zależności od podejścia). Natomiast tu należy wypisać wszystkie kombinacje wartości zmiennych, dla których funkcja zwraca wartość jeden – dla sumacyjnej postaci normalnej (albo zero – dla iloczynowej postaci normalnej)

Implikant funkcji boolowskiej f – taki iloczyn literałów, że dla wszystkich wektorów binarnych $x=(x_1, \dots, x_n)$, dla których jest on równy jedności, funkcja f jest równa jedności.

Chociaż bardziej praktyczny niż mapowanie Karnaugh’a w przypadku więcej niż czterech zmiennych, algorytm Quine-McCluskey ma również ograniczony zakres zastosowań. Czas działania algorytmu Quine – McCluskey rośnie wykładniczo wraz z liczbą zmiennych.

Algorytm postępowania przy minimalizacji funkcji

Rozpisuje się wyrażenia, dla których funkcja boolowska zwraca wartość zero, Uporządkowuje się te wyrażenia grupując je w taki sposób, aby w każdej grupie były takie wyrażenia (implikanty), które mają taką samą liczbę zer/jedynek.

Kolejna grupa powinna zawierać więcej jedynek(zer), niż poprzednia,

Aby znaleźć implikanty proste każdy element w jednej grupie porównuje się z każdym elementem grupy sąsiedniej, która zawiera o więcej o jedną zer/jedynek.

Jeżeli znajdzie się elementy, które różnią się między sobą jedynie jednym indeksem, wtedy się je grupuje w parę, zaznacza i w miejsce tego indeksu wpisuje się kreskę „-”, a następnie przepisuje jako jeden element.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Lp.	A	B	C	D	F
a0	0	0	0	0	1
a1	0	0	0	1	0
a2	0	0	1	0	1
a3	0	0	1	1	0
a4	0	1	0	0	1
a5	0	1	0	1	1
a6	0	1	1	0	0
a7	0	1	1	1	0
a8	1	0	0	0	1
a9	1	0	0	1	0
a10	1	0	1	0	1
a11	1	0	1	1	1
a12	1	1	0	0	0
a13	1	1	0	1	0
a14	1	1	1	0	0
a15	1	1	1	1	1

Liczba jedynek	Lp.	Wyrażenie
0	a0	0000
1	a2 a4 a8	0010 0100 1000
2	a5 a10	0101 1010
3	a11	1011
4	a15	1111

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Lp.	Wyrażenie	Implikanty rozmiaru 2	Implikanty rozmiaru 4
a0 a2	0000 0010	00-0 (a0,a2)	-0-0* (a0,a2,a8,a10)
a0 a4	0000 0100	0-00* (a0,a4)	
a0 a8	0000 1000	-000 (a0,a8)	-0-0 (a0,a8,a2,a10)
a2 a10	0010 1010	-010 (a2,a10)	
a4 a5	0100 0101	010-* (a4,a5)	
a8 a10	1000 1010	10-0 (a8,a10)	
a10 a11	1010 1011	101-* (a10,a11)	
a15 a11	1111 1011	1-11* (a15,a11)	

W tabeli wypisuje się te implikanty, które zostały zaznaczone gwiazdkami, jako istotne. Jeżeli dla danej kolumny występuje tylko jeden znak „X”, to odpowiadający mu implikant zaznacza się, jako konieczny. Te znaki X zostały tu pogrubione.

- implikanty pierwszy, trzeci i piąty są konieczne, ponieważ obsługują takie przypadki, jakich nie obsługują inne implikanty (są one zaznaczone pogrubioną literą X),
- drugi może być zastąpiony przez pierwszy i trzeci,
- czwarty implikant może być zastąpiony przez pierwszy i piąty.

W przypadku minimalizowania funkcji przy wykorzystaniu tej metody można otrzymać kilka równoważnych równań.

	0	2	4	5	8	10	11	15		A	B	C	D
(a0,a2,a8,a10)	X	X			X	X				-	0	-	0
(a0,a4)	X		X							0	-	0	0
(a4,a5)			X	X						0	1	0	-
(a10,a11)						X	X			1	0	1	-
(a15,a11)							X	X		1	-	1	1

$$Y = \overline{B}\overline{D} + \overline{A}BC + ACD$$

Nieokreślone wartości funkcji.

Jeżeli dla jakiejś kombinacji zmiennych nie jest istotne, jaką wartość zwraca funkcja lub funkcja ma wartość nieokreśloną, to takie wartości wpisuje się, jako implikanty proste.

Lp.	A	B	C	D	F
a0	0	0	0	0	1
a1	0	0	0	1	0
a2	0	0	1	0	1
a3	0	0	1	1	0
a4	0	1	0	0	X
a5	0	1	0	1	X
a6	0	1	1	0	0
a7	0	1	1	1	0
a8	1	0	0	0	1
a9	1	0	0	1	0
a10	1	0	1	0	1
a11	1	0	1	1	1
a12	1	1	0	0	0
a13	1	1	0	1	0
a14	1	1	1	0	0
a15	1	1	1	1	1



W tych przypadkach nie jest określone, jaką wartość zwraca funkcja.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Grupowanie

Lp.	Wyrażenie	Implikanty rozmiaru 2	Implikanty rozmiaru 4
a0 a2	0000 0010	00-0 (a0,a2)	-0-0* (a0,a2,a8,a10)
a0 a4	0000 0100	0-00* (a0,a4)	
a0 a8	0000 1000	-000 (a0,a8)	-0-0 (a0,a8,a2,a10)
a2 a10	0010 1010	-010 (a2,a10)	
a4 a5	0100 0101	010-* (a4,a5)	
a8 a10	1000 1010	10-0 (a8,a10)	
a10 a11	1010 1011	101-* (a10,a11)	
a15 a11	1111 1011	1-11.* (a15,a11)	

Implikant istotny	0	2	4	5	8	10	11	15		A	B	C	D
(a0,a2,a8,a10)	X	X			X	X				-	0	-	0
(a0,a4)	X		X							0	-	0	0
(a4,a5)			X	X						0	1	0	-
(a10,a11)						X	X			1	0	1	-
(a15,a11)							X	X		1	-	1	1

W tabeli wypisuje się te implikanty, które zostały zaznaczone gwiazdkami jako istotne. Kombinacje 4 i 5 są przypadkami, kiedy wartość zwracana przez funkcję jest nieokreślona lub nieistotna.

W tym przypadku okazuje się, że:

- trzeci implikant jest niepotrzebny, ponieważ obsługuje tylko nieistotne przypadki,
- pierwszy i piąty implikanty są konieczne, ponieważ obsługują takie przypadki, jakich nie obsługują inne implikanty (zaznaczone pogrubionymi znakami X),
- Drugi może być zastąpiony przez pierwszy,
- Czwarty implikant może być zastąpiony przez pierwszy i piąty.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

W tym bowiem przypadku nie ma potrzeby wypisywania wyrażeń dla nieistotnych wartości funkcji.

Zatem funkcja będzie wyglądała następująco:

$$Y = \overline{B}\overline{D} + ACD$$