6. Bazele Aritmetice ale Calculatoarelor Numerice.

6.1. Introducere.

Intrucat elementele de memorare sunt constituite din dispozitive cu doua stari stabile, iar elementele de prelucrare a informatiei sunt bazate pe circuite logice, care opereaza pe baza logicii bivalente, intr-un calculator numeric datele sunt reprezentate in binar, sub forma unor succesiuni de unitati si zerouri. Exista numeroase posibilitati pentru reprezentarea datelor, care se deosebesc intre ele prin expresibilitate, cost de implementare, usurinta conversiilor de la un format la altul, cat si prin alte considerente.

Astfel, in cadrul unui calculator numeric, la nivel hardware, se folosesc mai multe tipuri de date:

- **Bit**: 0,1
- **Sir de biti**: secvente de biti de lungimi date:
  - tetrada: 4 biti,
  - octet/byte: 8 biti,
  - semicuvant: 16 biti,
  - cuvant: 32 de biti,
  - cuvant dublu: 64 de biti
- **Caracter**:
  - ASCII: cod de 7 biti,
  - EBCDIC: cod de 8 biti,
  - UNICODE: cod de 16 biti
- **Zecimal**:
  - cifrele zecimale 0-9 codificate binar 0000_2 -1001_2 (doua cifre zecimale pot fi impachetate pe un octet sau intr-un octet se poate plasa o singura cifra zecimala);
- **Intreg (Virgula fixa):**
  - fara semn,
  - cu semn, reprezentare in:
    - semn si modul (cod direct),
    - complementul fata de 1 (cod invers),
    - complementul fata de 2 (cod complementar).

- **Real (Virgula mobila)**:
  - precizie simpla (cuvant de 32 de biti),
  - precizie dubla (cuvant dublu: 64 de biti),
  - precizie extinsa.

Calculatoarele poseda elemente de stocare a datelor, de tipul registrelor sau al celulelor/locatiilor de memorie, care dispun de un numar finit de elemente/ranguri, ceea ce afecteaza precizia calculului. Astfel, la programarea unor aplicatii numerice, trebuie avute in vedere aspectele legate de precizia limitata a reprezentarii informatiei.

6.2. Sisteme de numeratie.

6.2.1. Reprezentarea numerelor.

Un sistem de numeratie consta in totalitatea regulilor si simbolurilor/cifrelor folosite pentru reprezentarea numerelor. Sistemele de numeratie pot fi de doua tipuri: *pozitionale* si *nepozitionale*. Intr-un sistem pozitional valoarea/ponderea unui simbol depinde de pozitia pe care o ocupa in reprezentarea unui numar dat, in timp ce, intr-un sistem nepozitional acest lucru nu are loc. Ca exemplu de sistem nepozitional se poate da sistemul de numeratie roman.

Sistemele de numeratie pozitionale mai poarta numele si de *sisteme de numeratie ponderate*, intrucat valoarea unei cifre depinde de pozitia ei in reprezentarea numarului dat.

Un numar intreg N este reprezentat, intr-un sistem de numeratie pozitional, in baza b, sub forma unui n-tuplu de simboluri x_i,

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

unde x_i reprezinta o cifra a sistemului de numeratie.

O cifra x_i poate lua valori intregi cuprinse intre 0 si b-1 ($0 \leq x_i \leq b$ -1), baza b reprezentand numarul valorilor posibile pe care le poate lua o cifra oarecare x_i.

In general, un numar, constituit dintr-o parte intreaga si o parte subunitara are umatoarea reprezentare in baza b:

$$N_b = x_{n-1}\, x_{n-2}\, x_{n-3} \ldots x_i \ldots x_1\, x_0\, ,\, x_{-1}\, x_{-2}\, x_{-3} \ldots x_{-i} \ldots x_{-m}\, , (0 \leq x_i \leq b\text{-}1).$$

Valoarea N a numarului N_b se calculaeza cu ajutorul urmatoarei expresii:

$$N = \sum_{i=-m}^{n-1} x_i . b^i$$

Fie numarul $(435,25)_{10}$, in baza 10 (n = 3, m = 2 si b = 10).

$$4 \times 10^2 + 3 \times 10^1 + 5 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2} =$$

$$(400)_{10} + (30)_{10} + (5)_{10}\,(1) + (2/10)_{10} + (5/100)_{10} = (435,25)_{10}$$

In continuare se considera numarul $(1011,01)_2$, pentru care: n = 4, m = 2 si b = 2:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} =$$

$$(8)_{10} + (0)_{10} + (2)_{10} + (1)_{10} + (0/2)_{10} + (1/4)_{10} = (11,25)_{10}$$

In ultimul exemplu se prezinta metoda polinomiala de conversie a numerelor, reprezentate in baza 2, in numere reprezentate in baza 10.

6.2.2. Conversia numerelor dintr-o baza in alta.

Intr-un sistem de calcul datele sunt reprezentate in mai multe sisteme de numeratie. Astfel, datele de la intrare si cele de la iesire sunt, in general, reprezentate in baza 10. In memoria calculatorului si in unitatea de prelucrare datele sunt reprezentate in baza 2. Sunt situatii in care datele, care se prelucreaza, sunt reprezentate in baza 10, cifrele zecimale fiind codificate prin tetrade binare. Pentru a usura operatiile de programare, uneori, numerele binare sunt convertite in numere reprezentate in baza 8 sau baza 16.

Plecand de la reprezentarea numerelor sub forma coeficientilor dezvoltarii polinomiale, in raport cu baza, conversia se poate efectua prin operatii repetate de impartire/inmultire in conditiile numerelor intregi/subunitare.

Se presupune un numar N constituit dintr-o parte intreaga N_i si o parte subunitara N_f:

$$N = N_i + N_f$$

Astfel:

$N_i = x_{n-1}\, x_{n-2}\, x_{n-3} \ldots x_i \ldots x_1\, x_0$ si

$N_f = 0, x_{-1}\, x_{-2}\, x_{-3} \ldots x_{-i} \ldots x_{-m}$

<u>*Conversia numerelor intregi din baza b in baza q*</u> presupune impartirea repetata a catului de la impartirea precedenta la noua baza si retinerea restului, mai mic decat noua baza, pana cand catul curent devine mai mic decat noua baza. Ca prim cat se ia numarul N_i, care urmeaza sa fie convertit. Resturile obtinute formeaza cifrele noului numar, incepand cu cea mai putin semnificativa. Toate operatiile se efectueaza in baza de plecare b.

Din punctul de vedere al valorii exprimate:

$(N_i)_b = (N_i)_q$

$(N_i)_q$ se poate scrie ca un *p-tuplu* astfel:

$N_i = y_{p-1} y_{p-2} y_{p-3} \dots y_i \dots y_1 y_0$

Pentru a obtine coeficientii y_i, ai dezvoltarii in baza q, se va recurge la urmatoarea secventa de operatii:

$(N_i)_b : q = (N_i)_b^1 + y_0 / q$, cifra curenta a noului numar este y_0

$(N_i)_b^1 : q = (N_i)_b^2 + y_1 / q$, cifra curenta a noului numar este y_1

……..

$(N_i)_b^{p-2} : q = (N_i)_b^{p-1} + y_{p-2} / q$, cifra curenta a noului numar este y_{p-2}

unde $0 \leq (N_i)_b^{p-1} \leq q$-1 reprezinta cifra cea mai semnificativa a noii reprezentari

Exemplu:

$(17)_{10} = (?)_2$

17 : 2 = 8 +1/2 , $y_0 = 1$;

8 : 2 = 4 +0/2 , $y_1 = 0$;

4 : 2 = 2 +0/2 , $y_2 = 0$;

2 : 2 = 1 + 0/2 , $y_3 = 0$;

$y_4 = 1$

$(17)_{10} = 10001_2$

<u>*Conversia numerelor subunitare din baza b in baza q*</u> presupune inmultirea repetata a partii subunitare, care rezulta de la inmultirea precedenta, cu noua baza si retinerea partii intregi pana cand partea subunitara curenta devine 0 sau pana cand se epuizeaza rangurile de reprezentare in noua baza. Ca prima parte subunitara se ia numarul N_f , care urmeaza sa fie convertit. Intregii obtinuti formeaza cifrele noului numar, incepand cu cea mai semnificativa. Toate operatiile se efectueaza in baza de plecare *b*.

Din punctul de vedere al valorii exprimate:

$(N_f)_b = (N_f)_q$

$(N_f)_q$ se poate scrie ca un *m*-tuplu astfel:

$N_f = y_{-1} y_{-2} y_{-3} \dots y_{-i} \dots y_{-m}$

Pentru a obtine coeficientii y_{-i} ai dezvoltarii in baza q se va recurge la urmatoarea secventa de operatii:

$(N_f)_b \times q = (N_f)_b^{-1} + y_{-1}$, cifra curenta a noului numar este y_{-1}

$(N_f)_b^{-1} \times q = (N_f)_b^{-2} + y_{-2}$, cifra curenta a noului numar este y_{-2}

……..

$(N_f)_b^{-m+1} \times q = 0 + y_{-m}$, cifra curenta a noului numar este y_{-m}

In cazul in care $(N_f)_b^{-i} = 0$, procesul se opreste. Se poate constata faptul ca procesul de conversie a numerelor subunitare poate fi insotit de erori, in cazul unui numar limitat de ranguri pentru reprezentarea in noua baza sau in cazul aparitiei unor secvente repetate de unitati si zerouri.

Exemplu:

$(0,125)_{10} = (0,001)_2$

Se considera urmatorul exemplu:

$(0,1)_{10} = (?)_2$

$0,1 \times 2 = 0 + 0,2$; $y_{-1} = 0$;
$0,2 \times 2 = 0 + 0,4$; $y_{-2} = 0$;
$0,4 \times 2 = 0 + 0,8$; $y_{-3} = 0$;
$0,8 \times 2 = 1 + 0,6$; $y_{-4} = 1$;
$0,6 \times 2 = 1 + 0,2$; $y_{-5} = 1$;
$0,2 \times 2 = 0 + 0,4$; $y_{-6} = 0$;
$0,4 \times 2 = 0 + 0,8$; $y_{-7} = 0$;
$0,8 \times 2 = 1 + 0,6$; $y_{-8} = 1$;
$0,6 \times 2 = 1 + 0,2$; $y_{-9} = 1$;
$0,2 \times 2 = 0 + 0,4$; $y_{-6} = 0$;
……

$(0,1)_{10} = (0,00011000110....)_2$

Se poate observa ca, in acest ultim caz, conversia nu se efectueaza exact.

Conversiile intre baze, care reprezinta puteri ale lui 2, constituie cazuri particulare, si se efectueaza mecanic. Astfel, in cazul numerelor reprezentate in octal/hexazecimal, cifrele octale/hexazecimale se inlocuiesc cu triadele/tetradele binare corespunzatoare si invers.

Numerele octale/hexazecimale se exploreaza, pentru conversia in binar, de la dreapta la stanga.

Mai jos se prezinta un tabel de corespondenta intre cifrele octale, zecimale si hexazecimale, pe de-o parte si echivalentele lor binare, pe de alta parte.

binar	octal	zecimal	hexazecimal
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

Exemple:

$(011101)_2 = (011)_2(101)_2 = (3)_8\,(5)_8 = (35)_8$

$(011101)_2 = (0001)_2(1101)_2 = (1)_{16}\,(D)_{16} = (1D)_{16}$

6.2.3. Reprezentarea informatiei numerice in calculatoare.

Calculatoarele moderne opereaza, atat cu numere intregi (cu semn si fara semn), cat si cu numere reale.

In cazurile numerelor intregi cu semn si al mantisei numerelor reale semnul este codificat prin bitul plasat in extrema stanga. Semnul "-" este codificat prin "1", iar semnul "+" prin "0".

Pentru numerele intregi cu semn:

$$N_i = x_{n-1}\, x_{n-2}\, x_{n-3} \ldots x_i \ldots x_1\, x_0$$

semnul este codificat prin bitul x_{n-1}.

Un numar real N_r se reprezinta prin doua campuri: mantisa/fractia f, cu semn, si exponent e:

$$N_r = s\ e\ f$$

unde:

- s reprezinta semnul mantisei, codificat printr-un bit, conform conventiei mentionate mai sus;
- f constituie mantisa sub forma unui numar subunitar normalizat ($|m| \geq 1/2$);
- e specifica exponentul, de regula, deplasat cu o anumita cantitate pentru a-l face ≥ 0 .

O discutie mai amanuntita, privind reprezentarea numerelor reale se va face intr-un paragraf ulterior.

6.3. Coduri de reprezentare a numerelor intregi, cu semn, in calculatoare.

Numerele intregi, cu semn, se pot reprezenta in calculatoare in trei moduri diferite, numite uneori si coduri de reprezentare:

- *semn si modul* (*cod direct*),
- *complementul fata de* 1 (*cod invers*),
- *complementul fata de* 2 (*cod complementar*).

Codul direct (semn si modul).

$$[x]_d = semn\ |x|$$

$$[x]_d = 0\, x_{n-2}\, x_{n-3} \ldots x_i \ldots x_1\, x_0 \text{ , pentru } x > 0;$$

$$[x]_d = 1\, x_{n-2}\, x_{n-3} \ldots x_i \ldots x_1\, x_0 \text{ , pentru } x < 0$$

Se poate observa ca, in acest cod, 0 are doua reprezentari, daca este afectat de semn:

$$[+0]_d = 00000..000 \text{ si}$$

$$[-0]_d = 10000..000$$

Astfel, un numar x in cod direct, reprezentat pe n ranguri, poate lua valori in gama:

$$-2^{n-1} + 1 \leq x \leq 2^{n-1} - 1,$$

ceea ce face ca pentru:

- $n = 8$ valorile minime/maxime sa fie –127 / +127, iar
- $n = 16$ valorile minime/maxime sa fie –32767 / +32767.

Reprezentarea in modul si semn este utila pentru implementarea operatiei de inmultire, dar prezinta unele dificultati la adunare si scadere.

Codul invers (complementul fata de unu).

Denumirea de *complementul fata de unu* provine de la faptul ca reprezentarea in virgula fixa s-a realizat in calculatoare, mai intai, pentru numere subunitare, numerele negative fiind stocate sub forma complementului fata de 2, diminuat cu 1, prin scaderea lui $|x|$ din 1. In cazul numerelor binare negative intregi, reprezentate pe n ranguri, codurile inverse se obtin prin scaderea modulelor acestora din $2^n - 1$.

$$[x]_i = 0\, x_{n-2}\, x_{n-3} \ldots x_i \ldots x_0 \text{ , pentru } x > 0;$$

$$[x]_i = 1\, \overline{x_{n-2}}\, \overline{x_{n-3}} \ldots \overline{x_i} \ldots \overline{x_0} \text{ , pentru } x < 0$$

unde: $\overline{x_i}$ reprezinta inversul lui x_i

Se poate observa ca, in acest cod, 0 are doua reprezentari, daca este afectat de semn:

$[+0]_i$ = 00000..000 si

$[-0]_i$ = 11111..111

Astfel, un numar x in cod invers, reprezentat pe n ranguri, poate lua valori in gama:

$$-2^{n-1} + 1 \leq x \leq 2^{n-1} - 1,$$

ceea ce face ca pentru:

- n = 8 valorile minime/maxime sa fie –127 / +127, iar
- n =16 valorile minime/maxime sa fie –32767 / +32767.

Reprezentarea in codul invers este identica cu reprezentarea in cod direct, in cazul numerelor pozitive. Pentru a obtine codul invers al unui numar se inverseaza valorile binare ale tuturor rangurilor, operatie extrem de usor de realizat in hardware.

Codul complementar (complementul fata de doi).

Denumirea de *complementul fata de doi* provine de la faptul ca reprezentarea in virgula fixa s-a realizat in calculatoare mai intai pentru numere subunitare, iar numerele negative se stocau sub forma complementului fata de 2, prin scaderea lui $|x|$ din 2. In cazul numerelor binare, intregi negative, reprezentate pe n ranguri, codurile inverse se obtin prin scaderea modulelor acestora din 2^n.

$[x]_c = 0\, x_{n-2}\, x_{n-3} \ldots x_i \ldots x_0$, pentru $x \geq 0$;

$[x]_c = 1\, \tilde{x}_{n-2}\, \tilde{x}_{n-3} \ldots \tilde{x}_i \ldots \tilde{x}_0$, pentru $x < 0$

unde: $1\, \tilde{x}_{n-2}\, \tilde{x}_{n-3} \ldots \tilde{x}_i \ldots \tilde{x}_0$ se obtine ca urmare a operatiei: $2^n - |x|$

Se poate observa ca, in acest cod, 0 are o singura reprezentare:

$[0]_c = 0\,0000..000$

Astfel, un numar x in cod complementar , reprezentat pe n ranguri, poate lua valori in gama:

$$-2^{n-1} \leq x \leq 2^{n-1} - 1,$$

ceea ce face ca pentru:

- n = 8 valorile minime/maxime sa fie –128 / +127, iar
- n = 16 valorile minime/maxime sa fie –32768 / +32767.

Pentru: $x > 0$ $[x]_d = [x]_i = [x]_c$

Pentru: $x < 0$ $[x]_c = [x]_i + 1 = 2^n - |x|$.

Se poate observa usor ca, prin mijloace hardware, codul complementar al unui numar negativ se poate obtine adunand, la inversul numarului (obtinut prin negarea logica a tuturor rangurilor), o unitate in cel mai putin semnificativ rang.

Reprezentarea in cod complementar faciliteaza realizarea hardware-lui necesar operatiilor de adunare si scadere in calculatoare.

Reprezentarea in exces.

Reprezentarea in exces este cunoscuta si sub numele de reprezentare deplasata. Ideea de baza consta in aceea de a atribui celui mai mic numar (numar negativ) valoarea cea mai mica intreaga, care se poate reprezenta in calculator, manipuland in acest mod numere fara semn. In cazul reprezentarii numerelor $(14)_{10}$ si $(-14)_{10}$ sub forma unor numere binare pe 8 biti, folosind forma *exces* 128 va trebui sa se obtina corespondentele binare ale numerelor $(128 + 14 = 142)_{10}$ si $(128 + (- 14) = 114)_{10}$. Astfel, codul binar in exces 128 pentru $(-14)_{10}$ este $(01110010)_2$.

Nu exista o semnificatie numerica a valorii *in exces*, ea are ca efect deplasarea reprezentarii in complementul fata de doi. Pentru cazul studiat valoarea *exces* a fost astfel aleasa incat sa aibe acelas sablon de biti ca si cel mai mare numar negativ, ceea ce face ca numerele sa apara ca si cand ar fi sortate ca numere binare fara semn. Celui mai mic numar negativ $(-128)_{10}$ ii va corespunde $(00000000)_2$, in timp ce celui mai mare numar pozitiv $(127)_{10}$ ii va corespunde $(11111111)_2$. Aceasta reprezentare simplifica efectuarea operatiei de comparare a numerelor, ceea ce este esential pentru realizarea hardware-lui necesar operarii asupra exponentilor, la reprezentarea in virgula mobila.

6.4. Codul binar-zecimal.

Numerele pot fi reprezentate in sistemul de numeratie cu baza 10, in conditiile in care rangurile zecimale sunt codificate prin tetrade binare. Acesta este *codul binar-zecimal* sau BCD (*Binary Coded Decimal*). Posibilitatile de codificare oferite de catre o tetrada binara nu sunt epuizate in cazul BCD, deoarece raman neutilizate 6 tetrade, dintr-un total de 16.

In cazurile in care se doreste o codificare BCD cu semn, tetradele binare: (1100) si (1101) se folosesc pentru codificarea semnelor "+" si "-", in timp ce tetradele binare (0000),….., (1001) se utilizeaza pentru codificarea cifrelor zecimale.

Pe un octet/byte, in bitii cei mai putin semnificativi, poate fi plasata o singura tetrada: BCD "*neimpachetat*". In cazul plasarii a doua tetrade BCD pe un singur octet, se spune ca BCD este "*impachetat*" .

Codurile BCD se utilizeaza in calculatoarele destinate calculelor comerciale, financiare, inlaturand necesitatea conversiilor zecimal-binar si binar-zecimal.

Un numar negativ in baza 10, cu mai multe ranguri, se poate reprezenta in BCD in complementul fata de 9 sau fata de 10.

In cazul numerelor $(+402)_{10}$ si $(-402)_{10}$ se pot da reprezentarile lor in complementul fata de 9 si fata de 10:

a) $\underline{0000}\ \underline{0100}\ \underline{0000}\ \underline{0010}$ $(+402)_{10}$

$(0)_{10}\ (4)_{10}\ (0)_{10}\ (2)_{10}$

b) $\underline{1001}\ \underline{0101}\ \underline{1001}\ \underline{0111}$ $(-402)_{10}$ complementul fata de 9

$(9)_{10}\ (5)_{10}\ (9)_{10}\ (7)_{10}$

c) $\underline{1001}\ \underline{0101}\ \underline{1001}\ \underline{1000}$ $(-402)_{10}$ complementul fata de 10

$(9)_{10}\ (5)_{10}\ (9)_{10}\ (8)_{10}$

In ultimul exemplu numerele pozitive vor fi reprezentate in gama 0 – 4999, iar numerele negative in gama 5000 – 9999.

6.5. Reprezentarea in virgula mobila.

Reprezentarea numerelor in virgula fixa precizeaza un numar de ranguri la stanga virgulei, pentru partea intreaga, si un alt numar de ranguri la dreapta virgulei, pentru partea subunitara.

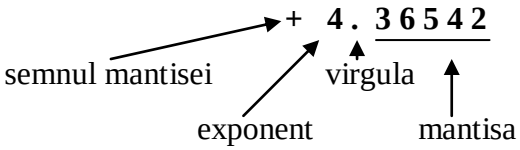
In vederea asigurarii unei game largi de reprezentare, cat si a unei precizii convenabile, in cazul virgulei fixe trebuie sa se aloce un numar mare de ranguri. Astfel, in cazul in care se doreste manipularea numerelor cu valori pana la 1 trilion (10^{12}) sunt necesare 40 de ranguri binare , intrucat $10^{12} \approx 2^{40}$. Acelasi numar de 40 ranguri binare este necesar in cazul asigurarii unei precizii de o trilionime. Numerele ar fi astfel reprezentate pe 80 de biti. In practica solicitarile privind gama si precizia pot fi si mai mari.

Virgula mobila ofera posibilitatea reprezentarii cu un numar mic de ranguri binare a unei game largi de numere exprimabile, prin efectuarea unui compromis intre numarul de ranguri care asigura precizia si numarul de ranguri care asigura gama.

In numarul de mai jos, reprezentat in virgula mobila:

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

gama este stabilita de catre numarul de ranguri ale exponentului si de catre baza, care este 10, in cazul de fata. Precizia este asociata cu numarul de ranguri ale partii subunitare, 5 in exemplul de mai sus. Precizia si gama impun un numar de 6 ranguri zecimale, la care se mai adauga unul pentru codificarea semnului partii subunitare/mantisei:



Pentru marirea gamei de reprezentare, in afara compromisului intre precizie si gama, bazat pe modificarea numarului de biti din reprezentarile exponentului si mantisei, se mai poate actiona si asupra bazei, in sensul cresterii acesteia. Prin aceasta creste precizia pentru numerele mici, iar pentru numerele mari scade.

Reprezentarea normalizata. Un numar poate fi reprezentat, in virgula mobila, in mai multe moduri:

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

Pentru a evita reprezentarile multiple ale aceluiasi numar se introduce *forma normalizata*. Aceasta se obtine prin deplasarea spre stanga a mantisei, astfel incat, imediat la dreapta virgulei sa se afle o cifra diferita de 0. Pe masura deplasarii mantisei la stanga se incrementeaza si exponentul. Operatia nu are sens atunci cand mantisa are toate rangurile egale cu 0.

De regula, un exponent este rezervat pentru reprezentarea lui zero si a altor cazuri speciale, cum ar fi ∞. Cu exceptia cazului cand este egala cu zero, mantisa poseda in bitul cel mai semnificativ o unitate, pentru o baza egala cu 2. Acest bit, egal cu 1, este prezent in mod implicit, ceea ce permite deplasarea mantisei spre stanga cu un bit, in scopul maririi preciziei. Bitul in cauza poarta numele de "*bit ascuns*".

Terminologia folosita in legatura cu erorile de calcul.

In analiza erorilor posibile privind prelucrarea datelor se folosesc o serie de termeni si concepte specifice.

Precizia. Constituie un termen asociat cu lungimea cuvantului, numarul de biti disponibili intr-un cuvant pentru reprezentarea unui numar dat. In cazul unui registru de 8 biti, considerand ca se reprezinta numai numere naturale, precizia de reprezentare va fi de 1/256. Precizia nu trebuie confundata cu acuratetea.

Acurateta. Aceasta reprezinta o masura a apropierii unei aproximatii fata de valoarea exacta. Acest termen nu trebuie confundat cu precizia. Ca exemplu se poate considera reprezentarea numarului natural 6 in binar, pe 4 biti. Reprezentarea exacta, fara nici o eroare va fi 0110. Daca se ia numarul binar fractionar 0,10101010101, care trebuie reprezentat pe 8 biti se va obtine: 0,1010101. Ultima forma constituie o reprezentare cu eroare a numarului initial. Astfel, in al doilea caz reprezentarea, care contine o eroare, este mai precisa (8 biti in loc de 4), dar are o acurateta mai mica .

Gama. Gama reprezinta multimea numerelor reprezentabile intr-un sistem dat. Astfel, in reprezentarea numerelor intregi in complementul fata de doi, pe patru biti, gama de reprezentare va fi de la -8 la + 7, adica (+7) - (-8) = 15.

Rezolutia. Aceasta constituie marimea diferentei/distantei intre doua numere sau cifre adiacente. Pentru reprezentarea cu patru cifre zecimale, gama fiind intre 0000 si 9999, rezolutia este constanta si egala cu 1. *In cazul reprezentarii in virgula mobila rezolutia nu mai este constanta in cadrul gamei*. Ea este dependenta de valoarea exponentului utilizat.

Trunchierea. Cunoscuta si sub denumirea de rotunjire prin lipsa, aceasta tehnica este utilizata in cazurile in care precizia nu este suficienta pentru reprezentarea corecta a numarului stocat. Considerand stocarea valorii lui *pi* = 3,141592654 intr-un dispozitiv capabil sa memoreze numai 6 cfre zecimale, numarul *pi* = 3,14159 se va reprezenta cu o eroare de trunchiere egala cu 0,000002654.

Rotunjirea. Metoda prin care se cauta sa se selecteze valoarea cea mai apropiata de valoarea initiala a numarului poarta numele de rotunjire. In zecimal, daca cifra aflata la dreapta ultimei cifre, care intra in reprezentare, este mai mare sau egala cu 5 atunci ultima cifra se incrementeaza cu 1, in caz contrar se lasa neschimbata. In binar, daca cifra aflata la dreapta ultimei cifre, care intra in reprezentare, este 1 atunci la ultima cifra se adauga o unitate, in caz contrar cifra nu se modifica.

Depasirea. Situatia de depasire apare cand rezultatul unui calcul este prea mare pentru a putea fi reprezentat in sistem. Spre exemplu, daca se lucreaza in binar cu numere intregi reprezentate in complementul fata de doi, pe 4 biti, gama de reprezentare fiind -8 la +7, orice rezultat care depaseste gama va conduce la depasire.

Depasirea superioara si depasirea inferioara. Aceste situatii apar la reprezentarea numerelor in virgula mobila, atunci cand rezultatul este mai mare sau mai mic decat cel mai mare sau cel mai mic numar care poate sa fie reprezentat in sistem. Pentru a preveni obtinerea unor rezultate false/eronate/pseudorezultate se elaboreaza tehnici prin care se semnalizeaza aparitia unor asemenea situatii.

Erori introduse la conversia numerelor din baza zece in baza doi. Aceste erori apar datorita faptului ca cele mai multe numere nu reprezinta multipli ai unor fractii binare. Situatia se intalneste in mod frecvent la introducerea datelor in calculator. De regula, numarul rangurilor binare este destul de mare pentru a se putea reprezenta numarul necesar de cifre zecimale.

Reprezentarea numerelor reale.

Calculatoarele destinate calculelor stiintifice si ingineresti opereaza in principal cu numere reale. Se cunoaste ca multimea numerelor reale este convexa, adica in oricare interval, indiferent de marimea sa, exista un numar infinit de valori reale. Intr-un calculator pot fi reprezentate precis valori discrete, dintr-o multime finita, deoarece fiecare informatie/data poate fi reprezentata numai printr-un numar fix de biti. Astfel, oricare reprezentare a unor numere reale, intr-un calculator, constituie o aproximare a valorilor exacte.

Un aspect important se refera la faptul ca datele reale presupun in acelasi timp, atat o gama mare de valori, cat si o anumita precizie.

In principiu numerele reale se pot reprezenta in doua moduri: in *virgula fixa* si in *virgula mobila*.

Reprezentarea in virgula fixa presupune un numar dat de ranguri pentru partea intreaga si pentru partea subunitara:

$$X = x_{n-1}\, x_{n-2} \ \ldots\ x_i \ldots\ x_1\, x_0\, ,\ x_{-1}\, x_{-2} \ \ldots\ x_{-i} \ldots\ x_{-m}$$

Desi prin aceasta reprezentare se poate asigura o precizie satisfacatoare, in multe cazuri

(in functie de numarul de ranguri $n + m$) apar dificultati importante in privinta gamei de reprezentare a datelor de intrare, a rezultatelor partiale si finale in cadrul unui program executat pe un calculator. Aceasta implica introducerea unor factori de scara, care sa asigure limitarea valorilor maxime la posibilitatile reale de reprezentare, in calculatorul dat.

Reprezentarea in virgula mobila sau notatia stiintifica foloseste in principal doua campuri: unul pentru *exponent* (*e*) si altul pentru *mantisa/parte subunitara/fractie* (*f*). Astfel, numarul real *X* se poate scrie:

$$X = f.r^{e}$$

unde *r* este *baza*, *f* - *mantisa*, iar *e* - *exponentul*.

Baza se ia egala cu 2 sau o putere a acestuia (64).

Mantisa f se reprezinta ca un numar subunitar, normalizat afectat de semn ($|f| \geq 1/2$) , pentru a nu pierde din precizie; numarul de biti din mantisa asigura precizia de reprezentare.

Exponentul stabileste gama de reprezentare; pentru simplificarea algoritmilor de efectuare a operatiilor aritmetice, in virgula mobila, se utilizeaza numai exponenti pozitivi (deplasati), prin adunarea unei cantitati date, astfel incat, cel mai mic exponent sa nu ia o valoare negativa (sa fie zero).

Daca *exponentul* are *k* biti si mantisa *m* biti (exclusiv semnul *s*), numarul *X* se va putea reprezenta astfel:

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

Semnul *s*, al mantisei, va fi plasat in rangul cel mai semnificativ al reprezentarii binare.

Stabilirea formatului de reprezentare a numerelor reale, in virgula mobila, a constituit obiectul a numeroase studii, care s-au concretizat prin elaborarea unui standard (IEEE 754). Astfel, au fost alese doua formate standard:

- *formatul scurt* sau *de baza*, pe un cuvant de 32 de biti, care asigura o viteza mai mare de operare;
- *formatul lung* sau *dublu de baza*, pe un cuvant de 64 de biti, care asigura o precizie mai mare de lucru.

In alegerea bazei s-a plecat de la observatia ca aceasta trebuie sa permita cea mai buna precizie posibila si pentru formatul scurt. S-a demonstrat ca o baza egala cu 2 asigura o precizie mai buna in cazul rotunjirii mantisei rezultatului unei operatii aritmetice, decat alte baze (4, 8, 16), cel putin cu un rang in raport cu baza 16.

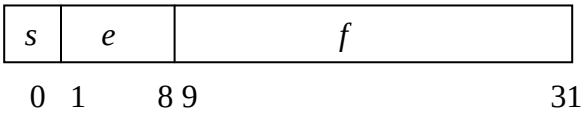
In privinta reprezentarii se considera ca folosirea codului direct (semn si modul) este superioara codului complementar.

Daca lungimea cuvantului este fixata, gama si precizia se influenteaza reciproc.

O gama de $10^{+/-75}$ este insuficienta, in timp ce o gama de $10^{+/-300}$ corespunde cerintelor celor mai multe aplicatii. De aceea, in cazul formatului scurt, a fost sacrificata gama, intr-o oarecare masura, pentru a asigura o buna precizie.

Daca se foloseste aceeasi unitate aritmetica, atat pentru operatiile in formatul scurt, cat si pentru operatiile in formatul lung, in primul caz nu se vor inregistra depasiri de tip superior sau inferior. De asemenea, se poate observa ca numarul de biti (52) din mantisa in formatul lung este mai mare decat dublul numarului de biti (2×23) ai mantisei in formatul scurt. Aceasta face ca la inmultirea in formatul scurt adunarea produselor partiale sa nu fie afectata de erori.

Formatul scurt (*simplu - de baza*).



- bitul 0: *s* = *semnul mantisei* (*s* = 1 - mantisa negativa);
- bitii 1-8: *e* = *exponentul deplasat* (deplasarea este 2^7 - 1);
- bitii 9-31: *f* = *mantisa/fractia* (cand e $\neq$ 0, se presupune un

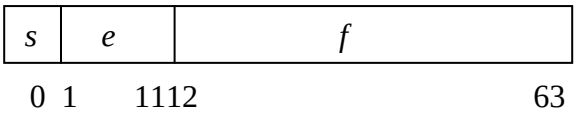
bit egal cu 1, la stanga lui *f*; virgula se plaseaza intre acest bit si primul bit explicit al mantisei;

valoare: numarul reprezentat intr-un format scurt este:

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

Valoarea/marimea numerelor reprezentate este in gama: -2^{-126} .(1,0) pana la 2^{127} .($2 - 2^{-23}$), ceea ce reprezinta aproximativ $-1,8 \times 10^{-38}$ pana la $3,40 \times 10^{38}$

Formatul lung (*dublu - de baza*).



- bitul 0: *s* = *semnul mantisei* (*s* = 1 - mantisa negativa);
- bitii 1-11: *e* = *exponentul deplasat* (deplasarea este 2^{10} - 1);
- bitii 12-63: *f* = *mantisa/fractia* (cand e $\neq$ 0, se presupune un

bit egal cu 1, la stanga lui *f*; virgula se plaseaza intre acest bit si primul bit explicit al mantisei;

valoare: numarul reprezentat intr-un format lung este:

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

Observatii privind formatele.

1. Baza a fost aleasa din considerente de echilibrare a gamei, cu restrictia ca toate numerele mici sa aibe valori reciproce reprezentabile. Contrar altor practici, gama obtinuta este deplasata pentru a putea asigura depasirea inferioara (underflow), deoarece aceasta situatie se poate rezolva mai usor, prin hardware, decat depasirea superioara (overflow)
2. S-a constatat ca prezenta unui prim bit implicit, inaintea bitilor mantisei, asigura depasirea inferioara gradata. Mai jos se va arata cum au fost rezolvate aspectele mentionate la punctele 1 si 2.
3. Cazuri speciale.

   3.1 In conditiile unui exponent egal cu zero s-a rezervat un camp pentru:

   - *zero*: toti bitii sunt zero;
   - *date initializate*: bitul de semn este unu, iar ceilalti biti sunt zero;
   - *numere denormalizate*: toate formatele de biti cu exponent zero si mantisa diferita de zero se interpreteaza ca numere denormalizate; in efectuarea operatiilor aritmetice bitul cel mai semnificativ este fortat la zero iar bitul implicit unu este adunat la exponent, ceea ce va permite implementarea gradata a depasirii inferioare.

     Valoarea denormalizata va fi: $(-1)^s 2^{1-(2\uparrow 7- 1)}.(0 + f)$

   3.2 In conditiile unui exponent cu toti bitii egali cu unu s-a rezervat un camp pentru:

   - $+\infty$: semnul este zero, iar celalti biti sunt unu;

- -∞: toti bitii sunt unu:
- nedefinit: semnul si mantisa sunt zero, iar exponentul este format din unitati;
- alte situatii: celelalte combinatii sunt rezervate pentru utilizari inca nedefinite.

Standardul IEEE 754 pentru reprezentarea numerelor in virgula mobila.

Acest standard a fost acceptat de numerosi producatori de microprocesoare si de unitati centrale pentru minicalculatoare si calculatoare de capacitate medie-mare.

In aceste conditii se vor putea transporta, fara dificultati majore, pachetele de programe stiintifice intre diferite tipuri de echipamente de calcul.

A. Formate.

Formatele prezentate mai sus fac parte din standardul IEEE 754. In plus, standardul mai contine formatul extins cu *formele scurta* (*simpla)* si *lunga* (*dubla*), in care mantisa este prevazuta cu un bit *j*, pentru specificarea explicita a partii intregi, care in formatul de baza discutat a fost presupus implicit. In acest format se dau alte valori minime si maxime pentru exponenti si mantise, in raport cu formatul de baza.

In cele ce urmeaza se prezinta in rezumat caracteristicele tuturor formatelor.

Formatul simplu - de baza.

Acest format este organizat pe 32 de biti: 1 bit pentru semnul mantisei, 8 biti pentru exponent si 23 de biti pentru mantisa.

Valoarea *V* a numarului se calculeaza astfel:

- daca $e = 255$ si $f \neq 0$, atunci V = NaN (nu este un numar -Not a Number - simbol codificat in format);
- daca $e = 255$ si $f = 0$, atunci $V = (-1)^s.\infty$
- daca $0 < e < 255$, atunci $V = (-1)^s.2^{e-127}.(1,f)$;
- daca $e = 0$ si $f \neq 0$, atunci $V = (-1)^s.2^{-126}.(0,f)$;
- daca $e = 0$ si $f = 0$, atunci $V = (-1)^s.0$ (zero).

Formatul dublu - de baza.

Acest format este organizat pe 64 de biti: 1 bit pentru semnul mantisei, 11 biti pentru exponent si 52 de biti pentru mantisa.

Valoarea *V* a numarului se determina dupa cum urmeaza:

- daca $e = 2047$ si $f \neq 0$, atunci V = NaN (nu este un numar -Not a Number - simbol codificat in format);
- daca $e = 2047$ si $f = 0$, atunci $V = (-1)^s.\infty$
- daca $0 < e < 2047$ atunci $V = (-1)^s.2^{e-1023}.(1,f)$;
- daca $e = 0$ si $f \neq 0$, atunci $V = (-1)^s.2^{-1022}.(0,f)$;
- daca $e = 0$ si $f = 0$, atunci $V = (-1)^s.0$ (zero).

***

Formatul simplu - extins.

Acest format este structurat astfel: 1 bit de semn, 1 bit pentru partea intreaga (*j*) a mantisei, cel putin 31 de biti pentru partea fractionara a mantisei (*f*) si un exponent ce poate lua valori cuprinse intre minimum $m = -1022$ si maximum $M = 1023$.

Valoarea *V* a numarului se determina dupa cum urmeaza:

- daca $e = M$ si $f \neq 0$, atunci V = NaN;
- daca $e = M$ si $f = 0$, atunci $V = (-1)^s.\infty$;
- daca $m < e < M$, atunci $V = (-1)^s.2^e.(j,f)$;
- daca $e = m$ si $j = f = 0$, atunci $V = (-1)^s.0$ (zero);
- daca $e = m$ si $j \neq 0$ sau $f \neq 0$, atunci $V = (-1)^s.2^{e^*}.(j,f)$;

unde e* este egal cu *m* sau $m + 1$, selectat la implementare.

Formatul dublu extins.

Acest format are caracteristici identice cu formatul simplu extins, cu exceptia ca exponentul ia valori intre $m = -16382$ si $M = 16383$, avand cel putin 63 de biti pentru partea fractionara (*f*).

B.Rotunjirea.

Operatia de rotunjire trateaza numarul ca avand o precizie infinita si il modifica pentru a corespunde formatului destinatie.

Schema hardware va emite un semnal pentru a arata daca rezultatul este corect sau incorect.

Standardul IEEE 754 include atat un mecanism standard-implicit de rotunjire, cat si alte trei metode ce pot fi selectate de utilizator.

Mecanismul standard rotunjeste numarul la cea mai apropiata valoare reprezentabila. Daca sunt posibile doua valori, formatul este rotunjit la valoarea para, care face ca eroarea de rotunjire sa fie egala cu jumatate din valoarea celui mai putin semnificativ bit.

Celelalte trei mecanisme de rotunjire selectabile sunt urmatoarele:

- rotunjirea catre $+\infty$ asigura cea mai apropiata valoare dar nu mai mica decat a numarului dat;
- rotunjirea catre $-\infty$ furnizeaza cea mai apropiata valoare, dar nu mai mare decat numarul dat;
- rotunjirea catre 0 (trunchiere) asigura valoarea cea mai apropiata, dar nu mai mare decat numarul dat in modul.

Operatia de rotunjire se aplica formatului destinatie. In formatul dublu sau formatul extins standardul permite utilizatorului sa specifice daca rotunjirea trebuie sa se efectueze in formatul simplu - de baza. Aceasta face ca pe o masina, care are numai formatul dublu de baza sau formatul extins, sa se emuleze si formatul simplu de baza.

C. Valori speciale.

Standardul de reprezentare in virgula mobila suporta aritmetica cu numere infinite folosind *modalitatile afina* si *proiectiva*, selectabile de utilizator.

Modalitatea infinit-afina este definita prin relatia:

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

Modalitatea infinit-proiectiva compara intotdeauna numerele egale, indiferent de semn. Standardul defineste un set de operatii astfel incat folosirea unui operand infinit nu va conduce la un rezultat eronat; pentru detectarea unor asemenea cazuri nu sunt prevazute capcane. In calculele cu valori infinite toate exceptiile, care se vor discuta in continuare, raman valabile.

NaN constituie o valoare speciala, introdusa pentru a semnaliza operatii invalide sau operatii care produc rezultate invalide cu o valoare speciala. Standardul defineste posibilitatile de utilizare si neutilizare ale capcanelor pentru NaN. Folosirea capcanei asigura activarea unei exceptii, la efectuarea unei operatii cu un operand NaN. Neutilizarea capcanei conduce numai la pozitionarea indicatorilor corespunzatori de eroare (conditii).

D. Operatii.

In standard sunt definite operatiile aritmetice de baza: adunarea, scaderea, inmultirea si impartirea; radacina patrata si gasirea restului la impartire; conversia formatului in: virgula mobila, numere intregi si numere zecimale codificate binar BCD (cu exceptia numerelor BCD extinse EBCD); compararea numerelor in virgula mobila.

Operatia de comparare conduce la unele situatii particulare, cand se folosesc operanzi NaN sau cu valori infinite. Relatiile permise sunt: "mai mic decat", "egal cu", "mai mare decat" si "neordonat". Ultimul caz apare cand cel putin un operand este NaN si cand un numar infinit este comparat in modalitatea proiectiva cu un numar finit. Relatia "neordonat" afirma predicatele "neordonat" si "< >", negand insa pe toate celelalte. Atunci cand se testeaza daca un numar are valoarea NaN, nu va fi posibila verificarea intre o constanta NaN si un numar, deoarece relatia este intotdeauna "neordonat".

E. Exceptii si capcane.

Standardul IEEE754 defineste atat exceptiile, care trebuie detectate, cat si informatiile necesare elementului, care manipuleza capcana, pentru gasirea exceptiilor. Implementarea va asigura cate un indicator pentru fiecare tip de exceptie. Raspunsul standard la exceptie consta in continuarea operatiei, fara activarea capcanei.

Operatie invalida. Acest tip de exceptie se incadreaza in *doua clase*: exceptii de *operand invalid* si *rezultat invalid*. In ambele cazuri, daca nu este activata o capcana, rezultatul va fi NaN.

Exceptiile de operand sunt provocate de urmatoarele evenimente:

- un operand este NaN, fara a se genera un semnal pentru capcana si fara a se activa capcana;
- se executa una din urmatoarele operatii:
  - $(\infty) + (\infty)$ - (modalitatea proiectiva),
  - $(\infty) + (-\infty)$ - (modalitatea afina),
  - $(\infty)/(\infty)$, $0/0$ si $0 \times \infty$;
- in operatia *x* REM *y*, de aflare a restului la impartire, fie *x* este infinit, fie *y* este zero;
- cand se incearca extragerea radacinii patrate dintr-un numar negativ;

- la conversia de la formatul in virgula mobila la intreg sau BCD, cand conversia corecta conduce la o depasire, la o valoare infinita sau NaN;
- la comparatiile ce folosesc predicatele <, > sau negatiile lor, cand relatia este de "neordonare" si cand nici un operand nu este"neordonat".

Exceptiile de rezultat invalid apar atunci cand rezultatul unei operatii nu este corect in raport cu formatul destinatie.

Alte exceptii. Exceptiile standard sunt urmatoarele:

- impartire cu zero;
- depasire superioara;
- depasire inferioara;
- rezultat inexact, atunci cand nu se genereaza alte exceptii si cand valoarea rotunjita pentru rezultat conduce la depasire superioara, fara a activa capcana.

Parametrii pentru capcana. Capcanele, care corespund fiecarei exceptii pot fi activate/dezactivate de catre utilizator. Cand este activata, exceptia transfera controlul la o rutina pentru manipularea capcanei (furnizata de utilizator sau de catre sistem).

O asemenea rutina trebuie sa primeasca urmatoarele informatii:

- tipul exceptiei, care a aparut,
- tipul operatiei, care s-a executat,
- formatul destinatiei,
- rezultatul corect rotunjit (in cazul depasirii superioare/inferioare, rezultatului inexact/invalid), pe langa alte informatii referitoare la faptul ca rezultatul nu corespunde formatului destinatie,
- valorile operandului, in cazurile impartirii cu zero si exceptiilor de operand invalid.

Standardul aritmetic.

Scopul urmarit prin aritmetica in virgula mobila este acela de a pune la dispozitia utilizatorului un sistem convenabil de reprezentare a numerelor prin care se obtin rezultate precise ca urmare a efectuarii operatiilor aritmetice. D. Knuth a sugerat un model de aritmetica in virgula mobila, care permite obtinerea unor rezultate in virgula mobila foarte apropiate de rezultatul corect.

Algoritmii propusi de Knuth conduc la implementari extrem de costisitoare si nu ofera solutii pentru cazul rezultatelor aflate exact intre doua numere in virgula mobila.

In continuare se prezinta atat un set de reguli, care specifica toate cazurile, cat si algoritmii de implementare ai acestui set de reguli.

Reguli.

1. Multimea numerelor in virgula mobila trebuie sa contina 0 si 1 (identitati aditive si multiplicative), iar daca *x* apartine multimii atunci si *-x* va apartine multimii.
2. Daca rezultatul corect al unei operatii in virgula mobila este un numar in virgula mobila, atunci acel numar trebuie sa fie generat de operatia respectiva, in caz contrar rezultatul va fi rotunjit conform regulei 3.
3. Daca rezultatul corect se afla exact la mijlocul intervalului dintre doua numere in virgula mobila, atunci aritmetica respectiva trebuie sa genereze numarul cu mantisa para.

Regulile de mai sus (regula de rotunjire poarta numele de "rotunjire la par") asigura precizia maxima.

In continuare vor fi ilustrate metodele de implementare ale operatiilor aritmetice in virgula mobila (adunare, scadere, inmultire si impartire), care utilizeaza "rotunjirea la par", in vederea obtinerii rezultatului final.

Pentru exemplificare se va considera un acumulator cu urmatoarea structura, pentru efectuarea operatiilor aritmetice cu numere avand mantisa de patru biti:

D	F1	F2	F3	F4	G	R	ST

cu urmatoarele notatii:

- D: bitul de depasire,
- F1 - F4: cei 4 biti ai mantisei,
- G: Bitul de garda,
- R: bitul de rotunjire,
- ST: bitul de legatura/lipitura, "stiky".

Bitul ST este pozitionat in unu, daca in procesul de denormalizare are loc transferul unei unitati din bitul R spre dreapta. Daca ST este pozitionat in unu el va ramane in aceasta stare pe toata durata operatiilor. Toate deplasarile in acumulator implica bitii D, G, R si ST. Bitul ST nu este modificat de deplasarea la stanga. Deplasarea la dreapta introduce zero in bitul D.

In algoritmii de mai jos se considera ca acumulatorul este initializat la zero si ca operanzii au fost verificati daca sunt invalizi sau egali cu zero. Astfel, se considera numai operanzii valizi, normalizati, diferiti de zero. De asemenea, se considera ca operatiile aritmetice, privind exponentii pentru inmultire, imparire si ajustare prin deplasare, sunt evidente, fara a necesita detalieri, si ca semnul rezultatului este stabilit in mod corespunzator.

Adunarea si scaderea in virgula mobila vor fi tratate impreuna, evidentiind doua cazuri:

- *adunarea modulelor*, atunci cand se aduna numere cu acelasi semn sau cand se scad numere cu semne diferite;
- *scaderea modulelor*, atunci cand se scad numere cu acelasi semn sau se aduna numere cu semne diferite.

1. Adunarea modulelor.

1.1. *Denormalizarea*: numarul cu exponent mai mic se incarca in acumulator si se deplaseaza cu un numar de pozitii spre dreapta, egal cu diferenta exponentilor. Daca exponentii sunt egali oricare dintre numere/operanzi poate fi incarcat in acumulator, fara a se efectua vreo deplasare.

1.2. *Adunarea:* cel de-al doilea operand este adunat la acumulator.

1.3. *Normalizarea:* daca s-a pozitionat in unu bitul D atunci se deplaseaza continutul acumulatorului cu un bit spre dreapta.

1.4. *Rotunjirea*: se aduna 1 la pozitia G, dupa care, daca G = R = ST = 0, se forteaza F4 $\leftarrow$0.

1.5. *Renormalizarea*: daca D este pozitionat in unu se face deplasarea continutului acumulatorului spre dreapta.

1.6 .*Depasirea*: se verifica daca a avut loc o depasire a exponenului.

2. Scadera modulelor.

2.1.*Denormalizarea*: se incarca numarul cu modulul cel mai mic in acumulator si se deplaseaza la dreapta, daca este necesar. Rezultatul va fi zero daca si numai daca operanzii sunt egali, abandonadu-se operatiile urmatoare.

2.2. *Scadearea*: se scade continutul acumulatorului din celalalt operand, pastrand rezultatul in acumulator (daca ST = 1 se va genera un imprumut);

2.3. *Normalizarea*: se deplaseaza acumulatorul la stanga pana cand F1 devine egal cu 1;

2.4. *Rotunjirea*: se aduna 1 la bitul G si apoi, daca G = R = 0 si ST = 0, se forteaza F4 $\leftarrow$ 0; nu este necesara rotunjirea daca pentru normalizare au fost necesare mai multe deplasari la stanga;

2.5. *Renormalizarea*: in cazul in care rotunjirea a condus la depasire, se efectueaza o deplasare spre dreapta.

Inmultirea.

In algoritmul pentru inmultirea in virgula mobila se considera prezente resursele hardware necesare pentru a forma produsul a doua numere in lungime dubla. In caz contrar se poate folosi algoritmul "aduna si deplaseaza la dreapta" in acumulator, pentru a forma produsul, incepand cu rangul inferior. In continuare partea mai putin semnificativa este pierduta pe masura ce este deplasata in afara bitului ST.

1. *Inmultirea*: se formeaza produsul in lungime dubla;
2. *Normalizarea*: pentru o eventuala normalizare a produsului se face o deplasare cu un bit;
3. *Pozitionarea bitilor* G, R, ST: fie produsul normalizat in lungime dubla:

   F1 F2 F3 F4 F5 F6 F7 F8

   atunci G = F5, R = F6, ST = F7 $\cup$ F8.
4. *Rotunjirea*: se realizeaza dupa cum s-a aratat mai sus;
5. *Renormalizarea*: se realizeaza dupa cum s-a aratat mai sus;
6. *Erori:* se verifica depasirea superioara/inferioara a expnentului.

Impartirea.

Se considera ca, in procesul efectuarii impartirii, restul este disponibil in orice moment.

1. *Impartirea*: se formeaza primii sase biti ai catului normalizat:

   F1 F2 F3 F4 F5 F6
2. *Pozitionarea bitilor* G, R, ST: se forteaza:

   G = F5, R = F6 si ST = rest;
3. *Rotunjirea*: se efectueaza dupa regulile mentionate mai sus;
4. *Renormalizarea*: se efectueaza dupa regulile mentionate mai sus.

Prezenta bitului ST permite implementarea rotunjirii directionate, adica la dreapta sau la stanga pe un numar standard de biti.

In algoritmii de mai sus s-a cautat obtinerea unei precizii maxime cu pretul reducerii vitezei, intr-o oarecare masura.

Erori in reprezentarea numerelor in virgula mobila.

Precizia finita introduce erori, care pot sau nu pot sa fie acceptabile pentru aplicatia data. Spre exemplu, se considera reprezentarea unui milion, in virgula mobila, dupa care se scade 1, din aceasta. In cazul in care eroarea este mai mare decat 1, restul va fi tot de 1 milion.

Pentru a caracteriza *eroarea*, *gama* si *precizia*, se vor face urmatoarele notatii:

b – baza;

f - numarul rangurilor semnificative ale mantisei/fractiei, in baza data;

M - cel mai mare exponent;

m - cel mai mic exponent.

Numarul rangurilor semnificative din fractie este diferit de numarul de biti in cazul in care baza este diferita de 2. De exemplu, o baza egala cu 16 foloseste 4 biti pentru fiecare rang. Daca baza este de forma 2^k (o putere a lui 2), atunci sunt necesari k biti pentru fiecare rang exprimat in baza data. Utilizarea lui 1 ascuns mareste f cu 1 bit, chiar daca nu conduce la cresterea numarului de numere reprezentabile.

In cadrul analizei reprezentarii in virgula mobila intereseaza urmatoarele aspecte:

- *numarul numerelor reprezentabile*;
- *numarul care are valoarea/marimea cea mai mare*;
- *numarul diferit de zero, care are valoare/marimea cea mai mica*;
- *dimensiunea celei mai mari distante intre doua numere succesive*;
- *dimensiunea celei mai mici distante intre doua numere succesive*.

Numarul numerelor reprezentabile se poate calcula luand in considerare numarul de valori pe care le pot lua campurile distincte ale reprezentarii in virgula mobila:

- semnul poate lua doua valori;
- numarul de exponenti este dat de expresia: $((M - m) + 1)$;
- primul rang al fractiei: b-1;
- celelalte ranguri ale fractiei: b^{f-1};
- zero.

Astfel, *pentru numarul numerelor/valorilor reprezentabile normalizate*, se obtine urmatoarea expresie:

$$2 \times ((M - m) + 1) \times (b\text{-}1) \times b^{f-1} + 1$$

Trebuie aratat ca, in cazul exponentului, nu toate combinatiile de biti sunt valide. Spre exemplu, in standardul IEEE 754, desi sunt rezervati 8 biti pentru exponent, cel mai mic exponent este -126 si nu –128. Exponentii interzisi sunt folositi pentru reprezentari speciale: zero si infinit.

Se vor considera, in continuare, *numarul care are valoarea/marimea cea mai mare si numarul diferit de zero, care are valoare/marimea cea mai mica*;

Numarul cu marimea cea mai mica are, de asemenea, cel mai mic exponent b^m si cea mai mica fractie normalizata (diferita de zero), care va fi egala cu b^{-1}. Astfel, *numarul cel mai mic va fi*: b^{m-1}.

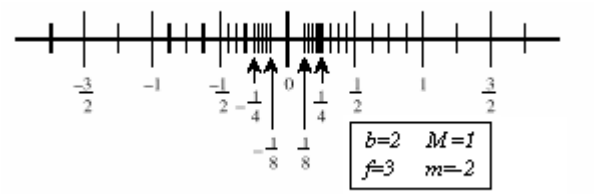
In mod similar *numarul cel mai mare* va avea cel mai mare exponent b^Msi cea mai mare fractie $(1 - b^{-f})$, ceea ce conduce la valoarea $b^M . (1 - b^{-f})$.

Cea mai mica distanta si cea mai mare distanta intre doua numere consecutive sunt calculate intr-o maniera similara.

Cea mai mica distanta apare atunci cand exponentul are cea mai mica valoare b^m si cand se modifica cel mai putin bit semnificativ al fractiei, b^{-f}. Aceasta conduce la *o valoare minima a distantei* egala cu b^{m-f}.

Cea mai mare distanta apare atunci cand exponentul are cea mai mare valoare b^M si cand se modifica cel mai putin bit semnificativ al fractiei,b^{-f}. Aceasta conduce la *o valoare maxima a distantei* egala cu b^{M-f}

Se va considera un exemplu, de reprezentare in virgula mobila, in care exista un bit de semn, un exponent pe 2 biti in exces 2, o fractie pe 3 biti normalizat, fara 1 ascuns. Reprezentarea lui zero este 000000. In figura de mai jos este data reprezentarea in acest format:



Observatii:

- au aparut distante mari intre zero si primele numere reprezentabile, deoarece reprezentarea normalizata nu permite structuri de biti in mantisa, care sa corespunda acestor distante;

- cel mai mic exponent este –2 si cea mai mica fractie normalizata este $(0,100)_2$, ceea ce face ca modulul celui mai mic numar reprezentabil, diferit de zero, sa fie 1/8.
- cel mai mare exponent este 2^1, iar cea mai mare fractie este $(1 - 2^{-3})$, ceea ce conduce la o valoare de 7/4 pentru cel mai mare numar reprezentabil;
- cel mai mic exponent este –2, iar valoarea celui mai putin semnificativ bit al fractiei este 2^{-3}, conducand la o valoare minima a distantei intre doua numere consecutive egala cu 2^{-5}, adica 1/32;
- cel mai mare exponent este 1, iar valoarea celui mai putin semnificativ bit al fractiei este 2^{-3}, conducand la o valoare maxima a distantei intre doua numere consecutive egala cu 2^{-2}, adica 1/4;
- datorita normalizarii numarul reprezentarilor valide este mai mic decat numarul reprezentarilor posibile;
- numarul reprezentarilor valide este dat de expresia:
  $2 \times ((M - m) + 1) \times (b\text{-}1) \times b^{f-1} + 1 = 2 \times ((1 - (-2) + 1) \times (2\text{-}1) \times 2^{3-1} + 1 = 33$.
- erorile relative sunt aproximativ aceleasi, comparand rapoartele intre distantele minima/maxima, intre doua numere consecutive, si valorile maxime ale celor mai mici/mari numere reprezentabile:

distanta $\nearrow$ $(b^{m-f}) / (\underbrace{b^m \times (1 - b^{-f})}) = 1/(b^f - 1)$, pentru numerele mici
($\nwarrow$ valoarea maxima pentru numarul cu exponent minim)

distanta $\nearrow$ $(b^{M-f}) / (\underbrace{b^M \times (1 - b^{-f})}) = 1/(b^f - 1)$, pentru numerele mari
($\nwarrow$ cel mai mare numar)

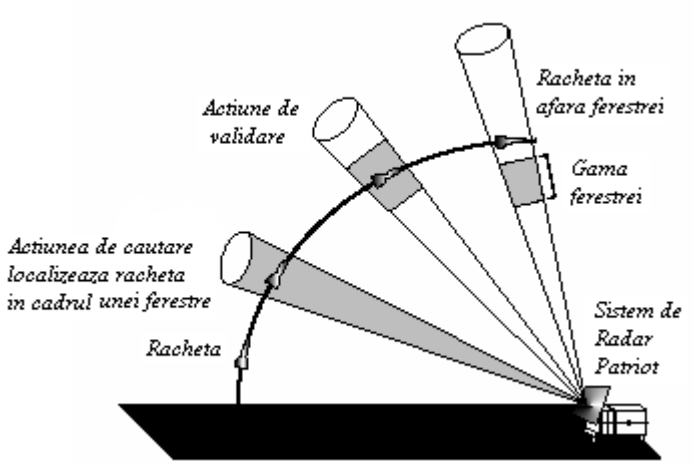
Observatie: Standardul IEEE 754 utilizeaza numere denormalizate pentru a umple distanta intre numerele plasate in apropierea zeroului.

Exemplu. Sa se reprezinte in virgula fixa numarul $(0,9375)_{10}$. Convertit in binar, in virgula fixa, acest numar va avea urmatoarea forma: $(0,0011)_2$. Valoarea sa exprimata in virgula mobila va fi: $1,1 \times 2^{-4}$.

Studiu de caz privind pierderea de precizie la conversia de la intregi la virgula mobila.

In timpul primului razboi din Golf, 1991-1992, s-au utilizat baterii de rachete Patriot, pentru urmarirea si distrugerea rachetelor irakiene Scud. Bateriile, prevazute cu radare si tehnica de calcul operau numai cateva ore, pentru a se evita detectarea lor. In 5 februarie, 1991, baracile armatei SUA, de la Dhahran, au fost atacate cu rachete, inregistrandu-se 28 de morti, datorita unor erori in partea de prelucrare a datelor de la sistemele radar.

Tinta este urmarita de catre radar, stabilindu-se, pe baza parametrilor calculati, pozitia viitoare a acesteia. Astfel, se genereaza o "fereastra" in care se considera a fi plasata tinta, ceea ce va permite eliminarea altor zgomote, din afara "ferestrei".



Predictia urmatoarei pozitii a rachetei Scud se calcula pe baza vitezei acesteia. Viteza este determinata prin modificarea pozitiei in raport cu timpul. Timpul in sistemul Patriot este actualizat de catre ceasul intern la intervale de 100 ms. Viteza este reprezentata in virgula mobila pe 24 de biti, iar timpul ca un intreg pe 24 de biti. Pentru a calcula noua pozitie, timpul si viteza trebuie sa fie reprezentate in virgula mobila, pe 24 de biti. Conversia de la intreg la virgula mobila, pentru timp, s-a facut cu erori, care cresc proportional cu timpul de operare al sistemului. Astfel, fereastra pentru noua pozitie a fost calculata eronat, eroarea depinzand de viteza rachetei si de durata de timp de cand sistemul era operational. In acea zi Patriot opera de peste 100 de ore, eroarea stabilirii ferestrei fiind de 687 metri, ceea ce a condus la pierderea tintei. Situatia fusese semnalata cu doua saptamani anterior de catre israelieni. Software-ul a fost modificat, a fost transmis pe campul de lupta, dar nu fusese instalat. O solutie simpla, pe

moment, ar fi constat in “reboot-area” a sistemului, la intervale de cateva ore, pentru eliminarea erorilor acumulate la conversia timpului.

Coduri alfanumerice

Spre deosebire de numerele reale, care au o gama infinita, numarul caracterelor alfanumerice este limitat, ceea ce permite ca un intreg set de caractere sa fie codificat cu un numar redus de biti pe caracter. In practica se utilizeaza trei sisteme de codificarea a caracterelor alfa numerice: ASCII (American Standard Code for Information Interchange), EBCDIC (Extended Binary Coded Decimal Interchange Code) si Unicode

Codul ASCII.

Acest cod, folosit pentru reprezentarea carcterelor alfanumerice, utilizeaza 7 biti pe caracter. Toate cele 2^7 coduri posibile reprezentand caractere valide. In tabela de mai jos se prezinta in hexazecimal codurile ASCII alfanumerice.

00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	´	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

NUL	Null	FF	Form feed	CAN	Cancel
SOH	Start of heading	CR	Carriage return	EM	End of medium
STX	Start of text	SO	Shift out	SUB	Substitute
ETX	End of text	SI	Shift in	ESC	Escape
EOT	End of transmission	DLE	Data link escape	FS	File separator
ENQ	Enquiry	DC1	Device control 1	GS	Group separator
ACK	Acknowledge	DC2	Device control 2	RS	Record separator
BEL	Bell	DC3	Device control 3	US	Unit separator
BS	Backspace	DC4	Device control 4	SP	Space
HT	Horizontal tab	NAK	Negative acknowledge	DEL	Delete
LF	Line feed	SYN	Synchronous idle		
VT	Vertical tab	ETB	End of transmission block		

Caracterele din pozitiile 00 – 1F si 7F, sunt caractere de control utilizate pentru: transmisie, controlul tiparirii, cat si in alte scopuri. Toate celelalte caractere sunt afisabile si includ: literele, cifrele, semnele de punctuatie si spatiul. Cifrele 0 – 9 apar in secventa ca, de altfel, si literele mici si mari, ceea ce simplifica manipularea caracterelor. Pentru a transforma reprezentarea

alfanumerica a unei cifre zecimale in valoarea sa numerica este suficient sa se scada $(30)_{16}$ din ea. Pentru a transforma codul unei majuscule in codul corespunzator al literei mici se va aduna la primul $(20)_{16}$.

Codul EBCDIC.

Codul ASCII permite reprezentarea a 128 de caractere, ceea ce constituie o limitare pentru cele mai multe tastaturi moderne. Aceste tastaturi contin, pe langa tastele alfanumerice, si taste dedicate unor caractere speciale. Codul EBCDIC este un cod pe 8 biti utilizat de catre IBM. In tabelul de mai jos se prezinta, in codificare hexazecimala, codurile EBCDIC.

00	NUL	20	DS	40	SP	60	–	80		A0		C0	{	E0	\
01	SOH	21	SOS	41		61	/	81	a	A1	~	C1	A	E1	
02	STX	22	FS	42		62		82	b	A2	s	C2	B	E2	S
03	ETX	23		43		63		83	c	A3	t	C3	C	E3	T
04	PF	24	BYP	44		64		84	d	A4	u	C4	D	E4	U
05	HT	25	LF	45		65		85	e	A5	v	C5	E	E5	V
06	LC	26	ETB	46		66		86	f	A6	w	C6	F	E6	W
07	DEL	27	ESC	47		67		87	g	A7	x	C7	G	E7	X
08		28		48		68		88	h	A8	y	C8	H	E8	Y
09		29		49		69		89	i	A9	z	C9	I	E9	Z
0A	SMM	2A	SM	4A	¢	6A	`	8A		AA		CA		EA	
0B	VT	2B	CU2	4B	.	6B	,	8B		AB		CB		EB	
0C	FF	2C		4C	<	6C	%	8C		AC		CC		EC	
0D	CR	2D	ENQ	4D	(	6D	_	8D		AD		CD		ED	
0E	SO	2E	ACK	4E	+	6E	>	8E		AE		CE		EE	
0F	SI	2F	BEL	4F	|	6F	?	8F		AF		CF		EF	
10	DLE	30		50	&	70		90		B0		D0	}	F0	0
11	DC1	31		51		71		91	j	B1		D1	J	F1	1
12	DC2	32	SYN	52		72		92	k	B2		D2	K	F2	2
13	TM	33		53		73		93	l	B3		D3	L	F3	3
14	RES	34	PN	54		74		94	m	B4		D4	M	F4	4
15	NL	35	RS	55		75		95	n	B5		D5	N	F5	5
16	BS	36	UC	56		76		96	o	B6		D6	O	F6	6
17	IL	37	EOT	57		77		97	p	B7		D7	P	F7	7
18	CAN	38		58		78		98	q	B8		D8	Q	F8	8
19	EM	39		59		79		99	r	B9		D9	R	F9	9
1A	CC	3A		5A	!	7A	:	9A		BA		DA		FA	|
1B	CU1	3B	CU3	5B	$	7B	#	9B		BB		DB		FB	
1C	IFS	3C	DC4	5C	*	7C	@	9C		BC		DC		FC	
1D	IGS	3D	NAK	5D	)	7D	'	9D		BD		DD		FD	
1E	IRS	3E		5E	;	7E	=	9E		BE		DE		FE	
1F	IUS	3F	SUB	5F	¬	7F	"	9F		BF		DF		FF	

STX	Start of text	RS	Reader Stop	DC1	Device Control 1	BEL	Bell
DLE	Data Link Escape	PF	Punch Off	DC2	Device Control 2	SP	Space
BS	Backspace	DS	Digit Select	DC4	Device Control 4	IL	Idle
ACK	Acknowledge	PN	Punch On	CU1	Customer Use 1	NUL	Null
SOH	Start of Heading	SM	Set Mode	CU2	Customer Use 2		
ENQ	Enquiry	LC	Lower Case	CU3	Customer Use 3		
ESC	Escape	CC	Cursor Control	SYN	Synchronous Idle		
BYP	Bypass	CR	Carriage Return	IFS	Interchange File Separator		
CAN	Cancel	EM	End of Medium	EOT	End of Transmission		
RES	Restore	FF	Form Feed	ETB	End of Transmission Block		
SI	Shift In	TM	Tape Mark	NAK	Negative Acknowledge		
SO	Shift Out	UC	Upper Case	SMM	Start of Manual Message		
DEL	Delete	FS	Field Separator	SOS	Start of Significance		
SUB	Substitute	HT	Horizontal Tab	IGS	Interchange Group Separator		
NL	New Line	VT	Vertical Tab	IRS	Interchange Record Separator		
LF	Line Feed	UC	Upper Case	IUS	Interchange Unit Separator		

Adesea codul ASCII, pe 7 biti, este reprezentat pe un octet, avand 0 sau 1 in bitul cel mai semnificativ, care este folosit drept bit de paritate. Aceasta nu este de natura sa favorizeze codul

ASCII in raport cu codul EBCDIC. In schimb, la transmisia seriala a datelor codurile pe 8 biti necesita mai mult timp decat cele pe 7 biti.

Unicode.

Unicode reprezinta un standard (ISO/IEC 10646) pentru reprezentarea caracterelor cu ajutorul unor cuvinte de 16 biti, ceea ce permite codificarea a 65536 caractere. Unicode ofera o modalitate consistenta pentru codificarea textelor in care sunt utilizate alfabete/caractere diferite cum ar fi cele Latine, Grecesti, Chinezesti, Japoneze etc. Utilizatorii de calculatoare care au de-a face cu texte in diferite limbi, oamenii de afaceri, lingvistii, cercetatorii, oamenii de stiinta, cat si alte categorii de utilizatori ai calculatoarelor beneficiaza de avantajele acestui cod. Unicode vine si in sprijinul matematicienilor si tehnicienilor care utilizeaza in mod regulat simboluri matematice, cat si alte caractere intalnite in textele tehnice. Editoarele de texte (Microsoft Word) utilizeaza in mod curent Unicode.

Coduri detectoare de erori.

Datele stocate, transmise si prelucrate in cadrul sistemelor numerice sunt reprezentate sub forma unor succesiuni de unitati si zerouri. Reprezentarea lor fizica se realizeaza prin niveluri ale unor tensiuni electrice, care pot fi afectate de diverse zgomote, avand cauze diferite. Zgomotele pot modifica unii biti din 0 in 1 si invers, ceea ce conduce la distorsionarea informatiei, la erori. Pe baza unui model statistic al erorilor este posibila crearea unor astfel de reprezentari ale datelor incat sa fie posibile, atat detectarea, cat si corectarea erorilor.

Daca se considera codul ASCII, in care sunt utilizate toate posibilitatile de codificare ale caracterelor, oricare modificare a unui bit intr-un cod al unui caracter dat va conduce la codul valid al altui caracter. Aceasta observatie sta la baza introducerii redundantei, sub forma unor biti suplimentari, ceea ce va permite detectarea si corectarea erorilor.

Detectarea si corectarea erorilor.

Distanta Hamming defineste distanta logica intre doua coduri de caractere valide si este masurata prin numarul de biti prin care difera acestea. Pentru codul ASCII aceasta distanta este egala cu 1. Adugand un singur bit redundant, la codul ASCII al fiecarui caracter alfanumeric, se poate detecta o singura eroare, intrucat codul eronat se va plasa intre doua coduri valide ASCII.

O metoda de recodificare a codului ASCII, pentru a obtine o distanta Hamming egala cu 2, consta in introducerea unui bit de paritate, plasat la stanga codului normal ASCII. Acest bit va fi calculat pe baza *sumei modulo 2* a bitilor egali cu 1 din codul ASCII. In cazul conventiei de *paritate para*, bitul de paritate va fi egal cu rezultatul sumei modulo 2, amintita mai sus, iar in cazul *paritatii impare* acesta va fi egal cu valoarea negata a acesteia.

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

Bit de paritate Codul ASCII

In aceste conditii tabela codurilor ASCII, cu bit de paritate, va avea 256 de intrari/coduri, dintre care jumatate vor fi invalide. In cazul receptionarii unui caracter invalid se poate cere retransmisia, ceea ce in multe situatii nu este convenabil. O solutie ar fi aceea prin care se poate detecta si corecta aparitia unei reori. In acest scop trebuie sa se mareasca numarul bitilor redundanti din codul ASCII.

Pentru detectarea si corectarea unei erori in cadrul fiecarei pozitii a unui cod ASCII, fiecarui cod valid ASCII trebuie sa i se asocieze alte 7 coduri invalide, in care se va modifica exact un singur bit. Acest lucru trebuie sa se intample cu fiecare cod valid ASCII, iar codurile invalide asociate fiecarui cod valid trebuie sa fie diferite. Spre exemplu codului ASCII valid pentru caracterul *d* i se vor asocia codurile invalide de mai jos:

11001100 } cod valid pentru *d*

01001100
10001100
11101100
11011100 } coduri invalide pentru *d*
11000100
11001010
11001101

Problema se pune in legatura cu numarul necesar al bitilor redundanti. Pentru un cod de *k* biti, se considera ca numarul bitilor redundanti egal cu *r*. Pentru fiecare din cele 2^k cuvinte initiale exista caractere de *k* biti, in care cate un bit este modificat, la care se adauga caracterele de *r* biti, cu cate un bit modificat plus caracterul nemodificat. Astfel, vor exista $2^k \times (k + r + 1)$ coduri in total. Pentru a suporta toate aceste coduri trebuie sa fie suficiente combinatii de $k + r$ biti. Relatia care trebuie indeplinita este data mai jos:

$$2^k \times (k + r + 1) \leq 2^{k+r} \text{ sau}$$

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

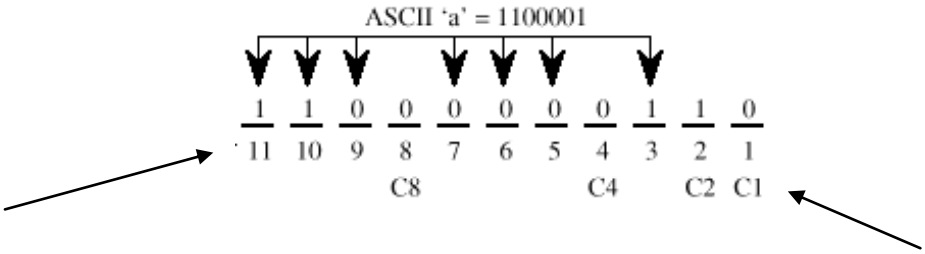
In cazul $k = 7$, daca se cauta intregul *r* care satisface relatia de mai sus, rezulta $r = 4$, ceea ce conduce la un cod ASCII, prevazut cu caractere redundante, de $7 + 4 = 11$ biti.

Asignarea celor 4 biti redundanti la cuvintele originale se va face astfel incat sa poata fi identificata o eroare la un singur bit. Fiecarui bit din cuvantul codificat, incluzand bitii de verificare/redundanti, ii este asignata o combinatie data de biti de verificare $C_8\ C_4\ C_2\ C_1$.

Combinatiile sunt calculate, in figura de mai jos, ca reprezentari binare ale pozitiilor bitilor care sunt verificati, incepand cu pozitia 1. C_1 este in pozitia de bit1, C_2 in pozitia de bit 2, C_4 in pozitia 4 samd. Plasarea lor in pozitii corespunzatoare puterilor lui 2 faciliteaza procesul de localizare a erorii. Acest mod particular de amplasare poarta numele de Cod de Corectare a unei Erori Singulare (CCES) sau SEC (Single Error Correcting)

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

Intrucat pozitiile unitatilor sunt unice in fiecare combinatie a bitilor de verificare, o eroare se poate localiza prin observarea bitilor redundanti eronati. Pentru exemplificare se considera codul ASCII al caracterului '*a'*



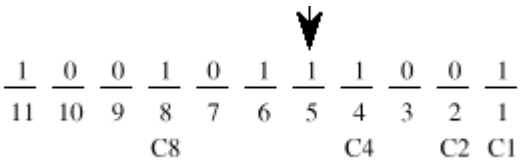
Pozitiile bitilor *Bitii de verificare*

Valorile bitilor de verificare se stabilesc conform tabelului de mai sus. Bitul de verificare C_1= 0 realizeaza paritatea para pentru grupul de biti {1, 3, 5, 7, 9, 11}. Bitul de verificare C_2 = 0 furnizeaza paritatea para pentru grupul de biti {2, 3, 6, 7, 10, 11}. In mod similar bitul de verificare C_4 = 0 asigura paritatea para pentru grupul de biti {4, 5, 6, 7}. In final C_8 = 0 stabileste paritatea para pentru grupul de biti {8, 9, 10, 11}. O posibilitate de cautare in tabela membrilor grupului de paritate, consta in aceea ca bitul *n* al cuvantului codificat este verificat prin bitii din pozitiile 1,…,*i* a caror suma este egala cu *n*. De exemplu bitul 7 este verificat de bitii din pozitiile 1, 2, 4, deoarece 1+ 2 + 4 = 7.

Se considera ca receptorul primeste sirul de biti 10010111001, in conditiile CCES pentru ASCII. Ce caracter a fost transmis?

Se vor calcula paritatile pentru fiecare din bitii de verificare

Bit eronat



	Paritate
C_1 verifica bitii{1, 3, 5, 7, 9, 11}	impara
C_2 verifica bitii{2, 3, 6, 7, 10, 11}	para
C_4 verifica bitii{4, 5, 6, 7}	impara
C_8 verifica bitii{8, 9, 10, 11}	para

Pentru a localiza eroarea, se vor aduna pozitiile bitilor de verificare cu paritate impara:

1 + 4 = 5. Astfel cuvantul care a fost transmis este: 10010101001. Daca se inlatura bitii de verificare, se va obtine codul ASCII al caracterului'D'.

Schema de detectare si corectare a unei erori poate fi vizualizata pe un hipercub, in varfurile caruia sunt plasate codurile valide si invalide. Distanta Hamming separa codurile valide, in timp ce un cod invalid este plasat la o distanta mai mica de un cod valid, ceea ce faciliteaza detectarea si corectarea erorii.

In cazul aparitiei a doua erori, schema de mai sus permite detectarea lor, fara a oferi posibilitatea de a le corecta. In cazul a doua erori, care ar putea fi corectate, distanta minima Hamming trebuie sa fie egala cu 5.

In general, pentru a *detecta* ***p*** erori distanta Hamming minima trebuie sa fie ***p* + 1**, iar o distanta Hamming de **2*p* + 1** permite *corectarea* a ***p*** erori

Verificare prin redundanta verticala.

Schema prezentata mai sus s-a referit la detectarea si corectarea unei erori singulare intr-un caracter individual. Aceasta mai poarta numele de *verificarea redundanta orizontala/transversala* (VRO).

O alternativa o reprezinta *verificarea redundanta verticala/longitudinala* (VRV) in cadrul careia, la sfarsitul unui grup de cuvinte transmise, se adauga o *suma de control.* In acest caz, paritatea este calculata pe coloane, *suma de control* fiind atasata, in final, la mesaj. La receptie, se calculaeaza din nou suma de control, care se compara cu cea receptionata. Daca apare o eroare se solicita retransmiterea mesajului intrucat nu exista suficienta redundanta pentru identificarea pozitiei unei erori. Tehnicile VRO si VRV se pot utiliza pentru a imbunatati verificarea aparitiei unor erori.

In unele cazuri erorile apar in rafala, modificand mai multi biti consecutivi, atat pe linii, cat si pe coloane, in cadrul unui mesaj. Pentu asemenea situatii se recomanda o schema mai puternica de verificare denumita Verificare Redundanta Ciclica (VRC sau CRC - Cyclic Redundancy Checking). VRC reprezinta o varianta a VRV.

Problema. Sa se calculeze numarul necesar de biti de verificare pentru corectarea unei erori duble in codul ASCII.

Solutie. In codul ASCII sunt folositi, pentru fiecare caracter, $k = 7$ biti, la care se vor mai adauga r biti redundanti de verificare. Pentru fiecare dintre cele 2^k caractere ASCII vor exista $(k+ r)$ caractere eronate la nivelul unui bit si $[(k+ r)(k+ r-1)]/2$ caractere eronate la nivelul a doi biti, la care se mai adauga caracterul corect. Numarul caracterelor codificate cu $(k+ r)$ biti este egal cu $2^{(k+ r)}$. In aceste conditii trebuie sa se indeplineasca conditia de mai jos:

$$2^k \text{ x } \{ (k+ r) + [(k+ r)(k+ r-1)]/2 +1 \} \leq 2^{(k+ r)}$$

Inlocuind k cu valoarea sa 7, rezulta ca valoarea cea mai mica pentru r, care satisface relatia de mai sus este $r = 7$.

Intrucat, pentru corectarea a p erori trebuie pastrata o distanta Hamming egala cu $2p + 1$, rezulta ca in acest caz, unde $p = 2$, distanta va fi egala cu 5. In cazul in care se mentine aceeasi distanta si pentru detectarea a erorilor, cunoscand ca distanta Hamming este egala cu numarul erorilor detectate $p + 1$, rezulta ca numarul erorilor detectate este egal cu 4.