ARYTMETYKA KOMPUTERA

Systemy liczbowe o różnych podstawach

System dziesiętny

Cyfry: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Liczba 764.5 oznacza $7 * 10^2 + 6 * 10^1 + 4 * 10^0 + 5 * 10^{-1}$.

System ten jest wygodny dla człowieka a dla maszyny mniej. Reprezentacja cyfry dziesiętnej zajmuje cztery bity pamięci komputera:

cyfra	reprezentacja
0	**0000**
1	**0001**
2	**0010**
3	**0011**
4	**0100**
5	**0101**
6	**0110**
7	**0111**
8	**1000**
9	**1001**

W ten sposób marnujemy część pamięci ponieważ są kombinacje (np. 1101), które nie oznaczają żadnej cyfry dziesiętnej.

System dwójkowy

Cyfry: 0, 1
Liczba $(1010.1)_2$ oznacza $1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1}$.

System dwójkowy dobrze pasuje do maszyny gdyż reprezentacja jednej cyfry zajmuje dokładnie jeden bit. n-cyfrowa liczba (bez znaku) pamiętana jest w słowie n-bitowym. Natomiast dla człowieka jest on zbyt rozwlekły.

Konwersja zapisu dziesiętnego na dwójkowy

Istnieje prosty sposób konwersji zapisu dziesiętnego liczby na dwójkowy:
1. część całkowitą dzielimy przez 2 i bierzemy reszty,
2. część ułamkową mnożymy przez 2 i bierzemy części całkowite (aż do uzyskania żądanej precyzji).

Przykład. Dla liczby x = $(43.625)_{10}$ liczymy

43	
21	1
10	1
5	0
2	1
1	0
0	1

	625
1	250
0	500
1	000

i otrzymujemy

$$x = (101011.101)_2$$

W przypadku niektórych liczb reprezentowanych w systemie dziesiątkowym możemy uzyskać tylko przybliżone reprezentacje dwójkowe.

Na przykład dla liczby $y = (69.76)_{10}$ powyższa procedura się nie zakończy

69	
34	1
17	0
8	1
4	0
2	0
1	0
0	1

	76
1	52
1	04
0	08
0	16
0	32
0	64
1	28
...	...

i otrzymujemy wynik przybliżony

$$y \approx (1000101.1100001)_2.$$

System szesnastkowy

Cyfry: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A,B,C,D,E, F
Liczba $(E9B.F)_{16}$ oznacza $14 * 16^2 + 9 * 16^1 + 11 * 16^0 + 15 * 16^{-1}$.

System ten w zasadzie łączy dobre cechy systemu dwójkowego i dziesiętnego. Jest bardziej zwięzły od dwójkowego i lepiej wykorzystuje pamięć niż system dziesiątkowy. Reprezentacja cyfry szesnastkowej zajmuje cztery bity pamięci komputera.

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

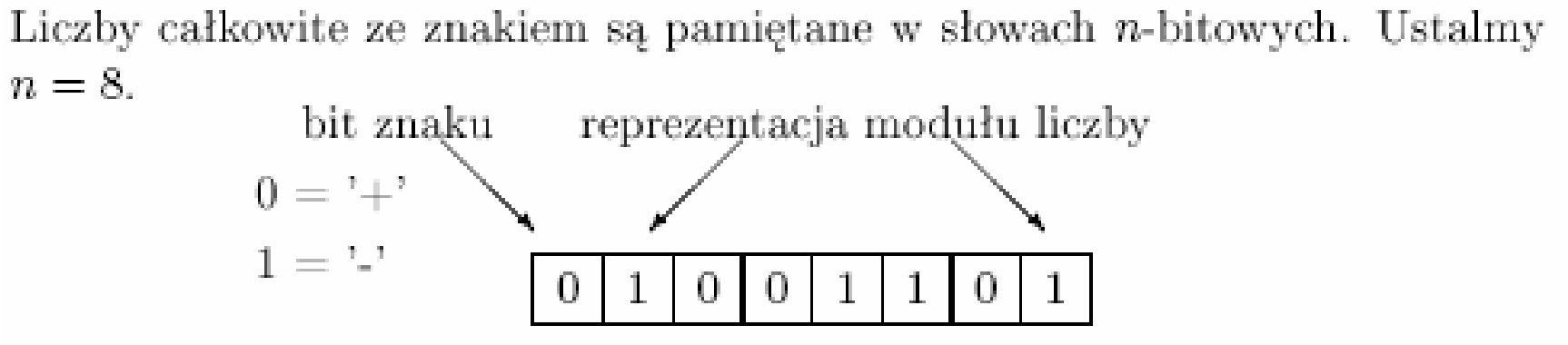
i każda kombinacja 4-bitowa odpowiada pewnej cyfrze szesnastkowej. Dlatego często system szesnastkowy jest stosowany do zapisu liczb dwójkowych.

Na przykład kody ASCII znaków są często podawane przy użyciu dwucyfrowych liczb szesnastkowych. Natomiast ludzkie przyzwyczajenie do operowania w systemie dziesiątkowym jest tak ugruntowane, że żaden inny system pozycyjny nie może mieć szerszego znaczenia w komunikacji między ludźmi.

Do zamiany postaci liczb całkowitych w zakresie od 0 do 255
można wykorzystać arkusz kalkulacyjny MS-Excel.
Należy tylko zainstalować i załadować dodatek ***Analysis ToolPak****.*

Do zamiany liczb służą następujące funkcje lub ich złożenia:
DEC2BIN()
BIN2DEC()
DEC2HEX()
HEX2DEC()
DEC2HEX(BIN2DEC())
DEC2BIN(HEX2DEC())

Reprezentacja stałopozycyjna liczb całkowitych



Nieujemna liczba całkowita jest pamiętana jako znak i moduł liczby zapisany w systemie dwójkowym. Powyższe słowo reprezentuje liczbę 77. Natomiast jeśli liczba jest ujemna to istnieje kilka sposobów jej reprezentacji:

1. **znak-moduł**: wygodny dla człowieka, ale przy operacjach arytmetycznych trzeba porównywać znaki i 0 ma dwie reprezentacje ('dodatnią' i 'ujemną').

2. **znak-uzupełnienie do 1**: ta reprezentacja jest mniej wygodna dla człowieka i w niej też 0 ma dwie reprezentacje. To, że 0 ma dwie reprezentacje nie jest tylko sprawą estetyki. Gdy jeden obiekt ma równe reprezentacje, sprawdzenie równości dwóch obiektów staje się niepotrzebnie skomplikowana procedurą.

3. **znak-uzupełnienie do 2**: ta reprezentacja jest jeszcze mniej wygodna dla człowieka ale w niej 0 (i każda inna reprezentowana liczba) ma tylko jedną reprezentację, ponadto operacje arytmetyczne są wykonywane w prosty sposób.

Ze względu na przytoczone powyżej zalety zajmiemy się bliżej sposobem reprezentacji liczb jako **znak-uzupełnienie do 2**. Ten jest w praktyce używany do reprezentacji liczb całkowitych w komputerach.

Uzupełnienie do 1 liczb całkowitych otrzymujemy negując wszystkie bity.

Dla

$$x = 10011000$$

uzupełnienie do 1 liczby x jest równe

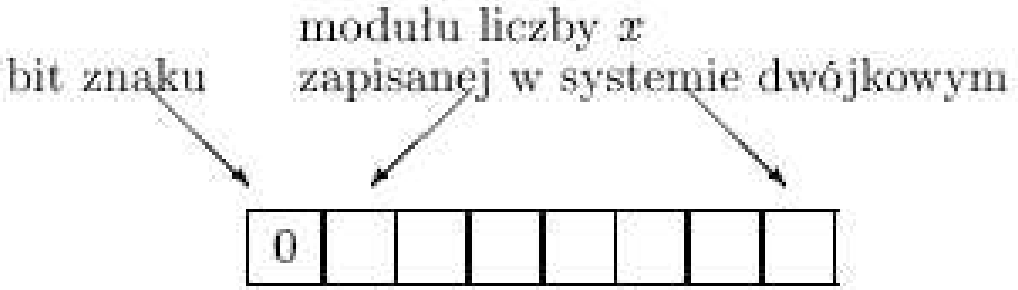
$$01100111$$

Uzupełnienie do 2 liczb całkowitych otrzymujemy negując wszystkie bity i dodając 1. Dla x jak wyżej uzupełnienie do 2 liczby x jest równe:

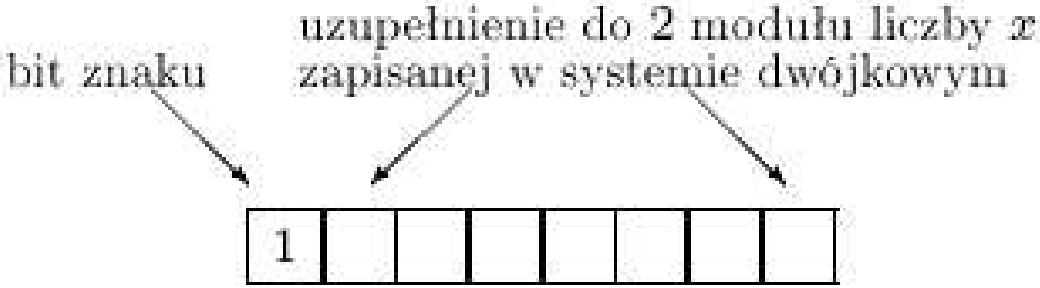
$$\begin{array}{rr} & 01100111 \\ + & 1 \\ \hline & 01101000 \end{array}$$

Reprezentacja liczb całkowitych w systemie znak-uzupełnienie do 2 ($n = 8$):

1. dla $0 \leq x \leq 127$



2. dla $-128 \leq x \leq -1$



Przykład. Słowo

0	1	0	1	0	0	0	0

reprezentuje liczbę dodatnią 80 natomiast słowo

1	1	0	1	0	0	0	0

reprezentuje liczbę ujemną $-(0110000)_2 = -48$.

Operacje arytmetyczne stałopozycyjne

Reprezentując liczby w systemie znak-uzupełnienie do 2, przy dodawaniu nie trzeba zwracać uwagi na znak liczby. Wyniki otrzymujemy poprawne i dokładne o ile argumenty i wartości mają swoje reprezentacje jako słowa n-bitowe w systemie znak-uzupełnienie do 2.

1. **Zmiana znaku** Uzupełniamy do 2 liczby (łącznie z bitem znaku):

$x = 26 \sim$ | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

$-x = -26 \sim$ | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

2. **Dodawanie** Dodajemy dwie liczby łącznie z bitem znaku i ewentualne przeniesienie pomijamy. Wynik reprezentuje sumę w systemie znak - uzupełnienie do 2:

$x = 24 \sim$ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

$y = -40 \sim$ | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |

$x + y = -16 \sim$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

3. **Odejmowanie** Dodajemy odjemną do liczby przeciwnej do odjemnika.

4. **Mnożenie i dzielenie** Nie tak prosto. Najlepiej mnożyć i dzielić moduły a potem ustalać znak. Jeśli mnożnik jest dodatni to można mnożyć jak zwykle.

Uwaga. Zakres liczb całkowitych reprezentowanych w komputerze jest zwykle dość mały a operacje są wykonywane dokładnie. Dla liczb rzeczywistych jest odwrotnie, zakres jest duży a operacje są wykonywane w sposób przybliżony.