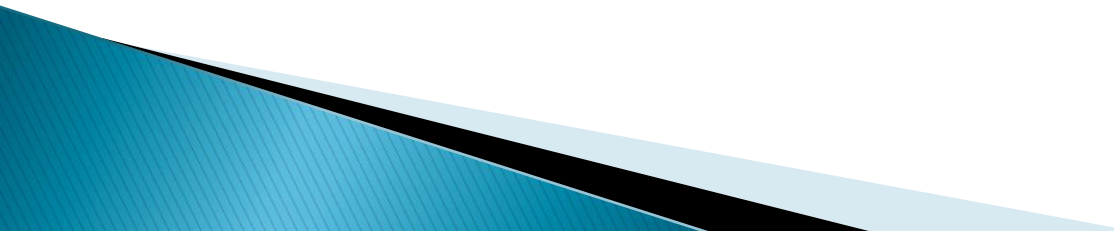


# Bazele aritmetice ale calculatoarelor numerice

– *Curs8* –

## Subiecte abordate:

---

- Sisteme de numerație
  - Reprezentarea informației numerice în calculatoare
  - Terminologia folosită în legătură cu erorile de calcul
  - Reprezentarea numerelor reale
  - Standardul IEEE 754 pentru reprezentarea numerelor în virgulă mobilă
  - Coduri alfanumerice
  - Coduri detectoare de erori
- 

## *La nivel hardware, se folosesc mai multe tipuri de date:*

- **Bit:** 0,1
  - **Șir de biți:** secvențe de biți de lungimi date:
  - **Caracter:**
    - ASCII: cod de 7 biți,
    - EBCDIC: cod de 8 biți,
    - UNICODE: cod de 16 biți
  - **Zecimal:** cifrele zecimale 0-9 codificate binar  $0000_2$  -  $1001_2$  (două cifre zecimale pot fi împachetate pe un octet sau într-un octet se poate plasa o singură cifră zecimală);
  - **Întreg (Virgulă fixă):**
    - fără semn,
    - cu semn, reprezentare în:
      - semn și modul (cod direct),
      - complementul față de 1 (cod invers)
      - complementul față de 2 (cod complementar).
  - **Real (Virgulă mobilă):**
    - precizie simplă
    - precizie dublă (cuvânt dublu: 64 de biți),
    - precizie extinsă.
- tetrada: 4 biți,  
octet/byte: 8 biți,  
semicuvânt: 16 biți,  
cuvânt: 32 de biți,  
cuvânt dublu: 64 de biți

## Sisteme de numerație

---

*Un sistem de numerație constă în totalitatea regulilor și simbolurilor/cifrelor folosite pentru reprezentarea numerelor.*

Sistemele de numerație pot fi de două tipuri: *poziționale* și *nepoziționale*.

Într-un sistem pozițional valoarea/ponderea unui simbol depinde de poziția pe care o ocupă în reprezentarea unui număr dat, în timp ce într-un sistem nepozițional acest lucru nu are loc.

Un număr întreg este reprezentat, într-un sistem de numerație pozițional, în baza  $b$ , sub forma unui  $n$ -tuplu de simboluri  $x_i$  unde  $x_i$  reprezintă o cifră a sistemului de numerație.

$$N_b = x_{n-1} x_{n-2} x_{n-3} \dots x_i \dots x_1 x_0$$

## Reprezentarea informației numerice în calculatoare

---

Calculatoarele moderne operează, atât cu numere întregi (cu semn și fără semn), cât și cu numere reale.

### Numere întregi cu semn:

- semn și modul (cod direct): 010001001110 ( $>0$ )  
110001001110 ( $<0$ )
- complementul față de 1 (cod invers): 001110110001
- complementul față de 2 (cod complementar): 001110110010
  
- reprezentare în exces: are ca efect deplasarea reprezentării în complementul față de doi.
  
- codul binar – zecimal: rangurile zecimale sunt codificate prin tetrade binare.

Un număr real se reprezintă prin două câmpuri: mantisa/fracția  $f$ , cu semn, și exponent  $e$ :

$$N_r = s e f$$

unde:

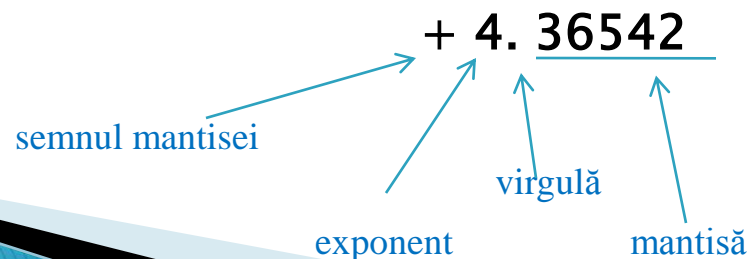
- $s$  reprezintă semnul mantisei, codificat printr-un bit
- $f$  constituie mantisa sub forma unui număr subunitar normalizat ,
- $e$  specifică exponentul, de regulă, deplasat cu o anumită cantitate pentru a-l face  $\geq 0$

În numărul:

$$0,36542 \times 10^4$$

gama este stabilită de către numărul de ranguri ale exponentului și de către baza, care este 10, în cazul de față.

Precizia este asociată cu numărul de ranguri ale părții subunitare, 5 în exemplul de mai sus. Precizia și gama impun un număr de 6 ranguri zecimale, la care se mai adaugă unul pentru codificarea semnelui părții subunitare/mantisei.



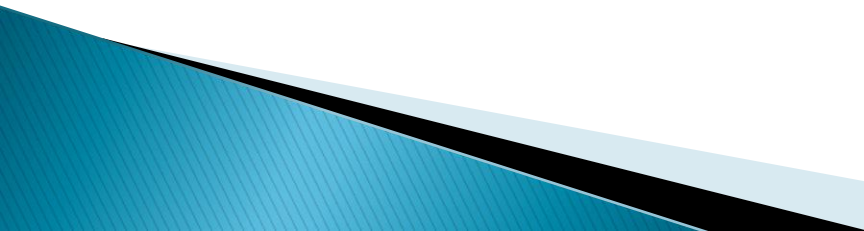
Reprezentarea normalizată a fost introdusă pentru a se evita reprezentările multiple ale aceluiași număr:

$$3654,2 \times 10^0 = 36,542 \times 10^2 = 0,36542 \times 10^4$$

*Aceasta se obține prin deplasarea spre stânga a mantisei, astfel încât, imediat la dreapta virgulei să se afle o cifră diferită de 0. Pe măsura deplasării mantisei la stânga se incrementează și exponentul. Operația nu are sens atunci când mantisa are toate rangurile egale cu 0.*

## Terminologia folosită în legătură cu erorile de calcul

---

- **Precizia** constituie un termen asociat cu lungimea cuvântului, numărul de biți disponibili într-un cuvânt pentru reprezentarea unui număr dat.
  - **Acuratețea** reprezintă o măsură a apropierii unei aproximații față de valoarea exactă.
  - **Gama** reprezintă mulțimea numerelor reprezentabile într-un sistem dat.
  - **Rezoluția** constituie mărimea diferenței/distanței între două numere sau cifre adiacente.
  - **Trunchierea**, cunoscută și sub denumirea de rotunjire prin lipsă, este utilizată în cazurile în care precizia nu este suficientă pentru reprezentarea corectă a numărului stocat.
  - **Rotunjirea** este metoda prin care se caută să se selecteze valoarea cea mai apropiată de valoarea inițială a numărului
- 



• *Depășirea* apare când rezultatul unui calcul este prea mare pentru a putea fi reprezentat în sistem.

• *Depășirea superioară și depășirea inferioară* apar la reprezentarea numerelor în virgulă mobilă, atunci când rezultatul este mai mare sau mai mic decât cel mai mare sau cel mai mic număr care poate să fie reprezentat în sistem.

• *Erorile introduse la conversia numerelor din baza zece în baza doi* apar datorită faptului că cele mai multe numere nu reprezintă multipli ai unor fracții binare.

## Reprezentarea numerelor reale

---

$$X = f * r^e$$

Dacă *exponentul* are  $k$  biți și mantisa  $m$  biți (exclusiv semnul  $s$ ), numărul  $X$  se va putea reprezenta astfel:

$$X = s e_{k-1} e_{k-2} \dots e_0 f_{-1} f_{-2} \dots f_{-m}$$

Cele două formate

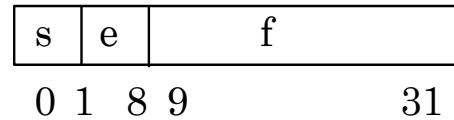
standard sunt:  
(IEEE 754)

→ *formatul scurt* sau *de bază*, pe un cuvânt de 32 de biți, care asigură o viteză mai mare de operare;

→ *formatul lung* sau *dublu de bază*, pe un cuvânt de 64 de biți, care asigură o precizie mai mare de lucru.

## Formatul scurt

---



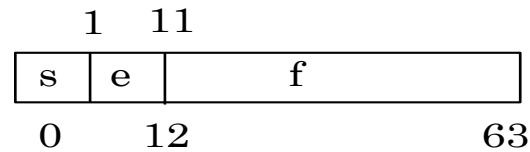
- bitul 0:  $s = \text{semnul mantisei}$  ( $s = 1$  - mantisa negativă);
- biții 1-8:  $e = \text{exponentul deplasat}$  (deplasarea este  $2^7 - 1$ );
- biții 9-31:  $f = \text{mantisa/fracția}$  (când  $e \neq 0$ , se presupune un bit egal cu 1, la stânga lui  $f$ ; virgula se plasează între acest bit și primul bit explicit al mantisei);

Numărul reprezentat într-un format scurt este:

$$(-1)^s 2^{e-(2^7-1)} \cdot (1 + f) , \text{ cu condiția } e \neq 0 .$$

## Formatul lung

---



- bitul 0:  $s = \text{semnul mantisei}$  ( $s = 1$  - mantisa negativă);
- biții 1-11:  $e = \text{exponentul deplasat}$  (deplasarea este  $2^{10} - 1$ );
- biții 12-63:  $f = \text{mantisa/fracția}$  (când  $e \neq 0$ , se presupune un bit egal cu 1, la stânga lui  $f$ ; virgula se plasează între acest bit și primul bit explicit al mantisei);

Numărul reprezentat într-un format lung este:

$$(-1)^s 2^{e-(2^{10}-1)} \cdot (1 + f), \text{ cu condiția } e \neq 0 .$$

# Standardul IEEE 754 pentru reprezentarea numerelor în virgula mobilă

## Formatul:

- format scurt
- format lung
- format simplu – extins este structurat astfel:



1 bit de semn, 1 bit pentru partea întreaga (j) a mantisei, cel puțin 31 de biți pentru partea fracționară a mantisei (f) și un exponent ce poate lua valori cuprinse între minimum  $m = -1022$  și maximum  $M = 1023$ .

- format dublu – extins este structurat astfel:



1 bit de semn, 1 bit pentru partea întreaga (j) a mantisei, cel puțin 63 de biți pentru partea fracționară a mantisei (f) și un exponent ce poate lua valori cuprinse între minimum  $m = -16382$  și maximum  $M = 16383$ .

# Rotunjirea



Standardul IEEE 754 include atât un mecanism standard-implicit de rotunjire, cât și alte trei mecanisme ce pot fi selectate de utilizator.

*rotunjeste  
numărul la cea  
mai apropiată  
valoare  
reprezentabilă*

- rotunjirea către  $+\infty$  asigură cea mai apropiată valoare dar nu mai mică decât a numărului dat;
- rotunjirea către  $-\infty$  furnizează cea mai apropiată valoare, dar nu mai mare decât numărul dat;
- rotunjirea către 0 (trunchiere) asigură valoarea cea mai apropiată, dar nu mai mare decât numărul dat în modul.

## Valori speciale

Standardul de reprezentare în virgula mobilă suporta aritmetica cu numere infinite folosind :

- *Modalitatea infinit-afină* definită prin relația:

$$-\infty < (\text{oricare număr finit}) < +\infty$$

- *Modalitatea infinit-proiectivă* compară întotdeauna numerele egale, indiferent de semn. Standardul definește un set de operații astfel încât folosirea unui operand infinit nu va conduce la un rezultat eronat.

**NaN** constituie o valoare specială, introdusă pentru a semnaliza operații invalide sau operații care produc rezultate invalide cu o valoare specială.

## Operații

În standard sunt definite operațiile aritmetice de bază: adunarea, scăderea, înmulțirea și împărțirea; rădăcina pătrată și găsirea restului la împărțire; conversia formatului în: virgula mobilă, numere întregi și numere zecimale codificate binar BCD (cu excepția numerelor BCD extinse EBCD); compararea numerelor în virgulă mobilă.

## Excepții și capcane

- implementarea va asigura câte un indicator pentru fiecare tip de excepție.
- răspunsul standard la excepție constă în continuarea operației, fără activarea capcanei.

*Operație invalidă* este un tip de excepție se încadrează în *două clase*: excepții de *operand invalid* și *rezultat invalid*. În ambele cazuri, dacă nu este activată o capcana, rezultatul va fi NaN.

*Excepțiile de rezultat invalid* apar atunci când rezultatul unei operații nu este corect în raport cu formatul destinație.

*Alte excepții:* {  
împărțire cu zero;  
depășire superioară;  
depășire inferioară;  
rezultat inexact.



*Capcanele*, care corespund fiecărei excepții pot fi activate/dezactivate de către utilizator. Când este activată, excepția transferă controlul la o rutină pentru manipularea capcanei (furnizată de utilizator sau de către sistem).

Rutină trebuie să primească următoarele informații:

- tipul excepției, care a apărut,
- tipul operației, care s-a executat,
- formatul destinației,
- rezultatul corect rotunjit
- valorile operandului, în cazurile împărțirii cu zero și excepțiilor de operand invalid.

## STANDARDUL ARITMETIC

---

Metodele de implementare ale operațiilor aritmetice în virgulă mobilă se vor exemplifica considerându-se un acumulator cu următoarea structură (numere au mantisa de patru biți):



D: bitul de depășire,  
F<sub>1</sub> - F<sub>4</sub>: cei 4 biți ai mantisei,  
G: bitul de gardă,  
R: bitul de rotunjire,  
ST: bitul de legatura/lipitura, "stiky".

## Adunarea și scăderea în virgulă mobilă

### Adunarea modulelor

- *Denormalizarea*: numărul cu exponent mai mic se încarcă în acumulator și se deplasează cu un număr de poziții spre dreapta, egal cu diferența exponenților. Dacă exponenții sunt egali oricare dintre numere/operanzi poate fi încărcat în acumulator, fără a se efectua vreo deplasare.
- *Adunarea*: cel de-al doilea operand este adunat la acumulator.
- *Normalizarea*: dacă s-a poziționat în unu bitul D atunci se deplasează conținutul acumulatorului cu un bit spre dreapta.
- *Rotunjirea*: se adună 1 la poziția G, după care, dacă  $G = R = ST = 0$ , se forțează  $F_4 \leftarrow 0$ .
- *Renormalizarea*: dacă D este poziționat în unu se face deplasarea conținutului acumulatorului spre dreapta.
- *Depășirea*: se verifică dacă a avut loc o depășire a exponenului.

## Scăderea modulelor

- *Denormalizarea*: se încarcă numărul cu modulul cel mai mic în acumulator și se deplasează la dreapta, dacă este necesar. Rezultatul va fi zero dacă și numai dacă operanzii sunt egali, abandonându-se operațiile următoare.
- *Scădearea*: se scade conținutul acumulatorului din celalalt operand, păstrând rezultatul în acumulator (dacă  $ST = 1$  se va genera un împrumut);
- *Normalizarea*: se deplasează acumulatorul la stânga până când  $F_1$  devine egal cu 1;
- *Rotunjirea*: se adună 1 la bitul G și apoi, dacă  $G = R = 0$  și  $ST = 0$ , se forțează ; nu este necesară rotunjirea dacă pentru normalizare au fost necesare mai multe deplasări la stânga;
- *Renormalizarea*: în cazul în care rotunjirea a condus la depășire, se efectuează o deplasare spre dreapta.

## Înmulțirea

- *Înmulțirea*: se formează produsul în lungime dublă;
- *Normalizarea*: pentru o eventuală normalizare a produsului se face o deplasare cu un bit;
- *Poziționarea biților G, R, ST*: fie produsul normalizat în lungime dublă:  
 $F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8$  , atunci  $G = F_5, R = F_6, ST = F_7 \cup F_8$  ;
- *Rotunjirea*: se realizează după cum s-a arătat anterior;
- *Renormalizarea*: se realizează după cum s-a arătat anterior;
- *Erori*: se verifică depășirea superioară/inferioară a exponentului.

## Împărțirea

- *Împărțirea*: se formează primii șase biți ai cântului normalizat:  
 $F_1 F_2 F_3 F_4 F_5 F_6$
- *Poziționarea biților G, R, ST*: se forțează:  $G = F_5, R = F_6$   
și  $ST = rest$  ;
- *Rotunjirea*: se efectueaza după regulile menționate anterior;
- *Renormalizarea*: se efectuează după regulile menționate anterior.

## Coduri alfanumerice

---

Se utilizează trei sisteme de codificarea a caracterelor alfa numerice:

- **ASCII** ( **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)
- **EBCDIC** (**E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode)
- **Unicode**

**ASCII** → este utilizat pentru reprezentarea caracterelor alfanumerice și folosește 7 biți pe caracter. Toate cele  $2^7$  coduri posibile reprezintă caractere valide.

**EBCDIC** → este un cod pe 8 biți utilizat de către IBM

**Unicode** → reprezintă un standard (ISO/IEC 10646) pentru reprezentarea caracterelor cu ajutorul unor cuvinte de 16 biți, ceea ce permite codificarea a 65536 caractere. Unicode oferă o modalitate consistentă pentru codificarea textelor în care sunt utilizate alfabetul/caractere diferite cum ar fi cele Latine, Grecești, Chinezești, Japoneze etc.

## Coduri detectoare de erori

*Distanța Hamming* definește distanța logică între două coduri de caractere valide și este măsurată prin numărul de biți prin care diferă acestea.



Pentru codul ASCII această distanță este egală cu 1.



Adăugând un singur bit redundant, la codul ASCII al fiecărui caracter alfanumeric, se poate detecta o singură eroare, întrucât codul eronat se va plasa între două coduri valide ASCII.

O metodă de recodificare a codului ASCII, pentru a obține o distanță Hamming egală cu 2, constă în introducerea unui bit de paritate, plasat la stânga codului normal ASCII. Acest bit va fi calculat pe baza *sumei modulo 2* a biților egali cu 1 din codul ASCII.

P	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	Caracterul
1	1	1	0	0	1	0	0	d
0	1	1	0	0	1	0	1	e
0	1	1	0	0	1	1	0	f
1	1	1	0	0	1	1	1	g
0	1	0	0	0	1	0	0	D



Pentru detectarea și corectarea unei erori în cadrul fiecărei poziții a unui cod ASCII, fiecărui cod valid ASCII trebuie să i se asocieze alte 7 coduri invalide, în care se va modifica exact un singur bit.

*Pentru un cod de  $k$  biți, se consideră că numărul biților redundanți egal cu  $r$ .  
Relația care trebuie îndeplinită este :*

$$(k + r + 1) \leq 2^r$$



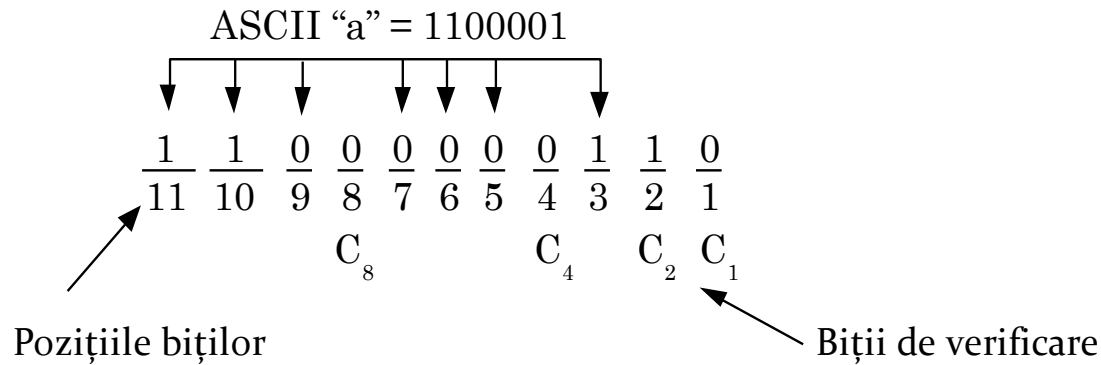
În cazul  $k = 7$ , dacă se caută întregul  $r$  care satisface relația de mai sus, rezultă  $r = 4$ , ceea ce conduce la un cod ASCII, prevăzut cu caractere redundante, de  $7 + 4 = 11$  biți.



Asignarea celor 4 biți redundanți la cuvintele originale se va face astfel încât să poată fi identificată o eroare la un singur bit. Fiecărui bit din cuvântul codificat, incluzând biții de verificare/redundanți, îi este asignată o combinație dată de biți de verificare  $C_8 C_4 C_2 C_1$  .

$C_8$	$C_4$	$C_2$	$C_1$	Bit verificat
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
0	0	1	1	4
0	1	0	0	5
0	1	0	1	6
0	1	1	0	7
0	1	1	1	8
1	0	0	0	9
1	0	0	1	10
1	0	1	0	11

*Acest mod particular de amplasare poartă numele de Cod de Corectare a unei Erori Singulare (CCES) sau SEC (Single Error Correcting).*



- Bitul de verificare  $C_1 = 0$  realizează paritatea pară pentru grupul de biți  $\{1, 3, 5, 7, 9, 11\}$ .
- Bitul de verificare  $C_2 = 0$  furnizează paritatea pară pentru grupul de biți  $\{2, 3, 6, 7, 10, 11\}$ .
- Bitul de verificare  $C_4 = 0$  asigură paritatea pară pentru grupul de biți  $\{4, 5, 6, 7\}$ .
- Bitul de verificare  $C_8 = 0$  stabilește paritatea pară pentru grupul de biți  $\{8, 9, 10, 11\}$ .



Bitul  $n$  al cuvântului codificat este verificat prin biții din pozițiile  $1, \dots, i$  a căror sumă este egală cu  $n$ .