Reprezentacja danych w komputerze

W obliczeniach komputerowych wszystkie liczby, teksty, kolory są reprezentowane ciągami **binarnymi**.

System pozycyjny zapisu liczb

Liczby w zapisie pozycyjnym (np. 32 875) są przedstawiane, jako ciągi cyfr (c_i): (liczbo naturalnych):

$$c_i \in \{0,1,.., R-1\}$$

np. system dziesiętny- podstawa R=10 - ma 10 cyfr: $c_i \in \{0,1,2,. . . ,9\}$

Podstawą R pozycyjnego systemu liczenia jest ilość różnych symboli (cyfr).

Natomiast rzeczywista wartość cyfry w liczbie zależy od jej pozycji zajmowanej w łańcuchu cyfr.
Np. cyfra 5 w liczbie, 535 co innego znaczy. Wartość liczby L jest sumą cyfr mnożonych przez wagi pozycji:

$$L_{(R)} = \sum_{i=0}^{n} c_i * R^i \qquad (1)$$

gdzie: i – numery pozycji na lewo od przecinka po części całkowitej liczby

Np. korzystając z wzoru (1) liczbę $32\,875_{10}$ można zapisać jako:

3	2	8	7	5	
10^4	10^3	10^2	10^1	10^0	← wagi kolejnych cyfr systemu

W podobny sposób można też przedstawić liczby Lu <1 tzn. liczby ułamkowe:

$$Lu_{(R)} = \sum_{i=1}^{n} c_i * R^{-i} \qquad (2)$$

Np. korzystając z (2) liczbę 0,625 można przedstawić jako:

$0.626 = 6*10^{-1} + 2*10^{-2} \quad + \quad 5*10^{-3}$

Korzystając z (1) można zapisać liczbę w systemie o podstawie R i odczytać jej wartość dziesiętną:

Sytem R=8 $\quad (237)_8 = 2*8^2 + 3*8^1 + 7*8^0 = 159_d$

System R=4 $\quad (233)_4 = 2*4^2 + 3*4^1 + 3*4^0 = 47_d$

System R=2 $\quad (101)_2 = 1*2^2 + 0*2^1 + 1*2^0 = 5_d$

Reprezentacja liczb całkowitych- kod binarny

Podstawą kodu binarnego jest podstawa R=2. W kodzie występują 2 cyfry: 0 i 1. Kod jest idealny do zapisu informacji w komputerze, bo komputery rozróżniają tylko dwa stany: 0 i 1, które zawiera najmniejsza porcja informacji nazywana **bitem [b]**. Zwykle stosuje się większą jednostkę **-bajt [B].** Bajt to 8 bitów. W komputerach używa się także systemów: szesnastkowego (heksadecymalny) i rzadziej- ósemkowego **(oktalny).**

Każda liczba L może być zapisana na pewnej ilości n bitów. Ilość n bitów niezbędną do zapisu liczby dziesiętnej L można wyznaczyć z nierówności:

$$(2^n - 1) \geq L \qquad (3)$$

Np. minimalna liczba bitów niezbędna dla zapisu liczby 255 to n=8 bitów.

W kodzie binarnym inaczej przedstawiane są liczby **całkowite nieujemne**, a inaczej **ujemne**. Inaczej też zapisuje się liczby **ułamkowe**.

Natomiast w przypadku **liczb rzeczywistych** część całkowita jest zapisywana w postaci jednego ciągu bitów a część ułamkowa w postaci drugiego ciągu.

W systemie o R=2 (dwójkowym, binarnym) są dwie cyfry: 0 i 1, a kolejne pozycje z n-pozycji liczby odpowiadają kolejnym potęgom liczby 2. Np.: dla n= 4 ciąg 1 1 1 0 $_{(2)}$ zawiera liczbę:

$1*2^3+1*2^2+1*2^1+0*2^0 = 14_{(10)}$

Zakładając 1 bajt (tzn. n= 8) to oznaczenia i wagi bitów są następujące:

Nr bitu:	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	2^7	……..			2^3	2^2	2^1	2^0
Wagi bitów:	128	64	32	16	8	4	2	1

Kod o takich wagach jest nazywany naturalnym kodem binarnym – NKB.

Bit b_0 -jest bitem najmniej znaczącym -LSB *(least significant bit)*,
a bit najstarszy bit tzn. b_{n-1} - MSB *(most significant bit)*
Największa liczba zapisana przy pomocy **n** bitów to **Lmax= (2^n –1)**- stąd dla n=8 Lmax=255 – co odpowiada jedynce na wszystkich bitach bajtu.

Zamiana liczby dziesiętnej (całkowitej, dodatniej) na binarną - algorytm Hornera

Zamienić liczbę dziesiętną np. 108 na binarną.
Sposób rozwiązania (jeden z możliwych):
Należy dzielić całkowicie tą liczbę przez dwa i zapisywać resztę z dzielenia całkowitego. Otrzymane wartości reszt , zapisane w kolejności odwrotnej dają zapis binarny liczby (na końcu jest najbardziej znacząca cyfra):

Działanie	Wynik	Reszta
108:2	54	0
54:2	27	0
27:2	13	1
13:2	6	1
6:2	3	0
3:2	1	1
1:2	0	1

Wynik:
$1101100_{(2)}$

Sprawdzenie: 1101100(2)=1*26+ l*25 + 1*23 + 1*22 = 64+32+8+4 =108(10)

Podobnie można odczytać wartość dziesiętną binarnej liczby rzeczywistej (dodatniej) np.:

część całkowita ← część ułamkowa →

$$101{,}101 = 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 5{,}625$$

101 101 $_{(2)}$ *-inna forma zapisu liczb z częścią ułamkową*

Ciągi binarne - podstawowe operacje logiczne

W operacjach logicznych liczba binarna jest traktowana, jako zbiór pojedynczych cyfr binarnych.

Operacje na bitach	
Negacja (NOT)	!1 = 0, !0 = 1 *lub tak:* ~0 =1 ~1 = 0
Koniunkcja (&- AND)	0 & 0 = 0, 1 & 0 = 0, 0 & 1 = 0, 1 & 1 = 1
Alternatywa (I - OR)	0 | 0 = 0, 1 | 0 = 1, 0 | 1 = 1, 1 | 1 = 1
Różnica symetryczna (^ - XOR)	0 ^ 0 = 0, 1 ^ 0 = 1, 0 ^ 1 = 1, 1 ^ 1 = 0

Przykłady:

```
NOT 1010
    ||||
    0101
```

```
    1010
OR  1011
--------
    1011
```

```
     1010
AND  1011
---------
     1010
```

Jest różnica na najmłodszym bicie!!!

```
     1010
XOR  1011
---------
     0001
```

Przykład:

Rozpatrzmy warunki logiczne dla istnienia trójkąta np. :

K= a+b>c

L=b+c>a

M=a+c>b

T=K & L & M ***<-- Jeżeli T==1 to trójkąt istnieje***

Np.: dla a=2 b=3 c=6 → mamy: K=0 L=1 M=1 →T=0→ *trójkąt* *nie istnieje*

Np.: dla a=2 b=7 c=6 →mamy: K=1 L=1 M=1 →T=1 → *trójkąt istnieje*

Operacje arytmetyczne na ciągach binarnych

Dodawanie

Liczby dwójkowe dodajemy podobnie, jak dziesiętne. Gdy po dodaniu dwóch cyfr uzyskuje się wartość niemożliwą do zapisania pojedynczą cyfrą, zachodzi tzw. *przeniesienie*. Odejmujemy wtedy od wyniku **podstawę systemu**, a do **następnej pozycji dodajemy 1** (np. 9+8=17-10) =7→(1)7

W przypadku liczb dwójkowych przeniesienie wystąpi już wtedy, gdy wynik dodawania dwu cyfr jest większy od 1.

Reguły dodawania:

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 1\,0 \end{array}$$

przeniesienie

$$\text{Przykład} \qquad \begin{array}{r} 5 \\ +\ 7 \\ \hline 12 \end{array} \qquad \begin{array}{r} 101 \\ +\ 111 \\ \hline 1100 \end{array}$$

Reguły odejmowania:

pożyczka

$$\begin{array}{r} 0 \\ -\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ -\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ -\ 1 \\ \hline 0 \end{array} \qquad \begin{array}{r} (1)\,0 \\ -\ 1 \\ \hline 1 \end{array}$$

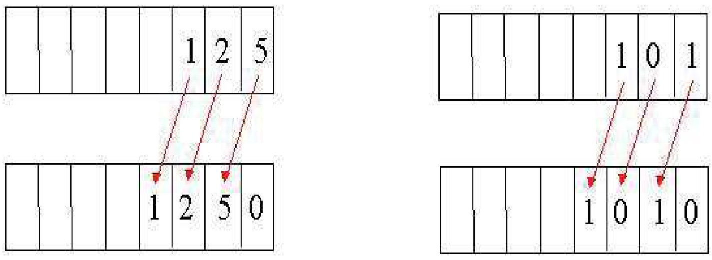
$$\text{Przykład} \qquad \begin{array}{r} 9 \\ -\ 5 \\ \hline 4 \end{array} \qquad \begin{array}{r} 1001 \\ -\ 0101 \\ \hline 0100 \end{array}$$

[Mnożenie przez 2](#) **w układzie binarnym - przesunięcie liczby o <u>jedną pozycję w lewo</u> i dopisanie zera z prawej strony liczby.**

[Dzielenie przez 2](#) **w układzie binarnym - przesunięcie liczby <u>o jedną pozycję w prawo</u> i odrzucenie ostatniej cyfry (jeśli liczba parzysta to tą cyfrą jest zero).**

<u>Przykład:</u>

Mnożenie przez 10 w układzie dziesiętnym i przez 2 w układzie dwójkowym.



Podobnie mnożenie i dzielenie w układzie dwójkowym przez 4 i inne potęgi dwójki – można realizować jako przesunięcie w lewo (dla mnożenia) lub w prawo (dla dzielenia) o odpowiednią liczbę pozycji.

System heksadecymalny

W systemie heksadecymalnym R=16 (szesnastkowym) mamy:
- **10 cyfr: (0, 1, 2,...,9)**
- **oraz sześć liter (A, B, C, D, E, F)**

Wartości 10 odpowiada A, 11→ B, 12 → C, 13 → D, 14 → E, 15 →F.

Kolejne pozycje liczby w tym systemie odpowiadają kolejnym potęgom liczby 16.

Np. liczba $5C_h = 5*16^1+12*16° = 80+12 = 92_d$

<u>Przykład:</u> Znaleźć notację heksadecymalną liczby dziesiętnej N = 107.
<u>Rozwiązanie:</u>

Należy liczbę dzielić całkowicie przez R=16 i zapisywać resztę z dzielenia całkowitego.
Reszty 10 i więcej zapisujemy jako cyfry A,B,C,D,E,F.

Działanie	Wynik	Reszta	
1708: 16	106	(12)	czyli C
106:16	6	(10)	czyli A
6:16	0		6

Wynik:
$1708_d = 6\ AC_h$

Sprawdzenie: $6\ AC_h = 6*16^2 + 10*16^1 + 12*16^0 = 1536 + 160 + 12 = 1708_{(10)}$

Przykład zamiany liczby binarnej na szesnastkową

Dana jest liczba naturalna n =**10010110011** zapisana w systemie binarnym. Znajdź jej notację w systemie szesnastkowym.

Rozwiązanie

Podstawą systemu jest R=16 (2^4 - **co wymaga 4 bitów**) a binarnego 2, należy więc **podzielić ciąg** bitów **na grupy 4 bitowe** i każdej przyporządkować cyfrę systemu hex (wg tabeli).

Cyfra heksadecymalna	Liczba
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Rozwiązanie:

Należy zamienić czteroelementowe ciągi cyfr z systemu binarnego na cyfry w systemie heksadecymalnym.

0100 1011 0011

$\downarrow \quad \downarrow \quad \downarrow$

4 B 3

W y n i k

Na liczbach hex można wykonywać typowe operacje np.:

	1B8	→ *440*
+	*C7*	→ *199*
	2 7 F	→ *639*

Zapis binarny ujemnych liczb całkowitych

Przyjęto, że na najstarszym bicie jest kodowany znak liczby. I tak jeśli jest tam „1” to oznacza liczbą ujemną. Wartość dziesiętna liczby ujemnej zależy jaki był zastosowany kod.
Taka sama liczba ujemna np. x=-23 ma inny ciąg binarny w ZM, U1 i U2!
Natomiast liczby dodatnie np. x=23 mają taki sam ciąg binarny w każdym kodzie - zapis w NKB.
Liczby ujemne są zapisywane w jednym z kodów:
- ZM (znak- moduł);
- U1 (Kod z uzupełnieniem do 1)
- U2 (Kod z uzupełnieniem do 2)

Kod ZM (znak moduł)

Zapisać liczbę –23 w kodzie ZM na n=8 bitach. Korzystając z definicji mamy:

$X_{ZM} = 2^{n-1} + |x|$ jeśli x < 0 - definicja dla zapisu liczby **x** ujemnej

$$
\begin{array}{llr}
2^{n-1} = 2^7 = 128 & \rightarrow & 1\ 0000000 \\
|-23| = & \rightarrow & +0\ 0010111 \\
\hline
 & & 1\ 0010111 \rightarrow -23_{ZM}
\end{array}
$$

Kod U1 (z uzupełnieniem do 1)

$X_{u1} = (2^n - 1) - |x|$ jeśli x < 0 - definicja dla zapisu **x** ujemnej

$$
\begin{array}{llr}
2^n - 1 = 2^8 - 1 = 255 & \rightarrow & 1\ 1111111 \\
|-23| = & \rightarrow & -\ 0\ 0010111 \\
\hline
 & & 1\ 1101000 \rightarrow -23_{U1}
\end{array}
$$

Warto zauważyć, że kod U1 liczby np.: x=-23 **można prosto otrzymać:**

a) zapisując moduł liczby ujemnej,
b) a następnie negując bit po bicie
c) otrzymujemy liczbę ujemną zapisaną w kodzie U1.

Np. dla x=-23 mamy w kodzie U1:

Moduł → **0 0010111**
NOT → **1 1101000** → **- 23$_{U1}$**

dla x=-23 w kodzie ZM:

Moduł → **0 0010111**
ZnM → **1 0010111**

Kod U2 (z uzupełnieniem do 2)

$X_{u2} = 2^n - |x|$ **jeśli x < 0** - definicja dla zapisu **x** ujemnej (3)

Poniżej posługując się zapisem w tabeli, pokazano zamianę x=-23 na kod U2 wg. wzoru 3.

1 etap

Zamieniamy na postać binarną składniki wzoru czyli liczby: 2^n (czyli 256) i 23.
(patrz wiersze 2 i 3 od góry tabeli)

2 etap.

Wykonujemy odejmowanie: $2^n - |x|$:

			b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0		
				..1	..1	..1	..1	..1	..1 ←	1		← Pożyczka	Dziesiętnie
2^n	→		1	0	0	0	0	0	0	0	0		← 256
- |-23|	→	-		0	0	0	1	0	1	1	1		← -23
-23_{u2}	→			1	1	1	0	1	0	0	1		← 233_{NKB}

Ostatni wiersz przedstawia zapis binarny liczby ujemnej x=-23 w kodzie U2. Kolor zielony to bit znaku.
Liczbę całkowitą **x<0** najprościej zapisać w kodzie U2 na n bitach wg algorytmu:

I. jeśli x ≥ 0 to liczbę x zapisujemy w NKB i Koniec,
II. jeśli x < 0 to:
    1. zapisujemy w NKB moduł liczby x,
    2. negujemy bit po bicie (uzupełnienie do U1),
    3. do najmniej znaczącego bitu tzn. b_0 dodajemy 1.

Przykład:

Moduł : → 0 0010111
Negacja: → 1 1101000 → dostajemy - 23_{U1}

Do wyniku: → 1 1101000
Dodajemy+1 + 1
Otrzymujemy: 1 1101001 → dostajemy - 23_{U2}

UWAGA:
Jeśli liczba jest zapisana na w kodzie U2 i ma na najstarszym bicie „1" to jego wagę można interpretować jako liczbę ujemną. Dla bajtu (gdzie n=8) będzie to liczba -128. Pozostałe bity bajtu zecowują swoje wagi : 64, 32, 16, 8, 4, 2, 1.
Tak więc otrzymany wyżej ciąg ma wartość dziesiętną:

x= -128 + 105 = -23

tabeli poniżej pokazano liczby charakterystyczne dla 8-bitowego U2 oraz dla porównania ich reprezentacje binarne w kodach: ZM i U1.

n= 8	**ZM**	**U1**	**U2**
-128	**-**	**-**	**1 0000000**
-127	**1 1111111**	**1 0000000**	**1 0000001**
-126	**1 1111110**	**1 0000001**	**1 0000010**
	…………..	**………**	
-2	**1 0000010**	**1 1111101**	**1 1111110**
-1	**1 0000001**	**1 1111110**	**1 1111111**
Zero	**0 0000000** **1 0000000**	**1 1111111** **0 0000000**	**0 0000000 jedna postać**
+1	**0 0000001**	**0 0000001**	**0 0000001**
+2	**0 0000010**	**0 0000010**	**0 0000010**
	…………..		
+126	**0 1111110**	**0 1111110**	**0 1111110**
+127	**0 1111111**	**0 1111111**	**0 1111111**