9. BAZELE ARITMETICE ALE CALCULATOARELOR NUMERICE

9.1 INTRODUCERE

Întrucât elementele de memorare sunt constituite din dispozitive cu două stări stabile, iar elementele de prelucrare a informației sunt bazate pe circuite logice, care operează pe baza logicii bivalente, într-un calculator numeric datele sunt reprezentate în binar, sub forma unor succesiuni de unități şi zerouri. Există numeroase posibilități pentru reprezentarea datelor, care se deosebesc între ele prin expresibilitate, cost de implementare, uşurința conversiilor de la un format la altul, cât şi prin alte considerente.

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

- **Bit**: 0,1
- **Şir de biți**: secvențe de biți de lungimi date:
  - tetrada: 4 biți,
  - octet/byte: 8 biți,
  - semicuvant: 16 biți,
  - cuvânt: 32 de biți,
  - cuvânt dublu: 64 de biți
- **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 şi modul (cod direct),
    - complementul față de 1 (cod invers),
    - complementul față de 2 (cod complementar).

- **Real** (**Virgulă mobilă)**:
  - precizie simplă (cuvânt de 32 de biți),
  - precizie dublă (cuvânt dublu: 64 de biți),
  - precizie extinsă.

Calculatoarele posedă elemente de stocare a datelor, de tipul registrelor sau al celulelor de memorie, care dispun de un număr finit de elemente/ranguri, ceea ce afectează precizia calculului. Astfel, la programarea unor aplicații numerice, trebuie avute în vedere aspectele legate de precizia limitată a reprezentării informației.

9.2 SISTEME DE NUMERAȚIE

9.2.1 REPREZENTAREA NUMERELOR

Un sistem de numerație constă în totalitatea regulilor şi simbolurilor/cifrelor folosite pentru reprezentarea numerelor. Sistemele de numerație pot fi de două tipuri: *poziționale* şi *nepoziționale*. Într-un sistem pozitional 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. Ca exemplu de sistem nepozițional se poate da sistemul de numerație roman.

Sistemele de numerație poziționale mai poartă numele şi de *sisteme de numerație ponderate*, întrucât valoarea unei cifre depinde de pozitia ei în reprezentarea numărului dat.

Un număr întreg N este reprezentat, într-un sistem de numerație pozițional, în baza *b*, sub forma unui *n*-tuplu de simboluri x_i,

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

unde x_i reprezintă o cifră a sistemului de numerație.

O cifră x_i poate lua valori întregi cuprinse între 0 şi *b*-1, baza *b* reprezentând numărul valorilor posibile pe care le poate lua o cifră oarecare x_i.

În general, un număr, constituit dintr-o parte întreagă şi o parte subunitară are umătoarea reprezentare în baza *b*:

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

Valoarea *N* a numărului N_b se calculaeză cu ajutorul următoarei expresii:

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

Fie numărul $435{,}25_{10}$ în baza 10 (n = 3, m = 2 şi b = 10).

$4 \times 10^2 + 3 \times 10^1 + 5 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2} = 400_{10} + 30_{10} + 5_{10}\, 1 +$

$+ \frac{2}{10_{10}} + \frac{5}{100_{10}} = 435{,}25_{10}$

În continuare se consideră numărul $1011{,}01_2$, pentru care: *n* = 4, *m* = 2 şi *b* = 2:

$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 8_{10} + 0_{10} + 2_{10} + 1_{10}$

$+ \frac{0}{2_{10}} + \frac{1}{4_{10}} = 11{,}25_{10}$

În ultimul exemplu se prezintă metoda polinomială de conversie a numerelor, reprezentate în baza 2, în numere reprezentate în baza 10.

9.2.2 CONVERSIA NUMERELOR DINTR-O BAZĂ ÎN ALTA

Într-un sistem de calcul datele sunt reprezentate în mai multe sisteme de numerație. Astfel, datele de la intrare şi cele de la ieşire sunt, în general, reprezentate în baza 10. În memoria calculatorului şi în unitatea de prelucrare datele sunt reprezentate în baza 2. Sunt situații în care datele, care se prelucreaza, sunt reprezentate în baza 10, cifrele zecimale fiind codificate prin tetrade binare. Pentru a uşura operațiile de programare, uneori, numerele binare sunt convertite în numere reprezentate în baza 8 sau baza 16.

Plecând de la reprezentarea numerelor sub forma coeficienților dezvoltării polinomiale, în raport cu baza, conversia se poate efectua prin operații repetate de împărțire/înmulțire în condițiile numerelor întregi/subunitare.

Se presupune un număr *N* constituit dintr-o parte întreaga N_i şi o parte subunitara N_f: $N = N_i - N_f$

Astfel:

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

$N_f = 0, x_{-1}\, x_{-2}\, x_{-3} \ldots x_i \ldots x_m$

Conversia numerelor întregi din baza b în baza q presupune împărțirea repetată a câtului de la împărțirea precedentă la noua bază şi reținerea restului, mai mic decât noua bază, până când câtul curent devine mai mic decât noua bază. Resturile obținute formează cifrele noului număr, începând cu cea mai puțin semnificativă. Toate operațiile se efectuează în 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 obține coeficienții y_i, ai dezvoltării în baza *q,* se va recurge la următoarea secvență de operații:

$(N_i)_{b:q} = (N_i)_{b^1} + \frac{y_0}{q}$, cifra curentă a noului număr este y_0

$(N_i)_{b^1:q} = (N_i)_{b^2} + \frac{y_1}{q}$, cifra curentă a noului număr este y_1

……..

$(N_i)_{b^{p-2}:q} = (N_i)_{b^{p-1}} + \frac{y_{p-2}}{q}$, cifra curentă a noului număr este y_{p-2} unde $0 \leq (N_i)_{b^{p-1}} \leq q$ reprezintă cifra cea mai semnificativă a noii reprezentări

Exemplu

$17_{10} = ?_2$

$17 : 2 = 8 + \frac{1}{2}, \quad y_0 = 1;$

$8 : 2 = 4 + \frac{0}{2}, \quad y_1 = 0;$

$4 : 2 = 2 + \frac{0}{2}, \quad y_2 = 0;$

$2 : 2 = 1 + \frac{0}{2}, \quad y_3 = 0;$

$y_4 = 1$

$17_{10} = 10001_2$

Conversia numerelor subunitare din baza b în baza q presupune înmulțirea repetată a părții subunitare, care rezultă de la înmulțirea precedentă, cu noua bază şi reținerea părții întregi până când partea

subunitară curentă devine 0 sau până când se epuizează rangurile de reprezentare în noua bază. Ca prima parte subunitară se ia numărul $N_{f,}$ care urmează să fie convertit. Întregii obținuți formează cifrele noului număr, începând cu cea mai semnificativă. Toate operațiile se efectuează în 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 obține coeficienții y_i ai dezvoltării în baza *q* se va recurge la următoarea secvență de operații:

$(N_f)_b \times q = (N_f)_{b^{-1}} + y_{-1}$, cifra curentă a noului număr este y_{-1}

$(N_f)_{b^{-1}} \times q = (N_f)_{b^{-2}} + y_{-2}$, cifra curentă a noului număr este y_{-2}

……..

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

În cazul în care $(N_f)_{b^{-i}} = 0$, procesul se opreşte. Se poate constata faptul că procesul de conversie a numerelor subunitare poate fi însoțit de erori, în cazul unui număr limitat de ranguri pentru reprezentarea în noua bază sau în cazul apariției unor secvențe repetate de unități şi zerouri.

Exemplu

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

Se consideră următorul exemplu:
$0{,}1_{10} = ?_2$
$0{,}1 \times 2 = 0 + 0{,}2; \quad y_{-1} = 0;$
$0{,}2 \times 2 = 0 + 0{,}4; \quad y_{-2} = 0;$
$0{,}4 \times 2 = 0 + 0{,}8; \quad y_{-3} = 0;$
$0{,}8 \times 2 = 1 + 0{,}6; \quad y_{-4} = 1;$
$0{,}6 \times 2 = 1 + 0{,}2; \quad y_{-5} = 1;$
$0{,}2 \times 2 = 0 + 0{,}4; \quad y_{-6} = 0;$
$0{,}4 \times 2 = 0 + 0{,}8; \quad y_{-7} = 0;$
$0{,}8 \times 2 = 1 + 0{,}6; \quad y_{-8} = 1;$
$0{,}6 \times 2 = 1 + 0{,}2; \quad y_{-9} = 1;$
$0{,}2 \times 2 = 0 + 0{,}4; \quad y_{-6} = 0;$
……
$(0{,}1)_{10} = (0, 00011000110 \dots)_2$
Se poate observa că, în acest ultim caz, conversia nu se efectuează exact.

Conversiile între baze, care reprezintă puteri ale lui 2, constituie cazuri particulare, şi se efectuează mecanic. Astfel, în cazul numerelor reprezentate în octal/hexazecimal, cifrele octale/hexazecimale se inlocuiesc cu triadele/tetradele binare corespunzătoare şi invers. Numerele octale/hexazecimale se explorează, pentru conversia în binar, de la dreapta la stânga.

Mai jos se prezintă un tabel de corespondență între cifrele octale, zecimale şi hexazecimale, pe de-o parte şi echivalentele lor binare, pe de alta parte.

Tabelul 9.1. Tabel de corespondență între cifrele octale, zecimale, hexazecimale şi echivalentele lor binare.

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

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

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

9.3 REPREZENTAREA INFORMAȚIEI NUMERICE ÎN CALCULATOARE

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

În cazurile numerelor întregi cu semn şi al mantisei numerelor reale semnul este codificat prin bitul plasat în extrema stângă. Semnul "-" este codificat prin "1", iar semnul "+" prin "0".

Pentru numerele întregi cu semn $N_i = x_{n-1}\, x_{n-2}\, x_{n-3} \dots x_i \dots x_1\, x_0$ semnul este codificat prin bitul x_{n-1}.

Un număr real N_r se reprezintă prin două câmpuri: mantisa/fracția *f*, cu semn, şi exponent *e*:

$$N_r = s\ e\ f$$

unde:

- *s* reprezintă semnul mantisei, codificat printr-un bit, conform convenției menționate mai sus,
- *f* constituie mantisa sub forma unui număr subunitar normalizat ($|m| < \frac{1}{2}$),
- *e* specifică exponentul, de regulă, deplasat cu o anumită cantitate pentru a-l face ≥ 0 .

O discuție mai amănunțită, privind reprezentarea numerelor reale se va face într-un paragraf ulterior.

9.3. 1 CODURI DE REPREZENTARE A NUMERELOR ÎNTREGI, CU SEMN, ÎN CALCULATOARE

Numerele întregi, cu semn, se pot reprezenta în calculatoare în trei moduri diferite, numite uneori şi coduri de reprezentare:

- semn şi modul (cod direct),
- complementul față de 1 (cod invers),
- complementul față de 2 (cod complementar).

<u>Codul direct (semn şi modul)</u>

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

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

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

Se poate observa ca, în acest cod, 0 are două reprezentări, dacă este afectat de semn:

$[+0]_d = 00000 \dots 000$ şi

$[-0]_d = 10000 \ldots 000$

Astfel, un număr *x* în cod direct, reprezentat pe *n* ranguri, poate lua valori în gama:

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

ceea ce face ca pentru:

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

Reprezentarea în modul şi semn este utilă pentru implementarea operației de înmulțire, dar prezintă unele dificultăți la adunare şi scădere.

Codul invers (complementul față de unu)

Denumirea de *complementul față de unu* provine de la faptul că, reprezentarea în virgula fixă s-a realizat în calculatoare, mai întâi, pentru numere subunitare, numerele negative fiind stocate sub forma complementului față de 2, diminuat cu 1, prin scăderea lui $|x|$ din 1. În cazul numerelor binare negative întregi, reprezentate pe *n* ranguri, codurile inverse se obțin prin scăderea modulelor acestora din $2^n - 1$.

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

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

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

Se poate observa că, în acest cod, 0 are două reprezentări, dacă este afectat de semn:

$[+0]_i = 0\ \ 0000 \ldots 000$ și

$[-0]_i = 11111 \ldots 111$

Astfel, un număr *x* în cod invers, reprezentat pe *n* ranguri, poate lua valori în gamă:

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

ceea ce face ca pentru:

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

Reprezentarea în codul invers este identică cu reprezentarea în cod direct, în cazul numerelor pozitive. Pentru a obține codul invers al unui număr se inversează valorile binare ale tuturor rangurilor, operație extrem de uşor de realizat în hardware.

Codul complementar (complementul față de doi)

Denumirea de *complementul față de doi* provine de la faptul că reprezentarea în virgulă fixă s-a realizat în calculatoare mai întâi pentru numere subunitare, iar numerele negative se stocau sub forma complementului față de 2, prin scăderea lui $|x|$ din 2. În cazul numerelor binare, întregi negative, reprezentate pe *n* ranguri, codurile inverse se obțin prin scăderea modulelor acestora din 2^n.

$[x]_c = 0\ x_{n-2}\ x_{n-3}\ \dots\ x_i\ \dots\ x_0$, pentru $x \geq 0$; $[x]_c = 1\ \widetilde{x_{n-2}}\ \widetilde{x_{n-3}}\ \dots\ \widetilde{x_i}\ \dots\ \widetilde{x_0}$, pentru $x < 0$

unde:

$1\ \widetilde{x_{n-2}}\ \widetilde{x_{n-3}}\ \dots\ \widetilde{x_i}\ \dots\ \widetilde{x_0}$ se obține ca urmare a operației: $2^n - |x|$

Se poate observa că, în acest cod, 0 are o singură reprezentare:

$[0]_c = 0\ 0000 \dots 000$

Astfel, un număr *x* în cod complementar , reprezentat pe *n* ranguri, poate lua valori în gamă:

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

ceea ce face ca pentru:

- *n* = 8 valorile minime/maxime să fie –128 / +127, iar
- *n* = 16 valorile minime/maxime să 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 uşor că, prin mijloace hardware, codul complementar al unui număr negativ se poate obține adunând, la inversul numărului (obținut prin negarea logică a tuturor rangurilor), o unitate în cel mai puțin semnificativ rang.

Reprezentarea în cod complementar facilitează realizarea hardware-lui necesar operațiilor de adunare şi scădere în calculatoare.

Reprezentarea în exces

Reprezentarea în exces este cunoscută şi sub numele de reprezentare deplasată. Ideea de bază constă în aceea de a atribui celui mai mic număr (număr negativ) valoarea cea mai mică întreagă ce se poate reprezenta în calculator, manipulând în acest mod numere fără semn. În cazul reprezentării numerelor $(14)_{10}$ şi $(-14)_{10}$ sub forma unor numere binare pe 8 biți, folosind forma *excess 128* va trebui să se obțină corespondentele binare ale numerelor (128 + 14 = 142)$_{10}$ şi (128 + (- 14) = 114)$_{10}$. Astfel, codul binar în exces 128 pentru (-14)$_{10}$ este $(01110010)_2$.

Nu există o semnificație numerică a valorii *în excess*, ea are ca efect deplasarea reprezentării în complementul față de doi. Pentru cazul studiat valoarea *exces* a fost astfel aleasă încât să aibe acelaşi şablon de biți ca şi cel mai mare număr negativ, ceea ce face ca numerele să apară ca şi când ar fi sortate ca numere binare fără semn. Celui mai mic număr negativ $(-128)_{10}$ îi va corespunde $(00000000)_2$, în timp ce celui mai mare număr pozitiv $(127)_{10}$ îi va corespunde $(11111111)_2$. Această reprezentare simplifică efectuarea operației de comparare a numerelor, ceea ce este esențial pentru realizarea hardware-lui necesar operării asupra exponenților, la reprezentarea în virgulă mobilă.

Codul binar-zecimal

Numerele pot fi reprezentate în sistemul de numerație cu baza 10, în condițiile în care rangurile zecimale sunt codificate prin tetrade binare. Acesta este codul binar-zecimal sau BCD (Binary Coded Decimal). Posibilitățile de codificare oferite de către o tetrada binară nu sunt epuizate în cazul BCD, deoarece rămân neutilizate 6 tetrade dintr-un total de 16.

În cazurile în care se doreşte o codificare BCD cu semn, tetradele binare: (1100) şi (1101) se folosesc pentru codificarea semnelor "+" şi "-", în timp ce tetradele binare (0000),….., (1001) se utilizează pentru codificarea cifrelor zecimale.

Pe un octet/byte, în biții cei mai puțin semnificativi, poate fi plasată o singură tetrada: BCD "neîmpachetat". În cazul plasării a două tetrade BCD pe un singur octet, se spune că BCD este "împachetat" .

Codurile BCD se utilizează în calculatoarele destinate calculelor comerciale, financiare, înlăturând necesitatea conversiilor zecimal-binar şi binar-zecimal.

Un număr negativ în baza 10, cu mai multe ranguri, se poate reprezenta în BCD în complementul față de 9 sau față de 10.

În cazul numerelor $(+402)_{10}$ şi $(-402)_{10}$ se pot da reprezentările lor în complementul față de 9 şi față de 10:

a) $\frac{0000}{(0)_{10}}$ $\frac{0100}{(4)_{10}}$ $\frac{0000}{(0)_{10}}$ $\frac{0010}{(2)_{10}}$ $(+402)_{10}$

b) $\frac{1001}{(9)_{10}}$ $\frac{0101}{(5)_{10}}$ $\frac{1001}{(9)_{10}}$ $\frac{0111}{(7)_{10}}$ $(-402)_{10}$ complementul față de 9

c) $\frac{1001}{(9)_{10}}$ $\frac{0101}{(5)_{10}}$ $\frac{1001}{(9)_{10}}$ $\frac{1000}{(8)_{10}}$ $(-402)_{10}$ complementul față de 9

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

9.3.2 REPREZENTAREA ÎN VIRGULĂ MOBILĂ

Reprezentarea numerelor în virgulă fixă precizează un număr de ranguri la stânga virgulei, pentru partea întreagă, şi un alt număr de ranguri la dreapta virgulei, pentru partea subunitară.

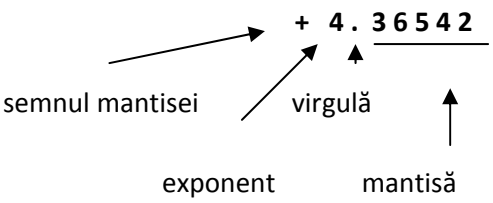
În vederea asigurării unei game largi de reprezentare, cât şi a unei precizii convenabile, în cazul virgulei fixe trebuie să se aloce un număr mare de ranguri. Astfel, în cazul în care se doreşte manipularea numerelor cu valori până la 1 trilion (10^{12}) sunt necesare 40 de ranguri binare, întrucât $10^{12} \cong 2^{40}$. Acelaşi număr de 40 ranguri binare este necesar în cazul asigurării unei precizii de o trilionime. Numerele ar fi astfel reprezentate pe 80 de biți. În practică solicitarile privind gama şi precizia pot fi şi mai mari. Virgula mobilă oferă posibilitatea reprezentării cu un număr mic de ranguri binare a unei game largi de numere exprimabile, prin efectuarea unui compromis între numărul de ranguri, care asigură precizia, şi numărul de ranguri, care asigură gama.

În numărul, reprezentat în virgula mobilă, de mai jos:

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

gama este stabilită de către numărul de ranguri ale exponentului şi 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 şi gama impun un număr de 6 ranguri zecimale, la care se mai adaugă unul pentru codificarea semnului părții subunitare/mantisei:



Pentru mărirea gamei de reprezentare, în afara compromisului între precizie şi gamă, bazat pe modificarea numărului de biți din reprezentările exponentului şi mantisei, se mai poate acționa şi asupra bazei, în sensul creşterii acesteia. Prin aceasta creşte precizia pentru numerele mici, iar pentru numerele mari scade.

Reprezentarea normalizată

Un număr poate fi reprezentat, în virgulă mobilă, în mai multe moduri:

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

Pentru a evita reprezentările multiple ale aceluiaşi număr se introduce *forma normalizată*. Aceasta se obține prin deplasarea spre stânga a mantisei, astfel încât, imediat la dreapta virgulei să se afle o cifra diferită de 0. Pe măsura deplasarii mantisei la stânga se incrementează şi exponentul. Operația nu are sens atunci când mantisa are toate rangurile egale cu 0.

De regula, un exponent este rezervat pentru reprezentarea lui zero şi a altor cazuri speciale, cum ar fi ∞. Cu excepția cazului când este egală cu zero, mantisa posedă în bitul cel mai semnificativ o unitate, pentru o bază egală cu 2. Acest bit, egal cu 1, este prezent în mod implicit, ceea ce permite deplasarea mantisei spre stânga cu un bit, în scopul măririi preciziei. Bitul în cauza poarta numele de "*bit ascuns*".

9.4 TERMINOLOGIA FOLOSITĂ ÎN LEGĂTURĂ CU ERORILE DE CALCUL

În analiza erorilor posibile privind prelucrarea datelor se folosesc o serie de termeni şi concepte specifice.

Precizia. Constituie un termen asociat cu lungimea cuvântului, numărul de biți disponibili într-un cuvânt pentru reprezentarea unui număr dat. În cazul unui registru de 8 biți, considerând că se reprezintă numai numere naturale, precizia de reprezentare va fi de 1/256. Precizia nu trebuie confundată cu acuratețea.

Acuratețea. Aceasta reprezintă o măsură a apropierii unei aproximații față de valoarea exactă. Acest termen nu trebuie confundat cu precizia. Ca exemplu se poate considera reprezentarea numărului natural 6 în binar, pe 4 biți. Reprezentarea exactă, fără nici o eroare va fi 0110. Dacă se ia numărul binar fracționar $0{,}10101010101$, care trebuie reprezentat pe 8 biți se va obține: $0{,}1010101$. Ultima formă constituie o reprezentare cu eroare a numărului inițial. Astfel, în al doile caz reprezentarea, care conține o eroare, este mai precisă (8 biți în loc de 4), dar are o acuratețe mai mică .

Gama. Gama reprezintă mulțimea numerelor reprezentabile într-un sistem dat. Astfel, în reprezentarea numerelor întregi în complementul față de doi, pe patru biți, gama de reprezentare va fi de la -8 la $+7$, adică $(+7) - (-8) = 15$.

Rezoluția. Aceasta constituie mărimea diferenței/distanței între două numere sau cifre adiacente. Pentru reprezentarea cu patru cifre zecimale, gama fiind între 0000 şi 9999, rezoluția este constantă şi egală cu 1. *În cazul reprezentării în virgula mobilă rezoluția nu mai este constantă în cadrul gamei*. Ea este dependentă de valoarea exponentului utilizat.

Trunchierea. Cunoscută şi sub denumirea de rotunjire prin lipsă, această tehnică este utilizată în cazurile în care precizia nu este suficientă pentru reprezentarea corectă a numărului stocat. Considerând stocarea valorii lui $\pi = 3{,}141592654$ într-un dispozitiv capabil să memoreze numai 6 cifre zecimale, numărul $\pi = 3{,}14159$ se va reprezenta cu o eroare de trunchiere egală cu $0{,}000002654$.

Rotunjirea. Metoda prin care se caută să se selecteze valoarea cea mai apropiată de valoarea inițială a numărului poartă numele de rotunjire. În zecimal, dacă cifra aflată la dreapta ultimei cifre, care intră în reprezentare, este mai mare sau egală cu 5 atunci ultima cifră se incrementează cu 1, în caz contrar se lasă neschimbată. În binar, dacă cifra aflată la dreapta ultimei cifre, care intră în reprezentare, este 1 atunci la ultima cifră se adaugă o unitate, în caz contrar cifra nu se modifică.

Depăşirea. Situația de depăşire apare când rezultatul unui calcul este prea mare pentru a putea fi reprezentat în sistem. Spre exemplu, dacă se lucrează în binar cu numere întregi reprezentate în complementul față de doi, pe 4 biți, gama de reprezentare fiind -8 la $+7$, orice rezultat care depaşeşte gama va conduce la depăşire.

Depăşirea superioară şi depăşirea inferioară. Aceste situații 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 sa fie reprezentat în sistem. Pentru a preveni obținerea unor rezultate false/eronate/pseudorezultate se elaborează tehnici prin care se semnalizează apariția unor asemenea situații.

Erori introduse la conversia numerelor din baza zece în baza doi. Aceste erori apar datorită faptului că cele mai multe numere nu reprezintă multipli ai unor fracții binare. Situația se întâlneşte în mod frecvent la introducerea datelor în calculator. De regulă, numărul rangurilor binare este destul de mare pentru a se putea reprezenta numărul necesar de cifre zecimale.

9.5 REPREZENTAREA NUMERELOR REALE

Calculatoarele destinate calculelor ştiinţifice şi inginereşti operează în principal cu numere reale. Se cunoaşte că mulțimea numerelor reale este convexă, adică în oricare interval, indiferent de mărimea sa, există un număr infinit de valori reale. Într-un calculator pot fi reprezentate precis valori discrete, dintr-o mulțime finită, deoarece fiecare informație/dată poate fi reprezentată numai printr-un număr fix de biți. Astfel, oricare reprezentare a unor numere reale, într-un calculator, constituie o aproximare a valorilor exacte.

Un aspect important se referă la faptul că datele reale presupun în acelaşi timp, atât o gamă mare de valori, cât şi o anumită precizie.

În principiu numerele reale se pot reprezenta în două moduri: în *virgulă fixă* şi în *virgulă mobilă*.

Reprezentarea în virgulă *fixă* presupune un număr dat de ranguri pentru partea întreagă şi pentru partea subunitară $X = x_{n-1}\, x_{n-2} \dots x_i \dots x_1\, x_0, x_{-1}\, x_{-2} \dots x_{-i} \dots x_{-m}$.

Deşi prin această reprezentare se poate asigură o precizie satisfăcătoare, în multe cazuri (în funcție de numărul de ranguri $n + m$) apar dificultăți importante în privința gamei de reprezentare a datelor de intrare, a rezultatelor parțiale şi finale în cadrul unui program executat pe un calculator. Aceasta implică introducerea unor factori de scară, care să asigure limitarea valorilor maxime la posibilitățile reale de reprezentare, în calculatorul dat.

Reprezentarea în virgulă mobilă *sau notația ştiințifică* foloseşte în principal două câmpuri: unul pentru *exponent* (*e*) şi altul pentru *mantisă/parte subunitară/fracție* (*f*). Astfel, numărul real *X* se poate scrie $X = f \cdot r^e$ unde *r* este *baza, f - mantisa*, iar *e - exponentul*.

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

Mantisa f se reprezintă ca un număr subunitar, normalizat afectat de semn ($|f| \geq \frac{1}{2}$, pentru a nu pierde din precizie); numărul de biți din mantisă asigură precizia de reprezentare.

Exponentul stabileşte gama de reprezentare; pentru simplificarea algoritmilor de efectuare a operațiilor aritmetice, în virgulă mobilă, se utilizează numai exponenți pozitivi (deplasați), prin adunarea unei cantități date, astfel încât, cel mai mic exponent să nu ia o valoare negativă (să fie zero).

Dacă *exponentul* are *k* biți şi mantisa *m* biți (exclusiv semnul *s*), numărul *X* se va putea reprezenta astfel $X = s\ \ e_{k-1}\, e_{k-2} \dots\ e_0\ \ f_{-1}\, f_{-2} \dots\ f_{-m}$

Semnul *s*, al mantisei, va fi plasat în rangul cel mai semnificativ al reprezentării binare. Stabilirea formatului de reprezentare a numerelor reale, în virgulă mobilă, a constituit obiectul a numeroase studii, care s-au concretizat prin elaborarea unui standard (IEEE 754). Astfel, au fost alese două formate standard:

- *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.

În alegerea bazei s-a plecat de la observația că aceasta trebuie să permită cea mai buna precizie posibilă şi pentru formatul scurt. S-a demonstrat că o bază egală cu 2 asigură o precizie mai

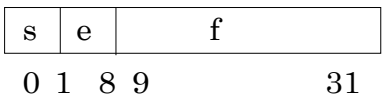
bună în cazul rotunjirii mantisei rezultatului unei operații aritmetice, decât alte baze (4, 8, 16), cel puțin cu un rang în raport cu baza 16. În privința reprezentării se consideră că folosirea codului direct (semn şi modul) este superioară codului complementar.

Dacă lungimea cuvântului este fixată, gama şi precizia se influențează reciproc.

O gamă de $10^{+/-75}$ este insuficientă, în timp ce o gama de $10^{+/-300}$ corespunde cerințelor celor mai multe aplicații. De aceea în cazul formatului scurt a fost sacrificată gama, într-o oarecare măsură, pentru a asigură o bună precizie.

Dacă se foloseşte aceeaşi unitate aritmetică, atât pentru operațiile în formatul scurt, cât şi pentru operațiile în formatul lung, în primul caz nu se vor înregistra depăşiri de tip superior sau inferior. De asemenea, se poate observa că numărul de biți (52) din mantisă în formatul lung este mai mare decât dublul numărului de biți (2×23) ai mantisei în formatul scurt. Aceasta face ca la înmulțirea în formatul scurt adunarea produselor parțiale să nu fie afectată de erori.

Formatul scurt (*simplu - de bază*).



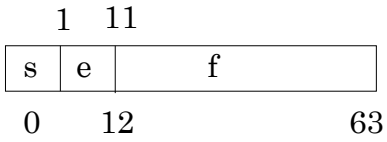
- bitul 0: *s = semnul mantisei* (*s* = 1 - mantisa negativă);
- biții 1-8: *e = exponentul deplasat* (deplasarea este $2^7 - 1$);
- biții 9-31: *f = mantisa/fracția* (când $e \neq 0$, se presupune un bit egal cu 1, la stânga lui *f*; virgula se plasează între acest bit şi primul bit explicit al mantisei);

valoare: numărul reprezentat într-un format scurt este:

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

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

Formatul lung (*dublu - de bază*).



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

valoare: numărul reprezentat într-un format lung este:

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

Observații privind formatele.

1. Baza a fost aleasă din considerente de echilibrare a gamei, cu restricția ca toate numerele mici să aibe valori reciproce reprezentabile. Contrar altor practici, gama obținută este deplasată pentru a putea asigură depăşirea inferioară (underflow), deoarece această situație se poate rezolva mai usor, prin hardware, decât depăşirea superioară (overflow)

2. S-a constatat că prezența unui prim bit implicit, înaintea biților mantisei, asigură depăşirea inferioară gradată. Mai jos se va arăta cum au fost rezolvate aspectele menționate la punctele 1 şi 2.
3. Cazuri speciale.

3.1. În condițiile unui exponent egal cu zero s-a rezervat un câmp pentru:

- *zero*: toți biții sunt zero;
- *date inițializate*: bitul de semn este unu, iar ceilalți biți sunt zero;
- *numere denormalizate*: toate formatele de biți cu exponent zero şi mantisa diferită de zero se interpretează ca numere denormalizate; în efectuarea operațiilor

aritmetice bitul cel mai semnificativ este forțat la zero iar bitul implicit unu este adunat la exponent, ceea ce va permite implementarea gradată a depăşirii inferioare.

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

3.2 În condițiile unui exponent cu toți biții egali cu unu s-a rezervat un câmp pentru:

- $+\infty$: semnul este zero, iar celalți biți sunt unu;
- $-\infty$: toți biții sunt unu:
- nedefinit: semnul şi mantisa sunt zero, iar exponentul este format din unități;
- alte situații: celelalte combinații sunt rezervate pentru utilizări încă nedefinite.

9.6 STANDARDUL IEEE 754 PENTRU REPREZENTAREA NUMERELOR ÎN VIRGULA MOBILĂ

Acest standard a fost acceptat de numeroşi producători de microprocesoare şi de unități centrale pentru minicalculatoare şi calculatoare de capacitate medie-mare. În aceste condiții se vor putea transporta, fără dificultăți majore, pachetele de programe ştiințifice între diferite tipuri de echipamente de calcul.

9.6.1. FORMATE

Formatele prezentate mai sus fac parte din standardul IEEE 754. În plus standardul mai conține formatul extins cu *formele scurtă* (*simplă)* şi *lungă* (*dublă*), în care mantisa este prevazută cu un bit *j*, pentru specificarea explicită a părții întregi, care în formatul de bază discutat a fost presupus implicit. În acest format se dau alte valori minime şi maxime pentru exponenți şi mantise, în raport cu formatul de bază. În cele ce urmează se prezintă în rezumat caracteristicele tuturor formatelor.

Formatul simplu - de bază

Acest format este organizat pe 32 de biți: 1 bit pentru semnul mantisei, 8 biți pentru exponent şi 23 de biți pentru mantisă. Valoarea *V* a numărului se calculează astfel:

- dacă $e = 255$ şi $f \neq 0$, atunci *V* = NaN (nu este un număr -Not a Number - simbol codificat în format);
- dacă $e = 255$ şi $f = 0$, atunci $V = (-1)^s . \infty$
- dacă $0 < e < 255$, atunci $V = (-1)^s . 2^{e-127} . (1, f)$;
- dacă $e = 0$ și $f \neq 0$, atunci $V = (-1)^s . 2^{-126} . (0, f)$;
- dacă $e = 0$ și $f = 0$, atunci $V = (-1)^s .0$ (zero).

Formatul dublu - de bază

Acest format este organizat pe 64 de biți: 1 bit pentru semnul mantisei, 11 biți pentru exponent şi 52 de biți pentru mantisă.

Valoarea *V* a numărului se determină după cum urmează:

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

Formatul simplu - extins

Acest format 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) şi un exponent ce poate lua valori cuprinse între minimum $m = -1022$ şi maximum $M = 1023$.

Valoarea *V* a numărului se determină după cum urmează:

- dacă $e = M$ şi $f \neq 0$, atunci V = NaN;
- dacă $e = M$ şi $f = 0$, atunci $V = (-1)^s . \infty$;
- dacă $m < e < M$, atunci $V = (-1)^s . 2^e . (j, f)$;
- dacă $e = m$ şi $j = f = 0$, atunci $V = (-1)^s . 0$ (zero);
- dacă $e = m$ şi $j \neq 0$ 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 excepția că exponentul ia valori între $m=-16382$ şi $M=16383$, având cel puțin 63 de biți pentru partea fracționară (f).

9.6.2. ROTUNJIREA

Operația de rotunjire tratează numărul ca având o precizie infinită şi il modifică pentru a corespunde formatului destinație. Schema hardware va emite un semnal pentru a arăta dacă rezultatul este corect sau incorect. Standardul IEEE 754 include atât un mecanism standard-implicit de rotunjire, cât şi alte trei metode ce pot fi selectate de utilizator.

Mecanismul standard rotunjeste numărul la cea mai apropiată valoare reprezentabilă. Dacă sunt posibile două valori, formatul este rotunjit la valoarea pară, care face ca eroarea de rotunjire să fie egală cu jumătate din valoarea celui mai puțin semnificativ bit.

Celelalte trei mecanisme de rotunjire selectabile sunt următoarele:

- 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.

Operația de rotunjire se aplică formatului destinație. În formatul dublu sau formatul extins standardul permite utilizatorului să specifice dacă rotunjirea trebuie să se efectueze în formatul simplu - de bază. Aceasta face ca pe o maşină, care are numai formatul dublu de bază sau formatul extins, să se emuleze şi formatul simplu de bază.

9.6.3. VALORI SPECIALE

Standardul de reprezentare în virgula mobilă suporta aritmetica cu numere infinite folosind *modalitățile afină* şi *proiectivă*, selectabile de utilizator.

Modalitatea infinit-afină este definită prin relația:

$-\infty <$ (oricare număr finit) $< +\infty$

Modalitatea infinit-proiectivă compară întotdeauna numerele egale, indiferent de semn. Standardul defineşte un set de operaţii astfel încât folosirea unui operand infinit nu va conduce la un rezultat eronat; pentru detectarea unor asemenea cazuri nu sunt prevazute capcane. În calculele cu valori infinite toate excepţiile, care se vor discuta în continuare, rămân valabile.

NaN constituie o valoare specială, introdusă pentru a semnaliza operaţii invalide sau operaţii care produc rezultate invalide cu o valoare specială. Standardul defineşte posibilităţile de utilizare şi neutilizare ale capcanelor pentru NaN. Folosirea capcanei asigură activarea unei excepţii, la efectuarea unei operaţii cu un operand NaN. Neutilizarea capcanei conduce numai la poziţionarea indicatorilor corespunzători de eroare (condiţii).

9.6.4. OPERAŢII

În standard sunt definite operaţiile aritmetice de bază: adunarea, scăderea, înmulţirea şi împărţirea; rădăcina pătrată şi găsirea restului la împărţire; conversia formatului în: virgula mobilă, numere întregi şi numere zecimale codificate binar BCD (cu excepţia numerelor BCD extinse EBCD); compararea numerelor în virgulă mobilă.

Operaţia de comparare conduce la unele situaţii particulare, când se folosesc operanzi NaN sau cu valori infinite. Relaţiile permise sunt: "mai mic decât", "egal cu", "mai mare decât" şi "neordonat". Ultimul caz apare când cel puţin un operand este NaN şi când un număr infinit este comparat în modalitatea proiectivă cu un număr finit. Relaţia "neordonat" afirmă predicatele "neordonat" şi "< >", negând însă pe toate celelalte.

Atunci când se testează dacă un număr are valoarea NaN, nu va fi posibilă verificarea între o constantă NaN şi un număr, deoarece relaţia este întotdeauna "neordonat".

9.6.5 EXCEPŢII ŞI CAPCANE

Standardul IEEE754 defineşte atât excepţiile, care trebuie detectate, cât şi informatiile necesare elementului, care manipulează capcana, pentru găsirea excepţiilor. Implementarea va asigura câte un indicator pentru fiecare tip de excepţie. Răspunsul standard la excepţie constă în continuarea operaţiei, fără activarea capcanei.

Operaţie invalidă. Acest tip de excepţie se încadrează în *două clase*: excepţii de *operand invalid* şi *rezultat invalid*. În ambele cazuri, dacă nu este activata o capcana, rezultatul va fi NaN. Excepţiile de operand sunt provocate de următoarele evenimente:

- un operand este NaN, fără a se genera un semnal pentru capcana şi fără a se activa capcana;
- se execută una din următoarele operații:

  $(\infty) + (\infty)$ (modalitatea proiectivă),

  $(\infty) + (-\infty)$ (modalitatea afină),

  $\frac{\infty}{\infty}, \frac{0}{0}$ şi $0 \times \infty$;

- în operația $x\ REM\ y$, de aflare a restului la împărțire, fie *x* este infinit, fie *y* este zero;
- când se încearcă extragerea rădăcinii pătrate dintr-un număr negativ;
- la conversia de la formatul în virgulă mobilă la întreg sau BCD, când conversia corectă conduce la o depăşire, la o valoare infinită sau NaN;
- la comparațiile ce folosesc predicatele <, > sau negațiile lor, când relația este de "neordonare" şi când nici un operand nu este"neordonat".

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

Alte excepții. Excepțiile standard sunt următoarele:

- împărțire cu zero;
- depăşire superioară;
- depăşire inferioară;
- rezultat inexact, atunci când nu se generează alte excepții şi când valoarea rotunjită pentru rezultat conduce la depăşire superioară, fără a activa capcana.

Parametrii pentru capcana. 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).

O asemenea rutină trebuie să primească următoarele informații:

- tipul excepției, care a apărut,
- tipul operației, care s-a executat,
- formatul destinației,
- rezultatul corect rotunjit (în cazul depăşirii superioare/inferioare, rezultatului inexact/invalid), pe lângă alte informații referitoare la faptul că rezultatul nu corespunde formatului destinație,
- valorile operandului, în cazurile împărțirii cu zero şi excepțiilor de operand invalid.

9.6.6 STANDARDUL ARITMETIC

Scopul urmărit prin aritmetica în virgula mobilă este acela de a pune la dispoziția utilizatorului un sistem convenabil de reprezentare a numerelor prin care se obțin rezultate precise ca urmare a efectuării operațiilor aritmetice. D. Knuth a sugerat un model de aritmetică în virgulă mobilă, care permite obținerea unor rezultate în virgulă mobilă foarte apropiate de rezultatul corect.

Algoritmii propuşi de Knuth conduc la implementări extrem de costisitoare şi nu oferă soluții pentru cazul rezultatelor aflate exact între două numere în virgula mobilă.

În continuare se prezintă atât un set de reguli, care specifică toate cazurile, cât şi algoritmii de implementare ai acestui set de reguli.

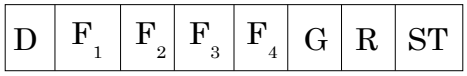
Reguli.

1. Mulțimea numerelor în virgulă mobilă trebuie să conțină 0 şi 1 (identități aditive şi multiplicative), iar dacă *x* aparține mulțimii atunci şi *-x* va aparține mulțimii.
2. Dacă rezultatul corect al unei operații în virgulă mobilă este un număr în virgulă mobilă, atunci acel număr trebuie să fie generat de operația respectivă, în caz contrar rezultatul va fi rotunjit conform regulei 3.
3. Dacă rezultatul corect se află exact la mijlocul intervalului dintre două numere în virgulă mobilă, atunci aritmetica respectiva trebuie să genereze numărul cu mantisa pară.

Regulile de mai sus (regula de rotunjire poartă numele de "rotunjire la par") asigură precizia maximă.

În continuare vor fi ilustrate metodele de implementare ale operațiilor aritmetice în virgulă mobilă (adunare, scădere, înmulțire şi împărțire), care utilizează "rotunjirea la par", în vederea obținerii rezultatului final.

Pentru exemplificare se va considera un acumulator cu următoarea structură, pentru efectuarea operațiilor aritmetice cu numere având mantisa de patru biți:



cu următoarele notații:

- D: bitul de depăşire,
- F_1 - F_4: cei 4 biți ai mantisei,
- G: bitul de gardă,
- R: bitul de rotunjire,
- ST: bitul de legatura/lipitura, "stiky".

Bitul ST este pozitionat în unu, dacă in procesul de denormalizare are loc transferul unei unități din bitul R spre dreapta. Dacă ST este poziționat în unu el va rămâne în această stare pe toată durata operațiilor. Toate deplasările în acumulator implică biții D, G, R şi ST. Bitul ST nu este modificat de deplasarea la stânga. Deplasarea la dreapta introduce zero în bitul D.

În algoritmii de mai jos se consideră că acumulatorul este inițializat la zero şi că operanzii au fost verificați dacă sunt invalizi sau egali cu zero. Astfel, se consideră numai operanzii valizi, normalizați, diferiți de zero. De asemenea, se consideră că operațiile aritmetice, privind exponenții pentru înmulțire, împărțire şi ajustare prin deplasare, sunt evidente, fără a necesita detalieri, şi că semnul rezultatului este stabilit în mod corespunzator.

Adunarea şi scăderea în virgulă mobilă vor fi tratate împreună, evidențiind două cazuri:

- *adunarea modulelor*, atunci când se adună numere cu acelaşi semn sau când se scad numere cu semne diferite;
- *scăderea modulelor*, atunci când se scad numere cu acelaşi semn sau se adună numere cu semne diferite.

1. ***Adunarea modulelor.***
    a. *Denormalizarea*: numărul cu exponent mai mic se încarcă în acumulator şi 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.
    b. *Adunarea:* cel de-al doilea operand este adunat la acumulator.
    c. *Normalizarea:* dacă s-a poziționat în unu bitul D atunci se deplasează conținutul acumulatorului cu un bit spre dreapta.
    d. *Rotunjirea*: se adună 1 la poziția G, după care, dacă G = R = ST = 0, se forțeaza $F_4 \leftarrow 0$.
    e. *Renormalizarea*: dacă D este poziționat în unu se face deplasarea conținutului acumulatorului spre dreapta.
    f. *Depăşirea*: se verifică dacă a avut loc o depăşire a exponenului.

2. ***Scădera modulelor*.**
    a. *Denormalizarea*: se încarcă numărul cu modulul cel mai mic în acumulator şi se deplasează la dreapta, dacă este necesar. Rezultatul va fi zero dacă şi numai dacă operanzii sunt egali, abandonându-se operațiile următoare.
    b. *Scădearea*: se scade conținutul acumulatorului din celalalt operand, păstrând rezultatul în acumulator (dacă ST = 1 se va genera un împrumut);
    c. *Normalizarea*: se deplasează acumulatorul la stânga până când F_1 devine egal cu 1;
    d. *Rotunjirea*: se adună 1 la bitul G şi apoi, dacă G = R = 0 şi ST = 0, se forțează $F_4 \leftarrow 0$; nu este necesară rotunjirea dacă pentru normalizare au fost necesare mai multe deplasări la stânga;
    e. *Renormalizarea*: în cazul în care rotunjirea a condus la depăşire, se efectuează o deplasare spre dreapta.

Înmulțirea

În algoritmul pentru înmulțirea în virgula mobilă se consideră prezente resursele hardware necesare pentru a forma produsul a două numere în lungime dublă. În caz contrar se poate folosi algoritmul "adună şi deplasează la dreapta" în acumulator, pentru a forma produsul, începând cu rangul inferior. În continuare partea mai puțin semnificativă este pierdută pe măsură ce este deplasată în afara bitului ST.

1. *Înmulțirea*: se formează produsul în lungime dublă;
2. *Normalizarea*: pentru o eventuală normalizare a produsului se face o deplasare cu un bit;
3. *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 $G = F_5, R = F_6, ST = F_7\ \cup\ F_8$.
4. *Rotunjirea*: se realizează după cum s-a arătat mai sus;
5. *Renormalizarea*: se realizează după cum s-a arătat mai sus;
6. *Erori:* se verifică depăşirea superioară/inferioară a exponentului.

Împărțirea.

Se consideră că, în procesul efectuării împărțirii, restul este disponibil în orice moment.

1. *Împărțirea*: se formează primii şase biți ai câtului normalizat: $F_1\ F_2\ F_3\ F_4\ F_5\ F_6$
2. *Poziționarea biților* G, R, ST: se forțează: $G = F_5, R = F_6$ și $ST = rest$;
3. *Rotunjirea*: se efectueaza după regulile menționate mai sus;
4. *Renormalizarea*: se efectuează după regulile menționate mai sus.

Prezența bitului ST permite implementarea rotunjirii direcționate, adică la dreapta sau la stânga pe un număr standard de biți. În algoritmii de mai sus s-a căutat obținerea unei precizii maxime cu prețul reducerii vitezei, într-o oarecare măsură.

9.7 ERORI ÎN REPREZENTAREA NUMERELOR ÎN VIRGULĂ MOBILĂ

Precizia finită introduce erori, care pot sau nu pot să fie acceptabile pentru aplicația dată. Spre exemplu, se consideră reprezentarea unui milion, în virgulă mobilă, după care se scade 1, din aceasta. În cazul în care eroarea este mai mare decât 1, restul va fi tot de 1 milion. Pentru a caracteriza eroarea, gama şi precizia, se vor face următoarele notații:

b – bază;

f - numărul rangurilor semnificative ale mantisei/fracției, în baza dată;

M - cel mai mare exponent;

m - cel mai mic exponent.

Numărul rangurilor semnificative din fracție este diferit de numărul de biți în cazul în care baza este diferită de 2. De exemplu, o bază egală cu 16 foloseşte 4 biți pentru fiecare rang. Dacă baza este de forma 2^k (o putere a lui 2), atunci sunt necesari *k* biți pentru fiecare rang exprimat în baza dată. Utilizarea lui 1 ascuns mareşte *f* cu 1 bit, chiar dacă nu conduce la creşterea numărului de numere reprezentabile.

În cadrul analizei reprezentării în virgula mobilă interesează următoarele aspecte:

- numărul numerelor reprezentabile;
- numărul care are valoarea/mărimea cea mai mare;
- numărul diferit de zero, care are valoare/mărimea cea mai mică;
- dimensiunea celei mai mari distanțe între două numere succesive;
- dimensiunea celei mai mici distanțe între două numere succesive.

Numărul numerelor reprezentabile se poate calcula luând în considerare numărul de valori pe care le pot lua câmpurile distincte ale reprezentării în virgulă mobilă:

- semnul poate lua două valori;
- numărul de exponenți este dat de expresia: $((M - m) + 1)$;

- primul rang al fracției: $b-1$;
- celelalte ranguri ale fracției: b^{f-1};
- zero.

Astfel, se obține următoarea expresie: $2 \times \big((M-m)+1\big) \times (b-1) \times b^{f-1} + 1$

Trebuie arătat că, în cazul exponentului, nu toate combinațiile de biți sunt valide. Spre exemplu, în standardul IEEE 754, deşi sunt rezervați 8 biți pentru exponent, cel mai mic exponent este -126 şi nu -128. Exponenții interzişi sunt folosiți pentru reprezentări speciale: zero şi infinit. Se vor considera, în continuare, numărul care are valoarea/mărimea cea mai mare şi numărul diferit de zero, care are valoare/mărimea cea mai mică;

Numărul cu mărimea cea mai mică are, de asemenea, cel mai mic exponent b^m și cea mai mică fracție normalizată (diferită de zero), care va fi egală cu b^{-1}. Astfel, numărul cel mai mic va fi: b^{m-1}. În mod similar numărul cel mai mare va avea cel mai mare exponent b^M şi cea mai mare fracție $(1-b^{-f})$, ceea ce conduce la valoarea $b^M.(1-b^{-f})$.

Cea mai mică distanță şi cea mai mare distanță între două numere consecutive sunt calculate într-o maniera similară. Cea mai mică distanță apare atunci când exponentul are cea mai mică valoare b^m şi când se modifică cel mai puțin semnificativ bit al fracției, b^{-f}. Aceasta conduce la o valoare minimă a distanței egală cu b^{m-f}. Cea mai mare distanță apare atunci când exponentul are cea mai mare valoare b^M şi când se modifică cel mai puțin bit semnificativ al fracției, b^{-f}. Aceasta conduce la o valoare maximă a distanței egală cu b^{M-f}.

Se va considera un exemplu, de reprezentare în virgulă mobilă, în care există un bit de semn, un exponent pe 2 biți în exces 2, o fracție pe 3 biți normalizată, fără 1 ascuns. Reprezentarea lui zero este 000000. În figura 9.1 este data reprezentarea în acest format:

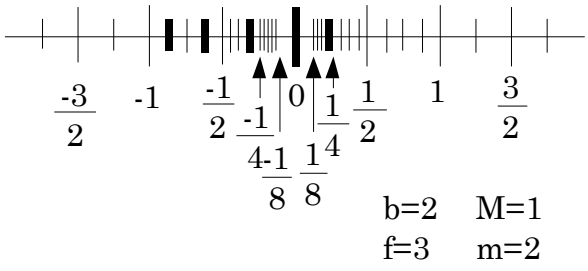


Figura 9.1. Exemplu de reprezentare în virgulă mobilă

Observații:

- au apărut distanțe mari între zero şi primele numere reprezentabile, deoarece reprezentarea normalizată nu permite structuri de biți în mantisă, care să corespundă acestor distanțe;
- cel mai mic exponent este -2 şi cea mai mică fracție normalizata este $(0{,}100)_2$, ceea ce face ca modulul celui mai mic număr reprezentabil, diferit de zero, să fie $\frac{1}{8}$.
- cel mai mare exponent este 2^1, iar cea mai mare fracție este $(1-2^{-3})$, ceea ce conduce la o valoare de $\frac{7}{4}$ pentru cel mai mare număr reprezentabil;
- cel mai mic exponent este -2, iar valoarea celui mai puțin semnificativ bit al fracției este 2^{-3}, conducând la o valoare minimă a distanței între două numere consecutive egală cu 2^{-5}, adică $\frac{1}{32}$;
- cel mai mare exponent este 1, iar valoarea celui mai puțin semnificativ bit al fracției este 2^{-3}, conducând la o valoare maximă a distanței între două numere consecutive egală cu 2^{-2}, adică $\frac{1}{4}$;
- datorită normalizării numărul reprezentarilor valide este mai mic decât numărul reprezentărilor posibile;
- numărul reprezentărilor valide este dat de expresia:

$$2 \times \big((M-m)+1\big) \times (b-1) \times b^{f-1} + 1 = 2 \times ((1-(-2)+1) \times (2-1) \times 2^{3-1} + 1 = 33$$

- erorile relative sunt aproximativ aceleaşi, comparând rapoartele între distanțele minimă/maximă, între două numere consecutive, şi valorile maxime ale celor mai mici/mari numere reprezentabile:

$\frac{b^{m-f}}{b^m \times (1-b^{-f})} = \frac{1}{b^{f-1}}$, pentru numerele mici

distanța — valoarea maximă pentru numărul cu exponent minim

$\frac{b^{M-f}}{b^M \times (1-b^{-f})} = \frac{1}{b^{f-1}}$, pentru numerele mari

distanța — cel mai mare număr

Observație: Standardul IEEE 754 utilizează numere denormalizate pentru a umple distanța între numerele plasate în apropierea zeroului.

Exemplu. Să se reprezinte în virgulă fixă numărul $(0{,}9375)_{10}$. Convertit în binar, în virgulă fixă, acest număr va avea următoarea formă: $(0{,}0011)_2$. Valoarea sa exprimată în virgulă mobilă va fi: $1{,}1 \times 2^{-4}$.

9.8 STUDIU DE CAZ PRIVIND PIERDEREA DE PRECIZIE LA CONVERSIA DE LA ÎNTREGI LA VIRGULĂ MOBILĂ

În timpul razboiului din Golf, 1991-1992, s-au utilizat baterii de rachete Patriot, pentru urmărirea şi distrugerea rachetelor irakiene Scud. Bateriile, prevăzute cu radare şi tehnică de calcul operau numai câteva ore, pentru a se evita detectarea lor. În 5 februarie, 1991, barăcile armatei SUA, de la Dhahran, au fost atacate cu rachete, înregistrându-se 28 de morți, datorită unor erori în partea de prelucrare a datelor de la sistemele radar.

Ținta este urmarită de către radar, stabilindu-se, pe baza parametrilor calculați, poziția viitoare a acesteia. Astfel, se generează o "fereastră" în care se consideră a fi plasată ținta, ceea ce va permite eliminarea altor zgomote, din afara "porții".

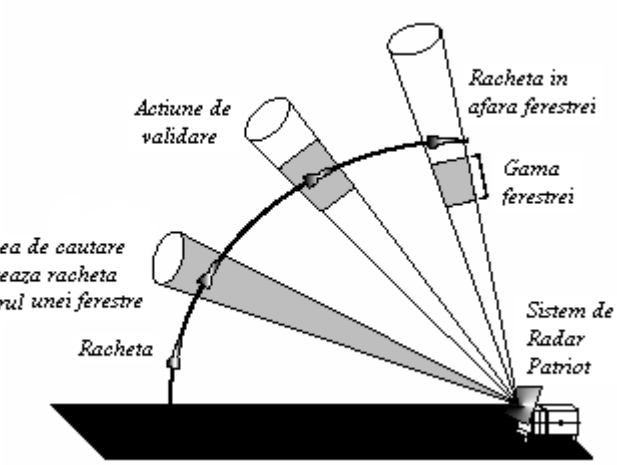


Figura 1.2. Sistem radar

Predicția următoarei poziții a rachetei Scud se calcula pe baza vitezei acesteia. Viteza este determinată prin modificarea poziției în raport cu timpul. Timpul în sistemul Patriot este actualizat de către ceasul intern la intervale de 100 ms. Viteza este reprezentată în virgula mobilă pe 24 de biți, iar timpul ca un întreg pe 24 de biți. Pentru a calcula noua pozitie, timpul şi viteza trebuie să fie reprezentate în virgulă mobilă, pe 24 de biți. Conversia de la întreg la virgula mobilă, pentru timp, s-a facut cu erori, care cresc proporțional cu timpul de operare al sistemului. Astfel, fereastra pentru noua poziție a fost calculată eronat, eroarea depinzând de viteza rachetei şi de durata de timp de când sistemul era operațional. În acea zi Patriot opera de peste 100 de ore, eroarea stabilirii ferestrei fiind de 687 metri, ceea ce a condus la pierderea țintei. Situația fusese semnalată cu două săptămâni anterior de către israelieni. Software-ul a fost modificat, a fost transmis pe câmpul de lupta, dar nu fusese instalat. O soluție simplă, pe moment,

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

9.9 CODURI ALFANUMERICE

Spre deosebire de numerele reale, care au o gama infinită, numărul caracterelor alfanumerice este limitat, ceea ce permite ca un întreg set de caractere să fie codificat cu un număr redus de biți pe caracter. În practică se utilizează trei sisteme de codificarea a caracterelor alfa numerice: ASCII (American Standard Code for Information Interchange), EBCDIC (Extended Binary Coded Decimal Interchange Code) şi Unicode

Codul ASCII

Acest cod, utilizat pentru reprezentarea caracterelor alfanumerice, utilizeaza 7 biți pe caracter. Toate cele 2^7 coduri posibile reprezentând caractere valide. În tabelul 9.2 se prezintă în hexazecimal codurile ASCII alfanumerice.

Tabelul 9.2. Tabel pentru codurile ASCII alfanumerice în codificare hexazecimală

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 pozițiile 00 – 1F şi 7F, sunt caractere de control utilizate pentru transmisie, controlul afişării/tipăririi, cât şi pentru alte scopuri. Toate celelalte caractere sunt afişabile şi includ: literele, cifrele, semnele de punctuație şi spatiul. Cifrele 0 – 9 apar în secvență ca, de altfel, şi literele mici şi mari,

ceea ce simplifică manipularea caracterelor. Pentru a transforma reprezentarea alfanumerica a unei cifre zecimale în valoarea sa numerică este suficient să se scadă $(30)_{16}$ din ea. Pentru a transforma codul unei majuscule în codul corespunzator al literei mici se va aduna la primul $(20)_{16}$.

Codul EBCDIC

Codul ASCII permite reprezentarea a 128 de caractere, ceea ce reprezintă o limitare pentru cele mai multe tastaturi moderne. Aceste tastaturi conțin, pe lângă tastele alfanumerice şi taste dedicate unor caractere speciale. Codul EBCDIC este un cod pe 8 biți utilizat de către IBM. În tabelul 9.3 se prezintă, în codificare hexazecimala codurile EBCDIC.

Tabelul 9.3. Tabel pentru codurile EBCDIC în codificare hexazecimală

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 biți, este reprezentat pe un octet, având 0 sau 1 in bitul cel mai semnificativ, care este folosit drept bit de paritate. Aceasta nu este de natura să favorizeze codul ASCII in raport cu codul EBCDIC. In schimb, la transmisia seriala a datelor codurile pe 8 biți necesita mai mult timp decât cele pe 7 biți.

Unicode

Unicode reprezinta un standard (ISO/IEC 10646) pentru reprezentarea caracterelor cu ajutorul unor cuvinte de 16 biți, ceea ce permite codificarea a 65536 caractere. Unicode ofera o modalitate consistentă pentru codificarea textelor în 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, lingviştii, cercetătorii, oamenii de ştiință, cât şi alte categorii de utilizatori ai calculatoarelor beneficiează de avantajele acestui cod.

Unicode vine şi în sprijinul matematicienilor şi tehnicienilor care utilizează în mod regulat simboluri matematice, cât şi alte caractere întâlnite în textele tehnice. Editoarele de texte (Microsoft Word) utilizeaza in mod curent Unicode.

9.10 CODURI DETECTOARE DE ERORI

Datele stocate, transmise şi prelucrate în cadrul sistemelor numerice sunt reprezentate sub forma unor succesiuni de unități şi zerouri. Reprezentarea lor fizică se realizează prin niveluri ale unor tensiuni electrice, care pot fi afectate de diverse zgomote, având cauze diferite.

Zgomotele pot modifica unii biți din 0 în 1 şi invers, ceea ce conduce la distorsionarea informației, la erori. Pe baza unui model statistic al erorilor este posibilă crearea unei astfel de reprezentari ale datelor încât să fie posibile, atât detectarea, cât şi corectarea erorilor.

Dacă se consideră codul ASCII, în care sunt utilizate toate posibilitățile de codificare ale caracterelor, oricare modificare a unui bit într-un cod al unui caracter dat va conduce la codul valid al altui caracter. Această observație stă la baza introducerii redundanței, sub forma unor biți suplimentari, ceea ce va permite detectarea şi corectarea erorilor.

Detectarea şi corectarea erorilor

Distanța Hamming defineşte distanța logică între două coduri de caractere valide şi 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. În cazul convenției de *paritate pară*, bitul de paritate va fi egal cu rezultatul sumei modulo 2, amintită mai sus, iar în cazul *parității impare* acesta va fi egal cu valoarea negată a acesteia.

Tabelul 9.4 Tabelul pentru codurile ASCII cu bit de paritate

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

În aceste condiții tabelul codurilor ASCII, cu bit de paritate, va avea 256 de intrări/coduri, dintre care jumătate vor fi invalide. În cazul recepționării unui caracter invalid se poate cere retransmisia, ceea ce în multe situații nu este convenabil. O soluție ar fi aceea prin care se poate detecta şi corecta apariția unei erori. În acest scop trebuie să se mărească numărul biților redundanți din codul ASCII.

Pentru detectarea şi 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. Acest lucru trebuie să se întâmple cu fiecare cod valid ASCII, iar codurile invalide asociate fiecărui cod valid trebuie să 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 în legătură cu numărul necesar al biților redundanți. Pentru un cod de *k* biți, se consideră că numărul biților redundanți egal cu *r*. Pentru fiecare cele 2^k cuvinte inițiale există caractere de *k* biți, în care câte un bit este modificat, la care se adaugă caracterele de *r* biți, cu câte un bit modificat plus caracterul nemodificat. Astfel, vor exista $2^k \times (k + r + 1)$ coduri în total. Pentru a suporta toate aceste coduri trebuie să fie suficiente combinații de $k + r$ biți. Relația care trebuie îndeplinită este dată mai jos:

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

În cazul *k* = 7, dacă se cauta 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 $C_8\ C_4\ C_2\ C_1$.

Combinațiile sunt calculate, în tabelul 9.5, ca reprezentări binare ale pozițiilor biților care sunt verificați, începând cu poziția 1. C_1 este în poziția de bit 1, C_2 în poziția de bit 2, C_4 în poziția 4 şamd. Plasarea lor în poziții corespunzătoare puterilor lui 2 facilitează procesul de localizare a erorii. Acest mod particular de amplasare poartă numele de Cod de Corectare a unei Erori Singulare (CCES) sau SEC (Single Error Correcting).

Tabelul 9.5. Tabelul pentru combinațiile de biți de verificare

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

Întrucât pozițiile unităților sunt unice în fiecare combinație a biților de verificare, o eroare se poate localiza prin observarea biților redundanti eronati. Pentru exemplificare se consideră codul ASCII al caracterului “a”.

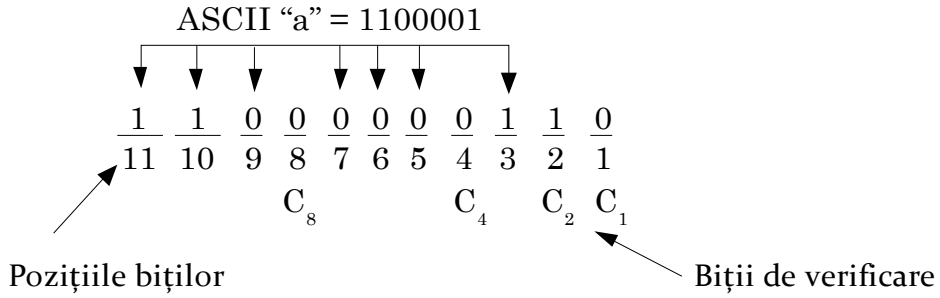


Figura 2.3. Exemplu: codul ASCII al caracterului “a”.

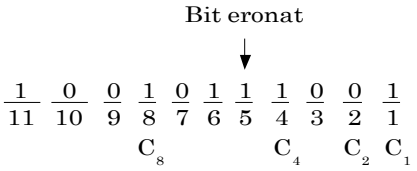
Valorile biților de verificare se stabilesc conform tabelului de mai sus. Bitul de verificare $C_1 = 0$ realizează paritatea pară pentru grupul de biți $\{1, 3, 5, 7, 9, 11\}$. Bitul de verificare $C_2 = 0$ furnizează paritatea pară pentru grupul de biți $\{2, 3, 6, 7, 10, 11\}$. În mod similar bitul de verificare $C_4 = 0$ asigură paritatea pară pentru grupul de biți $\{4, 5, 6, 7\}$. În final $C_8 = 0$ stabileşte paritatea pară pentru grupul de biți $\{8, 9, 10, 11\}$. O posibilitate de căutare în tabela membrilor grupului de paritate, constă în aceea că

bitul *n* al cuvântului codificat este verificat prin biții din pozițiile $1, \ldots, i$ a căror sumă este egală cu *n*. De exemplu bitul 7 este verificat de biții din pozițiile $1, 2, 4$, deoarece $1 + 2 + 4 = 7$.

Exemplu

Se consideră că receptorul primeşte şirul de biți 10010111001, în condițiile CCES pentru ASCII. Ce caracter a fost transmis?

Se vor calcula parităților pentru fiecare din biții de verificare



	Paritate
C_1 verifică biții $\{1, 3, 5, 7, 9, 11\}$	impară
C_2 verifică biții $\{2, 3, 6, 7, 10, 11\}$	pară
C_4 verifică biții $\{4, 5, 6, 7\}$	impară
C_8 verifică biții $\{8, 9, 10, 11\}$	pară

Pentru a localiza eroarea, se vor aduna pozițiile biților de verificare cu paritate impară: $1 + 4 = 5$. Astfel cuvântul care a fost transmis este: 10010101001. Dacă se înlătură biții de verificare, se va obține codul ASCII al caracterului 'D'.

Schema de detectare şi corectare a unei erori poate fi vizualizată pe un hipercub, în vârfurile căruia sunt plasate codurile valide şi invalide. Distanța Hamming separă codurile valide, în timp ce un cod invalid este plasat la o distanță mai mică de un cod valid, ceea ce facilitează detectarea şi corectarea erorii.

În cazul apariției a două erori, schema de mai sus permite detectarea lor, fără a oferi posibilitatea de a le corecta. În cazul a două erori, care ar putea fi corectate, distanța minimă Hamming trebuie să fie egală cu 5.

În general, pentru a *detecta* ***p*** erori distanța Hamming minima trebuie să fie ***p*** **+ 1**, iar o distanță Hamming de **2*p* + 1** permite *corectarea* a ***p*** erori

Verificare prin redundanță verticală

Schema prezentată mai sus s-a referit la detectarea şi corectarea unei erori singulare într-un caracter individual. Aceasta mai poartă numele de *verificarea redundanță orizontală/longitudinală* (VRO).

O alternativă o reprezintă *verificarea redundantă verticală/longitudinală* (VRV) în cadrul căreia, la sfârşitul unui grup de cuvinte transmise, se adaugă o *sumă de control.* În acest caz, paritatea este calculată pe coloane, *suma de control* fiind ataşată, în final, la mesaj. La recepție, se calculează din nou suma de control, care se compară cu cea receptionată. Dacă apare o eroare se solicită retransmiterea mesajului întrucât nu există suficientă redundanță pentru identificarea poziției unei erori. Tehnicile VRO şi VRV se pot utiliza pentru a îmbunătăți verificarea apariției unor erori.

În unele cazuri erorile apar în rafala, modificând mai multi biți consecutivi, atât pe linii, cât şi pe coloane, în cadrul unui mesaj. Pentu asemenea situații se recomandă o schema mai puternică de verificare denumită Verificare Redundantă Ciclică (VRC sau CRC - Cyclic Redundancy Checking). VRC reprezintă o variantă a VRV.

Problemă

Să se calculeze numărul necesar de biți de verificare pentru corectarea unei erori duble în codul ASCII.

Soluție

În codul ASCII sunt folosiți, pentru fiecare caracter, $k = 7$ biți, la care se vor mai adăuga *r* biți redundanți de verificare. Pentru fiecare dintre cele 2^k caractere ASCII vor exista $(k + r)$ caractere eronate la nivelul unui bit şi $\frac{(k+r)(k+r-1)}{2}$ caractere eronate la nivelul a doi biți, la care se mai adaugă caracterul corect. Numărul caracterelor codificate cu $(k + r)$ biți este egal cu $2^{(k+r)}$. În aceste condiții trebuie să se îndeplinească condiția următoare: $2^k \times \left\{(k + r) + \frac{(k+r)(k+r+1)}{2} + 1\right\} \leq 2^{k+r}$

Înlocuind *k* cu valoarea să 7, rezultă că valoarea cea mai mică pentru *r*, care satisface relația de mai sus este $r = 7$. Întrucât, pentru corectarea a *p* erori trebuie păstrată o distanță Hamming egală cu $2p + 1$, rezultă că în acest caz, unde $p = 2$, distanța va fi egală cu 5. În cazul în care se menține aceeaşi distanță şi pentru detectarea erorilor, cunoscând că distanța Hamming este egală cu numărul erorilor detectate $p + 1$, rezultă că numărul erorilor detectate este egal cu 4.