Zapis binarny danych w pamięci komputera

Autorem poniższego opracowania jest dr Piotr A. Dybczyński z **Instytutu Obserwatorium Astronomiczne UAM** w Poznaniu.

Pozycyjny system zapisu liczb

Powszechnie stosowany system dziesiętny zapisu liczb jest tzw. systemem pozycyjnym. Oznacza to, że wartość (waga) każdej z użytych cyfr, zależy od jej pozycji w liczbie.

Idąc od prawej strony mamy w każdej liczbie wielocyfrowej jednostki, dziesiątki, setki, tysiące itd. Dziesiętny system pozycyjny przypisuje każdej z cyfr w liczbie wagę równą odpowiedniej potędze podstawy systemu, czyli dziesiątki.

I tak, cyfra stojąca na skrajnej, prawej pozycji ma wagę (jest przy określaniu wartości liczby mnożona przez) $\mathbf{10^0}$, czyli **1**, jest to więc liczba jednostek.

Stojąca obok niej z lewej cyfra, ma wagę $\mathbf{10^1}$, czyli **10**, jest to więc liczba dziesiątek.

Ogólnie możemy powiedzieć, że jeżeli jakaś liczba jest w systemie dziesiętnym zapisana ciągiem cyfr, które oznaczymy jako

$$a_n a_{n-1} a_{n-2} \ldots a_2 a_1 a_0$$

to jej wartość wyliczamy ze wzoru:

$$a_n \times 10^n + a_{n-1} \times 10^{n-1} + a_{n-2} \times 10^{n-2} + \ldots + a_2 \times 10^2 + a_1 \times 10^1 + a_0 \times 10^0$$

Dla prostoty opisu zajmujemy się tylko liczbami całkowitymi, choć liczby ułamkowe w zapisie dziesiętnym są konstruowane analogicznie, tylko z ujemnymi potęgami dziesiątki w wagach cyfr po przecinku.

Liczby ujemne w tym zapisie oznaczamy specjalnym symbolem **'-' (minus)**, stawianym przed całą liczbą.

System binarny jest skonstruowany dokładnie na tej samej zasadzie!

Różnica jest tylko jedna: podstawą systemu jest liczba **2**, a w zapisie liczb występują tylko dwie cyfry, **0** i **1**. Jest to również system pozycyjny, a więc o wadze każdej cyfry decyduje jej pozycja w liczbie binarnej (dwójkowej), liczona też od prawej strony.

Zamiast jednostek, dziesiątek, setek i tysięcy mamy tu: jednostki, dwójki, czwórki, ósemki itd. Np. liczba dwójkowa:

$$10010101001$$

ma wartość:

$$1\times2^{10} + 0\times2^{9} + 0\times2^{8} + 1\times2^{7} + 0\times2^{6} + 1\times2^{5} + 0\times2^{4} + 1\times2^{3} + 0\times2^{2} + 0\times2^{1} + 1\times2^{0}$$

czyli w tym przypadku **1024+128+32+8+1 = 1193** w zapisie dziesiętnym. Liczba **-1193** będzie miała postać **-10010101001** .

```
   10010100101
 +    1000111
 ------------
   10011101100
```

```
     10010100101
 x       1000111
 ----------------
     10010100101
    10010100101
   10010100101
 10010100101
 ----------------
 101001001110000 11
```

Przykład pisemnego
dodawania i mnożenia
liczb w zapisie dwójkowym.

Działania na liczbach dwójkowych

Pisemne działania na liczbach w zapisie binarnym przeprowadzamy według tych samych zasad co na liczbach w zapisie dziesiętnym, pamiętając jedynie, że podstawą jest teraz dwa a nie dziesięć. Przykłady zapisów takich działań pokazane są na rysunku obok.

System binarny w komputerach

Liczby całkowite zapisywane są w pamięci komputerów w systemie dwójkowym opisanym wyżej, są jednak pewne specyficzne różnice. System binarny wybrano, bo stosunkowo najprościej było użyć dwustanowych elementów elektronicznych do zapisania dwóch różnych cyfr. Jedną cyfrę dwójkową w pamięci komputera nazywamy **bitem**. Już bardzo dawno przyjęto, że najmniejszą, bezpośrednio adresowalną porcją pamięci, będzie oktet (czyli osiem) bitów, nazwany **bajtem**.

Dla skrócenia (i ułatwienia) zapisu liczb binarnych występujących w komputerach stosuje się często system szesnastkowy. W praktyce oznacza to podzielenie każdego bajtu na dwie czwórki bitów i przypisanie każdej z nich cyfry systemu szesnastkowego:

Dziesiętnie	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binarnie	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Szestnastkowo	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Znaki (teksty) w systemie binarnym w komputerach

- **Kod ASCII** (American Standard Code for Information Interchange) - 7 bitowy kod, przyporządkowujący liczby z zakresu 0-127 literom alfabetu angielskiego, cyfrom, znakom przestankowym i niektórym innym symbolom oraz kodom sterującym. Przykładowo mała litera "a" jest kodowana liczbą 97 (binarnie **1100001**), duża litera "A" liczbą 65 (binarnie **1000001**, a znak spacji - 32 (binarnie **0100000**).

Litery, cyfry oraz inne znaki drukowane tworzą zbiór znaków ASCII. Jest to 95 znaków drukowalnych o kodach 32-126. Pozostałe 33 kody (0-31 i 127) to tzw. kody sterujące służące do sterowania urządzeniem, np. drukarką czy terminalem.

- **Standardy ISO-8859** Ponieważ na komputerach informacje kodujemy najczęściej w porcjach ośmiobitowych (w bajtach), dość szybko powstały rozszerzenia kodu **ASCII**. Wszystkie odmiany ISO-8859 mają znaki 0-127 (hex 80-9F) takie same jak ASCII, zaś pozycjom 128-159 (hex 80-9F) przypisane są dodatkowe kody sterujące, w praktyce nieużywane. Natomiast kody 160 - 255 (hex A0 - FF) zawierają zależnie od wariantu znaki specyficzne dla danej grupy języków. I tak na przykład tak zwane **polskie literki: ąćęłńóśźżĄĆĘŁŃÓŚŹŻ** występują w wariancie **ISO-8859-2**.
- **Unikod** (zestaw znaków) i jego różne kodowania. Wraz ze wzrostem pojemności (i spadkiem cen) komputerowych pamięci coraz popularniejsza staje się idea stosowania jednego, uniwersalnego systemu kodowania wszystkich potrzebnych znaków. Znaki zestawu UNICODE są kodowane albo w odmianie o stałej długości kodów (dowolny znak zajmuje zawsze 4 bajty czyli 32 bity - stąd nazwa: **UTF-32**) lub w oszczędniejszych wersjach, z których najpopularniejszą obecnie jest **UTF-8**. **Niektóre symbole astronomiczne w Unikodzie**

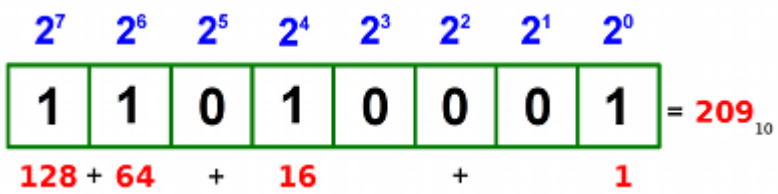
Liczby w systemie binarnym w komputerach

Liczby całkowite

Pierwsza zauważalna różnica to fakt, że liczby binarne w pamięci komputera mają liczbę cyfr będącą całkowitą wielokrotnością ośmiu. Realizowane jest to poprzez dopisywanie nieznaczących zer po lewej stronie, tak by "dopełnić" skrajny, lewy bajt. Mówimy wręcz w żargonie komputerowym o liczbach jedno-, dwu- czy np. cztero-bajtowych. Takie sformułowanie od razu określa zakres możliwych do zapisania liczb poprzez ustalenie maksymalnej ilości cyfr dwójkowych.

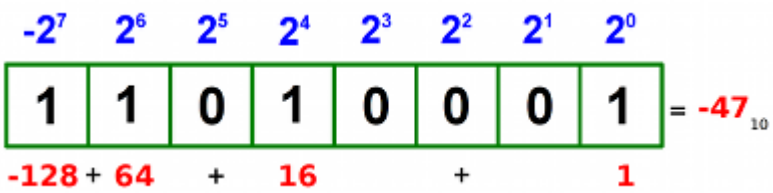
I tak:

- w jednym bajcie możemy zapisać liczby z zakresu **od 0 do 255**
- w dwóch bajtach **od 0 do 65535**
- w czterech **od 0 do 4294967295**
- w ośmiu: **0 do 18446744073709551615**



Bajt interpretowany jako liczba bez znaku.

No dobrze, a co ze znakiem? **I tu pojawia się druga, znacznie ważniejsza różnica.** Dla oszczędności miejsca w pamięci umówiono się, że zamiast stosować dodatkowy sposób sygnalizowania, że liczba jest ujemna, znak będzie kodowany na jednym z bitów. Aby jednak nie tracić tego bitu bezpowrotnie tylko na kodowanie znaku, przyjęto, że to **procesor musi wiedzieć "z innych źródeł"**, czy w danym kawałku pamięci zapisaliśmy liczbę bez znaku (czyli na pewno dodatnią) lub ze znakiem (czyli dodatnią lub ujemną). Z samego zapisu binarnego nie sposób zgadnąć, o który wariant kodowania chodzi. Te "inne źródła" to najczęściej interpretacja domyślna w danej implementacji lub (np. w języku C) jawne deklarowanie wszystkich zmiennych jako "ze znakiem" lub "bez znaku".

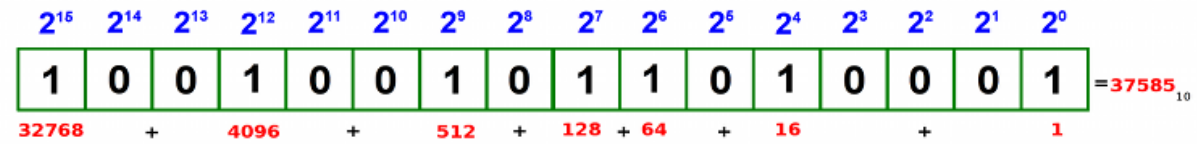


Bajt o tej samej zawartości, teraz interpretowany jako liczba ze znakiem.

Najpowszechniejsza dziś umowa, tzw. kodowanie z uzupełnieniem do 2 ("Two's complement") mówi tak:

- jeśli ma być zapisana liczba bez znaku (unsigned) to stosujemy normalny zapis dwójkowy w ramach limitu narzuconego przez ilość bajtów, czyli w zakresach podanych wyżej.
- jeśli natomiast ma być zakodowana liczba ze znakiem to umawiamy się że **waga najstarszego bitu zostaje pomnożona przez (-1)**. Dla liczb jedno-bajtowych oznacza to wagę -128 (-1×2^7), dla dwu-bajtowych -32768 (-1×2^{15}) itd.

  Powoduje to w praktyce, dla liczb ze znakiem, przesunięcie podanych wyżej zakresów tak, by zero wypadło po środku:
  - w jednym bajcie możemy zapisać liczby ze znakiem z zakresu **od -128 do +127**
  - w dwóch bajtach **od -32768 do +32767**
  - w czterech **od - 2147483648 do +2147483647**
  - w ośmiu: **od -9223372036854775808 do +9223372036854775807**



Dwa bajty interpretowane jako liczba bez znaku.

-2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
1	0	0	1	0	0	1	0	1	1	0	1	0	0	0	1	$= -27951_{10}$
-32768	+		4096	+		512	+	128	+ 64	+	16		+		1	

Dwa bajty o tej samej zawartości, teraz interpretowane jako liczba ze znakiem.

Jeszcze raz, dużymi literami: TO PROCESOR MA WIEDZIEĆ, JAK ZINTERPRETOWAĆ CIĄG BITÓW. Jeśli zastanie w jednym bajcie **11111111**, to gdy ma to być liczba bez znaku, zapis ten zostanie zinterpretowany jako (dziesiętnie) **255**, jeśli natomiast ma to być liczba ze znakiem, to wynosi ona **-1** !! Jak widać różnica jest zasadnicza!

Taki sposób kodowania liczb ze znakiem ma jeszcze inne ciekawe i korzystne własności.

Liczby dwójkowe są bardzo często skrótowo zapisywane w **systemie szesnastkowym**.

Liczby zmiennoprzecinkowe

są również kodowane w pamięci komputerów za pomocą bitów. Powszechnie stosowany jest sposób zapisu opisany w **standardzie IEEE-754**.

Ponieważ nie będą nam potrzebne detale tego zapisu, ograniczymy się tu do podania jego podstawowych cech. Liczba zmiennoprzecinkowa pojedynczej precyzji (typ **float** w języku C) jest zapisywana w czterech bajtach, czyli 32 bitach.

Wygląda ona następująco (rys.):

ZWWWWWWWWMMMMMMMMMMMMMMMMMMMMMMMM

Z jest bitem znaku całej liczby, zero oznacza liczbę dodatnią, jedynka - ujemną.

WWWWWWWW to osiem bitów wykładnika. Jest to liczba binarna bez znaku, od której odejmujemy 127_{10}, uzyskując w ten sposób wykładniki z zakresu (dziesietnie) od -127 do +128.

MMMMMMMMMMMMMMMMMMMMMMMM to 24 bity mantysy, interpretowanej jako część ułamkowa zmiennoprzecinkowej liczby binarnej: 1.MMMMMMMMMMMMMMMMMMMMMMMM . Daje to w zapisie dziesiętnym ok. 7 cyfr znaczących.

Ostatecznie wartość liczby to: znak 1.mantysa razy $2^{\text{wykładnik}}$.

Tak określony sposób kodowania pozwala zapisać liczby zmiennoprzecinkowe pojedynczej precyzji z zakresu od $1.17549435 \times 10^{-38}$ do $3.40282347 \times 10^{+38}$.

Liczba zmiennoprzecinkowa podwójnej precyzji (typ **double** w języku C) jest zapisywana w 8 bajtach, czyli 64 bitach. Budowa kodu jest podobna, jedynie wydłużają się wykładnik i mantysa. Wykładnik zajmuje 11 bitów, co pozwala zakodować wartości od -1023 do +1024. Część ułamkowa mantysy zajmuje 52 bity, dając w zapisie dziesiętnym 15-16 cyfr znaczących.

Tak określony sposób kodowania pozwala zapisać liczby zmiennoprzecinkowe podwójnej precyzji z zakresu od $2.2250738585072014 \times 10^{-308}$ do $1.7976931348623157 \times 10^{+308}$.

Przykład

$(10101100111)_2 = 1\cdot 2^0 + 1\cdot 2^1 + 1\cdot 2^2 + 0\cdot 2^3 + 0\cdot 2^4 + 1\cdot 2^5 + 1\cdot 2^6 + 0\cdot 2^7 + 1\cdot 2^8 + 0\cdot 2^9 +$
$+ 1\cdot 2^{10} = 1 + 2 + 4 + 32 + 64 + 256 + 1024 = (1383)_{10}$

Ćwiczenie 2.2.

1. Przelicz na system dwójkowy liczby:

a) $(248)_{10} =$
b) $(385)_{10} =$
c) $(724)_{10} =$
d) $(927)_{10} =$
e) $(1\ 342)_{10} =$
f) $(1\ 289)_{10} =$

2. Przelicz na system dziesiętny liczby:

a) $(10111010)_2 =$
b) $(11101001)_2 =$
c) $(10010111)_2 =$
d) $(101110101)_2 =$
e) $(110110111)_2 =$
f) $(1000111101)_2 =$

7 : 2	reszta 1
3 : 2	reszta 1
1 : 2	reszta 1

111

12 : 2	reszta 0
6 : 2	reszta 0
3 : 2	reszta 1
1 : 2	reszta 1

1100

1999 : 2	reszta 1
999 : 2	reszta 1
499 : 2	reszta 1
249 : 2	reszta 1
124 : 2	reszta 0
62 : 2	reszta 0
31 : 2	reszta 1
15 : 2	reszta 1
7 : 2	reszta 1
3 : 2	reszta 1
1 : 2	reszta 1

11111001111

Dwójkowy system liczbowy

127 ÷ 2 = 63 reszty **1**

63 ÷ 2 = 31 reszty **1**

31 ÷ 2 = 15 reszty **1**

15 ÷ 2 = 7 reszty **1**

7 ÷ 2 = 3 reszty **1**

3 ÷ 2 = 1 reszty **1**

1 ÷ 2 = 0 reszty **1**

$127_{10} = (1111111)_2$

19 ÷ 2 = 9 reszty **1**

9 ÷ 2 = 4 reszty **1**

4 ÷ 2 = 2 reszty **0**

2 ÷ 2 = 1 reszty **0**

1 ÷ 2 = 0 reszty **1**

$19_{10} = (10011)_2$

```
#include<iostream>
using namespace std;

int maks, d=0, p=1, tab[16];
char tablica[16];
main()
{
        cout<<("wprowadz liczbe binarna: \n");
        cin>>("%16s",tablica);                        /

        for (maks=0;maks<16;maks++)
                if (tablica[maks]=='\0')
                        break;

        for (int a=maks-1;a>=0;a--)                  //
        {
                tab[a]=tablica[a]-48;
        }
        for(maks;maks>0;maks--)                      //
        {
                if (tab[maks-1]==0 or tab[maks-1]==1)
                {
                        d+=(tab[maks-1])*p;
                        p=p*2;
                }
                else
                {
                cout<<("\n zla liczba");
                break;
                }
        }
        cout<<("\n %d",d);
        return 0;
}
```

5 4 3 2 1 0

$$101010_{(2)} = 0*2^0+1*2^1+0*2^2+1*2^3+0*2^4+1*2^5 = 0+2+0+8+0+32 = 42_{(10)}$$

1 0 1 0 1 0

42	0	42:2=21(r=0)
21	1	21:2=10 (r=1)
10	0	10:2=5 (r=1)
5	1	5:2=2 (r=1)
2	0	2:2=1 (r=0)
1	1	1:2=0 (r=1)
0		

(zamienioną liczbę w postaci binarnej odczytujemy od dołu)

$$42_{(10)} = 101010_{(2)}$$

P-PROGRAMOWANIE.PL