Sisteme de numeraţie

F.Boian, Bazele matematice ale calculatoarelor, UBB Cluj-Napoca, 2002

Sistem de numeraţie - totalitatea regulilor folosite pentru scrierea numerelor cu ajutorul unor simboluri (cifre).

1. **Sistemul de numeraţie roman**
   - sistem aditiv

Cifre:

I	V	X	L	C	D	M
1	5	10	50	100	500	1000

Reguli:

a) mai multe cifre de aceeaşi valoare, scrise consecutiv, reprezintă suma acestor cifre:
XX=20 MMM=3000

b) o pereche de cifre diferite, cu cifra mai mare aflată în faţa cifrei mai mici, reprezintă suma acestor cifre:
XIII=10+3=13 VII=5+2=7 DX=500+10=510

c) o pereche de cifre diferite, cu cifra mai mică în faţa cifrei mai mari, reprezintă diferenţa acestor cifre:
IV=5-1=4 IX=10-1=9 IL=50-1=49

d) pentru scrierea unor numere mai mari se foloseşte o bară orizontală deasupra simbolurilor, bară care indică înmulţirea valorii corespunzătoare simbolului cu 1000
$\overline{V}$ =5000

Dezavantaje:

- scriere greoaie
- reprezentări multiple ale numerelor (ex: 490=CDXC=XD)
- numere lungi
- operaţii lente
- nu există simbol pentru valoarea zero

Exemple

1989=MCMLXXXIX 2010=M M X

Palatul Josika (Casa cu picioare) este înălţat pe locul fostei reşedinţe clujene a principilor Transilvaniei. Clădirea a devenit reşedinţa lui Anton Josika, comite al Clujului, la mijlocul secolului al XVIII-lea. Clădirea în stil neoclasicist a căpătat înfăţişarea de astăzi în anul 1828, când a fost refăcută de Josika Janos, guvernator al Transilvaniei.
Elementul caracteristic în faţadă este porticul sobru cu coloanele dorice. Atica poartă inscripţia MDCCCXXVIII (1828), anul renovării clădirii.

2. Sistemul de numeraţie arab

- sistem poziţional

Aportul unei cifre în stabilirea valorii unui număr depinde de valoarea cifrei şi de poziţia ocupată în şirul de cifre folosit.

Regulă:

a) Fiecare grup de 10 elemente (unităţi, zeci, sute, etc.) formează o grupă de rang superior (zece, sută, mie, etc.)

Sistemul de numeraţie cu baza 10 (zecimal)
Cifre: 0 1 2 3 4 5 6 7 8 9
Sistemul de numeraţie octal - baza 8
cifre: 0 1 2 3 4 5 6 7
Sistemul de numeraţie binar - baza 2
cifre: 0 1
Sistemul de numeraţie hexazecimal - baza 16
cifre: 0 1 2 3 4 5 6 7 8 9 A (=10)
B(=11) C(=12) D(=13) E(=14) F(=15)

Reprezentarea numerelor naturale într-o bază oarecare b

Orice număr natural x poate fi scris sub forma:

$$x=c_0+c_1b+c_2b^2+...+c_{n-1}b^{n-1}+c_nb^n$$

unde $c_i<b$, i=0,1...,n se numesc cifrele numărului în baza b.

Exemple:

Numărul $(\overline{xyz})_b = z\cdot b^0 + y\cdot b^1 + x\cdot b^2$

$(271)_{10} = 1+7\cdot 10+2\cdot 100$

Conversia dintr-o bază oarecare b în baza 10

$(1101)_2=1\cdot 2^0+0\cdot 2+1\cdot 2^2+1\cdot 2^3=13$
$(A3C)_{16}=12\cdot 16^0+3\cdot 16+10\cdot 16^2=2620$

Pentru transformarea unui număr dintr-o bază oarecare b în baza 10 se înmulţeşte ultima cifră a numărului cu baza la puterea 0, penultima cifră se înmulţeşte cu baza la puterea a 1-a, antepenultima cu baza la puterea a 2-a, etc. şi se adună rezultatele acestor înmulţiri.

Pentru a face astfel de transformări trebuie cunoscute cifrele numărului care se transformă. Mai jos este dat ca exemplu un program care determină numărul de cifre ale unui număr natural în baza 10 şi valorile cifrelor acestui număr.
Cifrele numărului sunt resturile împărţirii întregi a numărului la baza sistemului de numeraţie în care este scris. După fiecare împărţire, noul număr care se împarte este câtul împărţirii precedente. Impărţirile continuă până când câtul devine 0.

Program exemplu: determinarea numărului de cifre ale unui număr natural

```
//Determina numarul de cifre ale unui numar natural
//Numarul trebuie sa fie <=32767  (de tip int)
//Cifrele numarului sunt stocate ca elemente ale sirului sirc
#include <stdio.h>
#include <conio.h>

void main()
{
  int c,r,nr,N,j,sirc[100];
```

```
  printf("\nProgramul determina numarul de cifre ale unui numar natural\n");
  printf("\nNumarul trebuie sa fie <=32767  (de tip int)\n\n\a");
  do
    {
      printf("\nIntroduceti un nr pozitiv: ");
      scanf("%d",&nr);
    }
  while(nr<0);

  j=0;
  do
      {
        c=nr/10;
        r=nr-c*10;
        sirc[j]=r;
        nr=c;
        j++;
      }
   while(nr>0);

  N=j-1;
  printf("\nNumarul are %d cifre. Sirul format din cifrele sale este\n\t",j);
  for(j=N;j>=0;j--)
    printf("\n%d",sirc[j]);
}
```

Pentru determinarea cifrelor zecimale unui număr real se procedează în felul următor:
Fie *nr* numărul real dat.

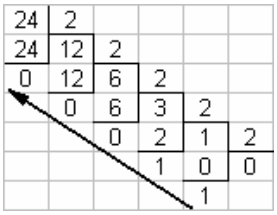
1. se determină prima cifră zecimală a numărului ca şi
   cif=[10*(nr-floor(nr))] //floor returneaza cel mai mare întreg <= *nr*
   //[] – semnifica partea întreaga a numărului
2. nr=nr*10
3. dacă nr≠floor(nr) continuă la 1
4. stop

Program exemplu: determinarea numărului de cifre zecimale ale unui număr real

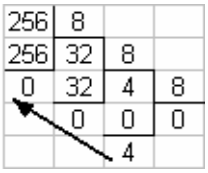
```
//Determina numarul de cifre zecimale ale unui numar real
//si construieste un sir cu aceste cifre

#include <iostream.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>

void main()
{
  double nr;
  long int j,N,sir[100],cif;

printf("\nProgramul determina numarul de cifre zecimale ale unui numar real\n");
printf("si construieste un sir avand ca elemente cifrele zecimale ale numarului
dat\n");

do
  {
      printf("\nIntroduceti un numar real:\t");
//    fflush(stdin);
  }
while(scanf("%lf",&nr)!=1);

j=0; // contor pentru sirul care va contine cifrele zecimale ale numarului real
```

```
while(nr!=floorl(nr))
//sau while((nr-floorl(nr))>1e-6)
{
  //determina cifra actuala
  cif=(long int) (10*(nr-floorl(nr)));
  sir[j]=cif;
  nr=nr*10.0;
  j++;
}
N=j;
printf("\nNumarul are %d cifre zecimale, iar sirul format din cifrele sale
este:\n",N);

for(j=0;j<N;j++)
  cout<< sir[j];
getch();
}
```

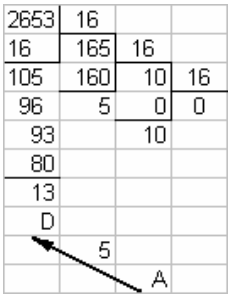
Conversia din baza 10 în alte baze



$(24)_{10} = (11000)_2$



$(256)_{10} = (400)_8$



$(2653)_{10} = (A5D)_{16}$

Pentru transformarea unui număr natural din baza 10 într-o bază oarecare b se împarte numărul respectiv la bază, obţinându-se un cât şi un rest mai mic decât b. Apoi se împarte câtul obţinut la b obţinându-se un nou cât şi un nou rest mai mic decât b. Împărţirea continuă până când se obţine câtul 0. Cifrele numărului în baza b sunt resturile împărţirilor, citite de la ultimul rest către primul.
Program exemplu: tipărirea unui număr natural în baza 10 introdus de la tastatură, în bazele 10, 8 şi 16, folosind specificatorii de tip %d, %o şi %x

```
//Tipareste un numar natural introdus de la tastatura
//in bazele 10, 8 si 16
#include <iostream.h>
#include <stdio.h>

void main()
{
  int n;

  cout << "Introduceti un nr. natural n (in baza 10) :  ";
  cin >> n;
  cout << "Baza 10: " << n;
  printf("\nBaza  8: %o", n);
  printf("\nBaza 16: %X", n);
}
```

Program exemplu: transformarea unui număr din zecimal în binar

```
//Transforma un numar natural din zecimal in binar

#include <stdio.h>
#include <conio.h>
```

```
void main()
{
  int c,r,nr,i,j,bin[100];
  printf("\nProgramul face transformarea in binar a unui numar natural\n");
  printf("\nNumarul trebuie sa fie <=32767  (de tip int)\n\n\a");
  do
      {
        printf("\nIntroduceti un nr pozitiv:   ");
        scanf("%d",&nr);
      }
      while(nr<0);
  j=0;

  do
    {
      c=nr/2;
      r=nr-c*2;
      bin[j]=r;
      j++;
      nr=c;
    }
    while(nr>0);

  printf("\nNumarul transformat in binar este:\t");
  for(i=j-1;i>=0;i--)
      printf("%d",bin[i]);
}
```

Conversia unui număr real între două baze de numeraţie

2 paşi

1. conversia părţii întregi prin metoda împărţirilor succesive
   - se fac împărţiri până când câtul devine zero, iar numărul se obţine din şirul resturilor scris în ordine inversă
2. conversia părţii fracţionare prin metoda înmulţirilor succesive
   - se fac înmulţiri până când:
     a) - partea de după virgulă devine zero
     b) - se ajunge la un număr prestabilit de cifre
     c) - se ajunge la o periodicitate
   - cifrele părţii fracţionare sunt cifrele care, în urma înmulţirilor succesive, ajung în faţa punctului zecimal

Partea fracţionară a unui număr scris într-o bază b se poate scrie sub forma:

$$c_{-1}/b + c_{-2}/b^2 + c_{-3}/b^3 + ...$$

unde c_{-1} este cifra aflată imediat după virgulă, c_{-2} este a doua cifră a părţii zecimale, etc.

Exemplu:
Fie numărul 0.369140625 în baza 10 care va fi convertit în baza 8. Vom înmulţi mereu părţile fracţionare cu 8

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

S-a îndeplinit criteriul a), obţinându-se zero după virgulă
Aşadar: $(0.369140625)_{10} = (0.275)_8$

Verificare: $(0.275)_8 = \frac{2}{8} + \frac{7}{8^2} + \frac{5}{8^3} = \frac{128+56+5}{512} = \frac{189}{512} = 0.369140625$

Temă: Transformaţi numărul 0.369140625 în baza 2.

Fie numărul 0.416382 pe care îl vom converti tot în baza 8, reţinând numai primele 3 cifre:

0	4	1	6	3	8	2	x	8
3	3	3	1	0	5	6		
2	6	4	8	4	4	8		
5	1	8	7	5	8	4		
.	.	.	.	.	.	.		

Aşadar: $(0.416382)_{10} = (0.325...)_8$
Convertind acum numărul $(0.325)_8$ în baza 10 obţinem:

$$(0.325)_8 = \frac{3}{8} + \frac{2}{8^2} + \frac{5}{8^3} = \frac{192+16+5}{512} = \frac{213}{512} = 0.416015625 \neq 0.416382$$

adică am obţinut o eroare de trunchiere. Astfel de erori apar frecvent în cazul conversiilor duble, atunci când se reţine un număr prestabilit de cifre semnificative.

Temă: Transformaţi numărul 0.3741 în baza 2, reţinând primele 6 cifre binare apoi faceţi transformarea inversă. Care este diferenţa dintre numărul iniţial şi cel obţinut prin transformarea inversă?

Algoritm pentru conversia părţii fracţionare dintr-o bază p în altă bază q, prin metoda înmulţirilor succesive:
Fie f partea fracţionară. După obţinerea cifrei de rang -1 cu relaţia $c_{-1}=[q \cdot f]$ ([ ] semnifică partea întreagă) în baza b se efectuează din nou înmulţirea înlocuind pe f cu gf, etc.

1. citeşte f în baza p
2. i=0;
3. dacă f=0 sau s-au obţinut suficiente cifre continuă la pasul 8
4. c_i=int(f·q) // partea întreagă a lui f·q
5. f=partea fractionară a lui f·q adică f = f·q-[f·q]
6. i++
7. continuă la pasul 3
8. n=i
9. scrie c_i, i=0, 1, ..., n-1
10. stop

Program exemplu: Conversia părţii fracţionare a unui număr scris în baza 10 într-o bază oarecare q

```
//Programul converteste partea fractionara a unui numar scris in baza 10
//intr-o alta baza q. Numarul se va introduce sub forma 0.xxxxxxx...

#include <iostream.h>
#include <stdio.h>
#include <math.h>

void main()
{
  double f;
  int i,n, c[20],q,nmax=15,cond;

  cout << "Introduceti partea fractionara a numarului, sub forma: 0.xxxxxxx...:
";
  cin >> f;
```

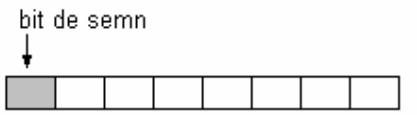
```
  cout << "Noua baza: ";
  cin>> q;
  i=0;
  do
  {
   c[i]=floor(f*q);
   f=f*q-floor(f*q);
   i++;
    //Stabilirea conditiei de incheiere: fie f=0, fie s-a
    // ajuns la numarul de cifre impus
   if(f==0.0)
      cond=1;
   else
      if(i==nmax)
        cond=1;
      else cond=0;
  }
  while(!cond);
  n=i;
  cout<<"Cifrele partii fractionare in baza " << q <<  " sunt: ";
  for(i=0;i<n;i++)
       cout << c[i];
}
```

Reprezentarea numerelor
Reprezentarea numerelor întregi cu semn în complement faţă de 2

Definiţii

1. Bit - unitatea de informaţie
2. Byte (octet) - unitatea de adresare (8 biţi)
3. Locaţie de memorie - unitatea de reprezentare a unei date, formată din unul sau mai mulţi octeţi
4. Word – număr de octeţi prelucraţi simultan de către procesor (numerele reale se reprezintă de obicei pe un cuvânt) = lăţimea de bandă

În memoria computerelor, numerele sunt reprezentate ca şi numere binare, pe un anumit număr (finit) de biţi. Valorile care pot fi reprezentate depind de numărul de biţi folosiţi pentru respectiva reprezentare.
Spre exemplu, pe 2 biţi poate fi reprezentată valoarea maximă 3 = $(11)_2$, iar pe 8 biţi poate fi reprezentată valoarea maximă 255=$(11111111)_2$.
Dacă trebuie reprezentate numere întregi cu semn, atunci un bit din numărul total de biţi ai reprezentării va fi folosit pentru semnul numărului. Bitul de semn va fi bitul de rang maxim (cel mai din stânga):



In reprezentarea în complement faţă de 2, un număr pozitiv se reprezintă pe cei n biţi, cu bitul de semn 0. Valoarea maximă reprezentabilă pe n biţi va fi:

pe patru biţi: | **0** | **1** | **1** | **1** | $(111)_2 = 7=2^3-1$

pe opt biţi: | **0** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | $(1111111)_2 = 127=2^7-1$

Aşadar, pe n biţi, valoarea maximă reprezentabilă este: $2^{n-1}-1$

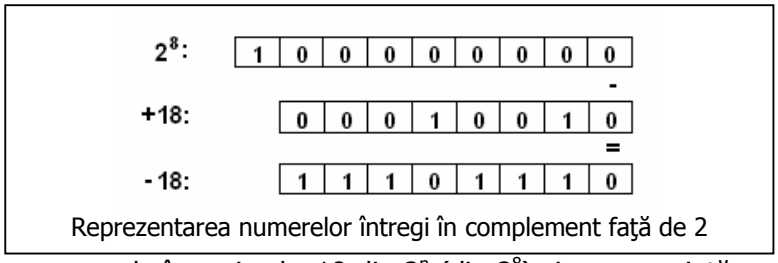
In reprezentarea în complement faţă de 2, numerele negative se obţin scăzând în binar, numărul pozitiv din 2^n. Bitul de semn pentru numerele negative va fi 1.

Aşadar, în acest cod, dacă X este pozitiv se reprezintă ca atare pe n-1 biţi, iar dacă este negativ, se reprezintă valoarea 2^n-|X|.

Exemplu:
Numărul +18, reprezentat pe 8 biţi este:

0	**0**	**0**	**1**	**0**	**0**	**1**	**0**

Numărul -18 se obţine scăzând din 2^n în binar valoarea +18 în binar (vezi figura de mai jos):



Reprezentarea numerelor întregi în complement faţă de 2

sau acelaşi lucru, se scade în zecimal +18 din 2^n (din 2^8) şi se reprezintă rezultatul scăderii în binar:

$$2^8\text{-}18=256\text{-}18=238$$
$$(238)_{10}=(11101110)_2$$

O metodă mai rapidă de deducere a reprezentării numerelor întregi negative în complement faţă de 2 pe n biţi rezultă din discuţia precedentă şi este dată de următoarea *regulă:*
Pentru a reprezenta în complement faţă de 2 un număr întreg negativ se reprezintă modulul său după care, începând de la bitul de ordin zero spre stânga toţi biţii 0 şi primul bit 1 se păstrează şi toţi ceilalţi îşi inversează valoarea (0->1 şi 1->0).

Acest lucru este echivalent cu faptul că valoarea oricărui număr negativ, cu excepţia celui mai mic se obţine pornind de la numărul pozitiv reprezentat în binar după care se răstoarnă toţi biţii (1->0 şi 0->1), adăugându-se apoi valoarea 1. Analog se procedează pentru a trece de la numărul negativ reprezentat în binar la numărul pozitiv corespunzător (vezi ca exemplu tabelul alăturat).

Exemple de reprezentare a numerelor întregi pe 8 biţi

0	1	1	1	1	1	1	1	127
0	1	1	1	1	1	1	0	126
…								
0	0	0	0	0	0	1	1	3
0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	-1
1	1	1	1	1	1	1	0	-2
1	1	1	1	1	1	0	1	-3
…								
1	0	0	0	0	0	1	0	-126
1	0	0	0	0	0	0	1	-127
1	0	0	0	0	0	0	0	-128

Valoarea -1 (cel mai mare număr negativ) reprezentată pe n biţi se obţine setând toţi cei n biţi pe valoarea 1.
Valoarea celui mai negativ număr pe n biţi se obţine setând toţi biţii, cu excepţia celui de semn (cel mai reprezentativ bit) pe valoarea 1 şi apoi răsturnarea tuturor biţilor. Astfel se obţine numărul negativ al cărui modul este dat de biţii astfel obţinuţi, fără a se ţine cont de semn. Implicit rezultă că în cod complementar faţă de 2, valoarea minimă reprezentabilă pe n biţi este -2^{n-1}.

Exemple:
Fie n = 4 biţi
Pentru X=6 reprezentarea în binar pe 4 biţi va fi: 0110
Pentru X=-6, reprezentarea în binar va fi cea a numărului pozitiv 2^4-6=10 şi care în binar este 1010. Aceeaşi reprezentare se obţine pornind de la reprezentarea lui +6 şi folosind regula de mai sus.

Alte coduri de reprezentare a valorilor întregi sunt:

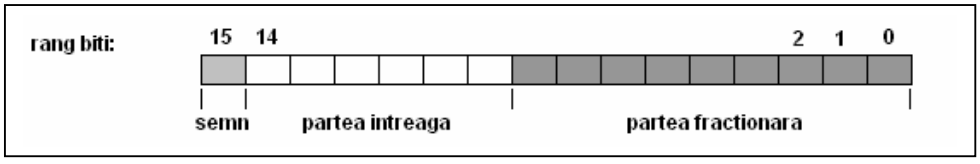
1. Codul direct
   Bitul de rang n-1 (cel mai din stânga) este rezervat pentru semn. Un număr negativ se reprezintă în cod direct reprezentând modulul său după care bitul de semn ia valoarea 1.
2. Codul invers
   Bitul de rang n-1 (cel mai din stânga) este rezervat pentru semn. Un număr negativ se reprezintă în cod invers reprezentând modulul său după care se inversează valorile tuturor biţilor reprezentării.

Calculatoarele actuale folosesc codul complementar pentru reprezentarea întregilor. Avantajul acestui cod constă în faptul că circuitele electronice pentru adunare şi scădere nu trebuie să examineze semnul numărului (vor efectua întotdeauna adunări). În plus, în acest cod, valoarea 0 (zero) are reprezentare unică.

Reprezentarea numerelor reale
Reprezentarea numerelor reale se poate face în virgulă fixă sau în virgulă mobilă.
Pentru **reprezentarea numerelor reale în virgulă fixă** se foloseşte bitul cel mai semnificativ ca bit de semn. Modulul părţii întregi şi partea fracţionară au un număr prefixat de biţi pe care se reprezintă şi se aplică următoarele reguli:

- alinierea în locaţia de memorie se face la virgula virtuală.
- dacă valoarea părţii întregi este mai mică decât valoarea maximă ce poate fi reprezentată pe biţii alocaţi părţii întregi se adaugă la stânga zerouri suplimentare.
- dacă valoarea părţii întregi este mai mare decât valoarea maximă ce poate fi reprezentată pe biţii alocaţi părţii întregi se pierd cifrele cele mai semnificative.
- dacă valoarea părţii fracţionare este mai mică decât valoarea maximă ce poate fi reprezentată pe biţii alocaţi părţii fracţionare se adaugă la dreapta zerouri nesemnificative.
- dacă valoarea părţii fracţionare este mai mare decât valoarea maximă ce poate fi reprezentată pe biţii alocaţi părţii fracţionare se pierd cifrele cele mai nesemnificative.

Exemplu
Să presupunem că se folosesc 2 octeţi (16 biţi) pentru reprezentarea numerelor reale, din care bitul de rang 15 va fi folosit pentru semn, 6 biţi vor fi folosiţi pentru reprezentarea părţii întregi şi 9 biţi pentru reprezentarea părţii fracţionare.



Numărul 19.270751953125 are reprezentarea binară (10011.010001010101). Reprezentarea acestui număr va fi:

0	0	1	0	0	1	1	0	1	0	0	0	1	0	1	0

Numărul negativ -19.270751953125 are reprezentarea binară ca şi cea a numărului pozitiv, cu deosebirea că bitul de semn este 1:

1	0	1	0	0	1	1	0	1	0	0	0	1	0	1	0

In schimb, 243.270751953125 are reprezentarea binară (11110011,010001010101) şi partea întreagă a numărului este mai mare decât valoarea maximă reprezentabilă pe cei 6 biţi alocaţi părţii întregi. Astfel, acest număr se va reprezenta sub forma:

producându-se o aşa-numită depăşire, adică pierzându-se 2 biţi cei mai semnificativi, numărul reprezentat fiind de fapt 51. 270751953125.

Reprezentarea în virgulă mobilă a numerelor reale este un tip superior de reprezentare, astfel concepută încât la depăşire se pierd cifrele cele mai puţin semnificative. Această reprezentare se bazează pe faptul că orice număr real x se poate scrie sub forma:

$$x = \pm 0.m \cdot b^e$$

unde m este mantisa numărului, b este baza de numeraţie, iar e este exponentul.
In notaţia ştiinţifică, numerele reale se notează sub forma:

$$\pm \text{ mantisa} \cdot \text{baza}^{\text{exponent}}$$

Exemple:

Valoare reală	Notaţie ştiinţifică
125,7323	$1.257323 \cdot 10^2$
-10,375	$-1.0375 \cdot 10^1$
-0,00642	$-6.42 \cdot 10^{-3}$

Scrierea valorilor reale sub forma $x = \pm 0.m \cdot b^e$ este o scriere cu mantisă subunitară, în baza 10. Orice valoare reală poate fi scrisă însă şi sub forma:

$$x = \pm 1.m \cdot 2^e$$

care înseamnă scrierea numărului în baza 2, cu mantisă între 1 şi 2, m fiind partea fracţionară a mantisei.
Valorile date mai sus ca exemplu se scriu în baza 2 sub următoarea formă:

Zecimal	Binar	Semn
125,7323	1111101.101110110111	+
-10,375	1010.011	-
-0,00642	0.00000001101001001	-

Zecimal	Reprezentarea cu mantisă între 1 şi 2
125,7323	$1.11110110111011011 \cdot 2^6$
-10,375	$-1.010011 \cdot 2^3$
-0,00642	$-1.101001001 \cdot 2^{-8}$

Astfel, din exemplele de mai sus rezultă că pentru reprezentarea valorilor reale în virgulă flotantă trebuie folosit un anumit număr de biţi, care să permită reprezentarea:

- semnului numărului
- mantisei
- exponentului
- semnului exponentului

Pentru aceasta, se foloseşte standardul IEEE (**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers), pentru reprezentarea numerelor în simplă precizie (pe 32 biţi) sau în dublă precizie (pe 64 biţi).

Bitul de semn:

- 0 corespunde unui număr pozitiv şi 1 corespunde unui număr negativ

Exponentul

Pe numărul de biţi rezervaţi pentru exponent trebuie reprezentate atât numere pozitive cât şi negative. În acest scop, exponentului propriu-zis al numărului care se reprezintă i se adaugă o anumită valoarea care depinde de tipul de precizie folosită (simplă sau dublă), numită caracteristică. În standardul IEEE simplă precizie aceasta are valoarea de 127 (2^7-1) şi în dublă precizie are valoarea 1023 (2^{10}-1). Astfel, pentru un număr al cărui exponent este 0, pe biţii alocaţi exponentului se stochează valoarea 127 (în binar 01111111).
O valoare de 200 (în binar 11001000) stocată pe biţii exponentului înseamnă de fapt exponentul 200-127=73.
Exponenţii cu toţi biţii 0 sau toţi biţii 1 sunt rezervaţi pentru numere speciale.
Pentru standardul dublă precizie se alocă 11 biţi pentru exponent, iar caracteristica este 1023.

Mantisa

Mantisa reprezintă biţii de precizie ai unui număr. Aceasta este compusă dintr-un bit implicit principal (întotdeauna 1 în scrierea cu mantisă între 1 şi 2) şi biţii fracţiei.
Pentru a afla bitul implicit principal se ţine cont de faptul că în notaţia ştiinţifică orice număr poate fi reprezentat în mai multe feluri. Astfel, numărul 5 poate fi reprezentat într-unul din modurile următoare:

$5.00 \cdot 10^0$
$0.05 \cdot 10^2$
$5000 \cdot 10^{-3}$

În scopul maximizării cantităţii de numere reprezentabile, numerele floating point sunt stocate în formă normalizată, formă care se obţine punând punctul zecimal după prima cifră nenulă. În formă normalizată, numărul 5 este reprezentat sub forma: $5 \cdot 10^0$. In baza 2, singura cifră nenulă nu poate fi alta decât cifra 1, astfel încât nu este necesar ca ea să fie reprezentată explicit şi în simplă precizie de exemplu, toţi cei 23 de biţi sunt folosiţi pentru reprezentarea părţii fracţionare a mantisei, obţinându-se practic o precizie de 24 biţi folosind doar 23 de biţi.

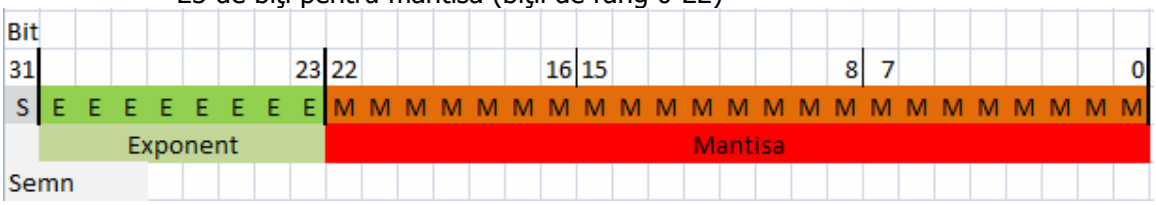
In zecimal, precizia corespunzătoare obţinută este:

$\frac{24}{\log_2 10} = 7.2$ adică de 7 cifre în simplă precizie şi

$\frac{52}{\log_2 10} = 15.65$ adică 15 cifre în dublă precizie.

Astfel, în simplă precizie, cei 32 biţi alocaţi reprezentării unui număr în virgulă flotantă sunt:

- 1 bit pentru semnul mantisei (bitul de rang 31)
- 8 biţi pentru exponent (care va include şi semnul exponentului, biţii de rang 23-30)
- 23 de biţi pentru mantisă (biţii de rang 0-22)



În cazul reprezentării cu mantisă între 1 şi 2, cifra 1 din stânga virgulei nu are bit rezervat, cifra adăugându-se numai în timpul calculelor.

nr.biţi[rangul biţilor]	Semn	Exponent	Fracţie	Caracteristică
Simplă precizie	1 [31]	8 [30-23]	23 [22-00]	127
Dublă precizie	1 [63]	11 [62-52]	52 [51-00]	1023

Valoarea V reprezentată pe un astfel de cuvât de 32 biţi se obţine în felul următor (E reprezintă e+caracteristica):

- Dacă 0<E<255 atunci $V=(-1)^S*2^{E-127}*1.F$ unde "1.F" reprezintă numărul binar creat prin adăugarea la partea fracţionară a părţii întregi 1.
- Dacă E=255 (toţi biţii exponentului sun setaţi pe valoarea 1) şi F este nenul, atunci V=NaN ("Not a number")
- Dacă E=255 (toţi biţii exponentului sun setaţi pe valoarea 1), F este nul şi S =1, atunci $V=-\infty$
- Dacă E=255 (toţi biţii exponentului sun setaţi pe valoarea 1), F este nul şi S = 0, atunci $V=+\infty$
- Dacă E=0 (toţi biţii exponentului sunt 0), F este zero (toţi biţii părţii fracţionare sunt 0) şi S =1, atunci V=-0
- Dacă E=0 (toţi biţii exponentului sunt 0), F este zero (toţi biţii părţii fracţionare sunt 0) şi S =0, atunci V=+0

Aşadar, ca reguli sunt stabilite reprezentările pentru valoarea zero, precum şi pentru valorile $\pm\infty$ şi respectiv pentru "valorile NaN.
In figura de mai jos sunt date câteva valori reprezentate în virgulă flotantă în standardul IEEE simplă precizie.

Exemplu de valori reprezentate în simplă precizie

Valoare în zecimal	bitul de semn	exponentul E=e+c	fracţia rezultată din mantisă normalizată
	1 bit	8 biţi	23 biţi
7/4	0	0 1 1 1 1 1 1 1	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-34.432175	1	1 0 0 0 0 1 0 0	0 0 0 1 0 0 1 1 0 1 1 1 0 1 0 1 0 0 0 1 1 0 0
-959818	1	1 0 0 1 0 0 1 0	1 1 0 1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0
+ 0	0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
cel mai mic număr reprezentabil	0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
cel mai mare număr reprezentabil	0	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
infinit	0	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NaN	0	1 1 1 1 1 1 1 1	Nu toţi biţii 0 sau 1

	Simplă precizie	Dublă precizie
Cel mai mic nr. pozitiv	2^{-126} sau 1.175×10^{-38}	2^{-1022} sau 2.225×10^{-308}
Cel mai mare nr. pozitiv	$(2- 2^{-23}) 2^{127}$ sau 3.403×10^{38}	$(2- 2^{-52}) 2^{1023}$ sau 1.798×10^{308}
Valoarea ε a maşinii	2^{-23} sau 1.192×10^{-7}	2^{-52} sau 2.220×10^{-16}
Precizia zecimală	6 cifre semnificative	15 cifre semnificative

Cel mai mic număr real pozitiv care poate fi reprezentat în simplă precizie are reprezentarea:
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
care înseamnă: $1.0 * 2^e$.

Partea fracţionară este zero deoarece toţi cei 23 de biţi folosiţi pentru reprezentarea acesteia sunt setaţi pe zero. La partea fracţionară se adaugă partea întreagă corespunzătoare bitului "ascuns" 1 (care nu se reprezintă în acest format simplă şi dublă precizie).
Valoarea exponentului e se deduce ţinând cont că pe biţii exponentului este reprezentată valoarea e+127. Cum E este $(00000001)_2 = 1_{10}$, rezultă e=-126. Aşadar, valoarea celei mai mici valori reale care poate fi reprezentată este:

$$1.0*2^{-126}=1.175494351\cdot 10^{-38}$$

Analog se deduce uşor valoarea celui mai mare număr real care poate fi reprezentat în simplă precizie, aceasta având reprezentarea binară:
0 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

Atenţie, nu putem folosi toţi biţii exponentului setaţi pe 1 deoarece acest caz este utilizat pentru reprezentarea valorilor speciale $\pm\infty$ şi Nan.

Din reprezentarea de mai sus rezultă că $E=2^8-1 -1=2^8-2$, adică $e=2^8-2-127=2^8-2-2^7-1 = 2(2^7-1)-2^7-1=2^7-1=127$.
A, această valoare va fi: $(1.11111111111111111111111)_2 \cdot 2^{127}$.
Valoarea părţii fracţionare este: $2^{-1}+2^{-2}+...+2^{-23} =$
$S_n = b_1 \cdot (1 + q + q^2 + ... + q^{n-1}) = b_1 \cdot (q^n - 1) / (q - 1)$, dacă $q \neq 1$, altfel $S_n = n \cdot b_1$.
$=2^{-1} \cdot (2^{-23}-1)/(2^{-1}-1)=1-2^{-23}$
Aşadar, cel mai mare număr pozitiv reprezentabil este:
$(1+1-2^{-23}*2^{127} = 3.402823467 \cdot 10^{38}$

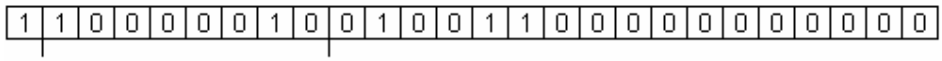
Precizia reprezentării numerelor este dată de următoarele exemple:
Valoarea 1.0 se reprezintă:
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Următoarea valoare reprezentabilă, imediat mai mare ca 1.0 va avea reprezentarea:
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Diferenţa dintre aceste valori este dată de prezenţa pe poziţia cel mai puţin semnificativă din biţii mantisei, a bitului 1. Poziţia şi valoarea acestui bit înseamnă în zecimal valoarea 0.0000001192. Această valoarea dă precizia acestei reprezentări. Aşadar, în acest format, numerele reale nu sunt reprezentate continuu ci discret, cu un pas de $1.1192\cdot 10^{-7}$.

Valoarea "Not A Number"
Valoarea NaN este folosită pentru a reprezenta valori care nu reprezintă un număr real. Aceste valori NaN sunt reprezentate printr-o succesiune de biţi cu exponentul având toţi biţii 1 şi o fracţie nenulă.

Există două feluri de valori Nan: QNaN (Quiet NaN) şi SNaN (Signalling NaN).
QNaN este un NaN cu cei mai semnificativi biţi ai fracţiei setaţi şi rezultă din operaţii aritmetice când rezultatul matematic nu este definit.
SNaN este un NaN cu cei mai semnificativi biţi a fracţiei şterşi şi astfel de valori sunt folosite pentru a semnala excepţii. Semantic, QNaN semnifică operaţie nedeterminată, iar SnaN semnifică operaţie invalidă.

Exemplu:
Care va fi reprezentarea numărului -10.375, în virgulă flotantă, simplă precizie?
1. Numărul pozitiv se transformă în binar şi se obţine: 1010.011
2. Se scrie numărul obţinut în binar sub formă normalizată: $1.010011 \cdot 2^3$
3. Se determină valoarea exponentului: 3+127=130
4. Se transformă noul exponent în binar: $(130)_{10}$=10000010
5. Se determină bitul de semn al mantisei: 1
6. Se scrie numărul:



Observaţie: Numărul real 18.0 nu are reprezentare identică cu cea a numărului întreg 18.