2 BAZELE ARITMETICE ALE CALCULATOARELOR ELECTRONICE

2.1 Sisteme de numeratie

Sistem de numeratie este totalitatea regulilor de reprezentare a numerelor cu ajutorul unor simboluri numite cifre. Cifra este un simbol care reprezinta o cantitate intreaga. Baza (radacina) sistemului de numeratie este numarul de simboluri permise pentru reprezentarea cifrei.

In activitatea de programare se utilizeaza cel mai mult sistemele de numeratie cu bazele 2, 8, 10 si 16. In continuare este prezentat un tabel cu reprezentarile unor numere naturale in cele patru baze:

b=10	b=2	b=8	b=16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
...	...	...	...

Se observa ca pentru fiecare sistem de numeratie numarul de cifre diferite utilizate este egal cu baza. In baza 16 pe langa cele 10 cifre zecimale se utilizeaza primele sase litere ale alfabetului pentru reprezentarea cifrelor cu valorile $10_{(10)}$-$15_{(10)}$.

Schimbarea bazei de numeratie

O problema importanta care se pune in legatura cu sistemele de numeratie este schimbarea bazei de numeratie pentru reprezentarea numerelor. Considerand un numar real fara semn (sau pozitiv), schimbarea bazei se face separat pentru partea intreaga si separat pentru partea subunitara.

Se considera un numar intreg fara semn *N* reprezentat in baza *x* si se doreste reprezentarea acestuia intr-o noua baza *y*. Operatia de conversie este echivalenta cu aflarea coeficientilor polinomului in puteri pozitive ale noii baze *y*, prin care se poate reprezenta numarul:

$$N = a_n y^n + a_{n-1} y^{n-1} + ... + a_1 y + a_0$$

Se efectueaza impartiri succesive la noua baza *y*, retinand la fiecare operatie restul. Se face o prima operatie de impartire la noua baza, obtinandu-se:

$$N / y = a_n y^{n-1} + a_{n-1} y^{n-2} + ... + a_1 + a_0/y$$

Restul impartirii a_0 este prima cifra, cea mai putin semnificativa a rezultatului, iar catul (numar intreg) se noteaza cu N_1 si se face o noua impartire:

$$N_1 / y = a_n y^{n-2} + a_{n-1} y^{n-3} + ... + a_2 y + a_1/y$$

Din nou, restul impartirii a_1 este cifra urmatoare a rezultatului, iar catul se noteaza cu N_2 si se face o noua impartire la *y*,..., astfel incat la un pas oarecare se obtine:

$$N_k / y = a_n y^{n-k-1} + a_{n-1} y^{n-k-2} + ... + a_{k+1} y + a_k/y$$

Rezulta cifra curenta a rezultatului conversiei a_k, ca fiind restul impartirii, iar catul se noteaza cu N_{k+1} si se continua. Conversia se incheie atunci cand dupa o operatie de impartire se obtine catul zero (chiar daca dupa acest moment se continua algoritmul se vor obtine numai cifre 0 la rest, ceea ce inseamna zerouri nesemnificative in stanga unui numar intreg).

Exemplu. Sa se realizeze conversia numarului $N = 41_{(10)}$ din baza 10 in noua baza 2.

41 : 2 => cat =20 si rest = 1 => a_0 = 1

```
20 : 2 => cat = 10  si rest = 0 => a1 = 0
10 : 2 => cat = 5   si rest = 0 => a2 = 0
 5 : 2 => cat = 2   si rest = 1 => a3 = 1
 2 : 2 => cat = 1   si rest = 0 => a4 = 0
 1 : 2 => cat = 0   si rest = 1 => a5 = 1
```

S-a obtinut astfel ca $N = 101001_{(2)}$. Se poate face si o verificare a rezultatului obtinut facand conversia inversa din baza 2 in baza 10, prin adunarea tuturor produselor intre fiecare cifra a reprezentarii in baza 2 si ponderea sa:

$$N = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 41$$

deci, conversia s-a facut corect.

Aplicatie. Sa se scrie un algoritm in pseudocod pentru conversia unui numar intreg din zecimal in binar.

```
citeste nr
i = 0
cat timp nr ≥ 2^i executa
|    a[i] = 0
|    i = i + 1
i = i - 1; n = i; a[n] = 1; nr = nr - 2^i
cat timp nr ≠ 0 executa
|    i = i - 1
|    daca nr ≥ 2^i atunci
|    |    a[i] = 1
|    |    nr = nr - 2^i
scrie (a[i], i = 0,n)
```

Se considera un numar *N* subunitar, fara semn, scris in baza *x* si se doreste realizarea conversiei intr-o noua baza *y*. Operatia de conversie este echivalenta cu aflarea coeficientilor polinomului in puteri negative ale noii baze *y*, prin care se poate reprezenta numarul:

$$N = a_{-1}y^{-1} + a_{-2}y^{-2} + ... + a_{-m}y^{-m}$$

Indicii negativi ai coeficientilor polinomului nu au nici o semnificatie speciala, s-a incercat numai punerea in corespondenta a acestora cu puterile corespunzatoare ale bazei *y*. Pentru aflarea coeficientilor este necesar sa se efectueze inmultiri succesive cu noua baza *y*, retinand de fiecare data partea intreaga a rezultatului. Astfel, la prima inmultire se obtine:

$$N \cdot y = a_{-1} + a_{-2}y^{-1} + a_{-3}y^{-2} + ... + a_{-m}y^{-m+1}$$

Partea intreaga a rezultatului a_{-1} este prima cifra, cea mai semnificativa, a rezultatului conversiei. Partea subunitara se noteaza cu N_1 si se face o noua inmultire:

$$N_1 \cdot y = a_{-2} + a_{-3}y^{-1} + a_{-4}y^{-2} + ... + a_{-m}y^{-m+2}$$

Rezulta a_{-2} cifra urmatoare a rezultatului, iar partea subunitara se noteaza cu N_2 executand o noua inmultire, etc., astfel incat la un pas oarecare se obtine:

$$N_k \cdot y = a_{-k-1} + a_{-k-2}y^{-1} + a_{-k-3}y^{-2} + ... + a_{-m}y^{-m+k+1}$$

Se obtine partea intreaga a_{-k-1}, iar partea subunitara se noteaza N_{k+1} si se continua. Conversia se incheie fie in momentul in care se obtine partea subunitara a rezultatului inmultirii egala cu zero (daca in acest moment s-ar continua algoritmul, atunci s-ar obtine numai cifre 0 nesemnificative in dreapta unui numar subunitar), fie cand s-a calculat numarul propus de cifre (s-a atins precizia dorita). Spre deosebire de conversia numerelor intregi, care se realizeaza intotdeauna exact, conversia numerelor subunitare se face de cele mai multe ori aproximativ (in functie de cele doua baze sau de numerele care se convertesc). De aceea, pentru numerele subunitare se propune de la inceput numarul de cifre pe care sa se reprezinte rezultatul, iar in momentul in care s-au calculat toate cifrele conversia se incheie.

Exemplu. Sa se converteasca numarul $N = 0.37_{(10)}$ din baza 10 in baza 2. Ne propunem ca rezultatul sa fie obtinut pe 7 biti (este precizia rezultatului).

```
0.37 x 2 = 0.74    => a-1 = 0
0.74 x 2 = 1.48    => a-2 = 1
0.48 x 2 = 0.96    => a-3 = 0
0.96 x 2 = 1.92    => a-4 = 1
0.92 x 2 = 1.84    => a-5 = 1
0.84 x 2 = 1.68    => a-6 = 1
0.68 x 2 = 1.36    => a-7 = 1
```

S-a obtinut $N \approx .0101111_{(2)}$. Se poate face o verificare asupra preciziei rezultatului, efectuand conversia din baza 2 in baza 10, cum s-a facut si

pentru numerele intregi, prin adunarea produselor intre cifre si ponderile acestora.

$$N \approx 0\cdot 2^{-1}+1\cdot 2^{-2}+0\cdot 2^{-3}+1\cdot 2^{-4}+1\cdot 2^{-5}+1\cdot 2^{-6}+1\cdot 2^{-7} =$$

$$= (0\cdot 2^{6}+1\cdot 2^{5}+0\cdot 2^{4}+1\cdot 2^{3}+1\cdot 2^{2}+1\cdot 2^{1}+1\cdot 2^{0})/2^{7} =$$

$$= 47 / 128 = 0.367$$

Rezultatul este destul de aproape de valoarea initiala, tinand cont de numarul mic de biti care s-au utilizat pentru reprezentarea rezultatului. Evident, marind numarul de biti se poate obtine o precizie mai buna.

Aplicatie. Sa se scrie un algoritm in pseudocod pentru conversia unui numar subunitar din zecimal in binar, cu precizia pe *m* biti.

```
citeste nr, m
pentru i=1,m executa
└    a[i] = 0
i = 0
cat timp (nr ≠ 0)si(i < m) executa
│    i = i + 1
│    daca nr ≥ 2^-i atunci
│    │    a[i] = 1
└    └    nr = nr - 2^-i
scrie (a[i], i = 1,m)
```

Exemplu. Sa se converteasca numarul $N = 41.37_{(10)}$ din baza 10 in baza 2. Ne propunem ca partea subunitara sa fie obtinuta pe 7.

Se realizeaza conversia separat pentru numarul intreg 41 si separat pentru numarul subunitar 0.37 (ca in exemplele precedente), iar apoi se concateneaza cele doua rezultate, obtinand in final $N \approx 101001.0101111_{(2)}$.

Cazuri particulare

Exista doua cazuri particulare la schimbarea bazei de numeratie, cand conversia se face in mod direct, fara operatii de impartire si inmultire, iar rezultatul este exact si pentru partea subunitara.

1) $x = y^n$. In acest caz se inlocuieste fiecare cifra a reprezentarii in baza *x* printr-un grup de *n* cifre corespunzatoare in baza *y*.

Exemplu. Sa se converteasca numarul $N = 3CF.4AE_{(16)}$ din baza 16 in baza 2.

Intre cele doua baze exista relatia $16 = 2^4$, deci $n = 4$. Pentru conversie se va inlocui fiecare cifra a reprezentarii in baza 16 printr-un grup de patru cifre in baza 2. Astfel:

```
 3    C    F    .    4    A    E
 ↓    ↓    ↓         ↓    ↓    ↓
0011 1100 1111   .   0100 1010 1110
```

Se poate renunta la scrierea zerourilor de la inceput si sfarsit, asa ca rezultatul final obtinut este $N = 1111001111.01001010111_{(2)}$. Foarte important, conversia s-a facut exact.

2) $x^n = y$. In acest caz se formeaza in cadrul reprezentarii in vechea baza x, grupuri de cate n cifre de la punctul zecimal spre stanga pentru partea intreaga si de la punctul zecimal spre dreapta pentru partea subunitara (daca este necesar se vor completa grupurile extreme cu zerouri), iar apoi se inlocuieste fiecare grup printr-o cifra corespunzatoare in noua baza y. De asemenea, rezultatul este exact.

Exemplu. Sa se converteasca numarul $N = 11001111.1101011_{(2)}$ din baza 2 in baza 8.

Avem relatia $2^3 = 8$ intre cele doua baze, deci $n = 3$. Se formeaza grupuri de cate trei cifre in cadrul reprezentarii in baza 2, se completeaza cu zerouri grupurile extreme si se inlocuiesc apoi grupurile cu cifrele corespunzatoare in baza 8.

```
011 001 111 . 110 101 100
 ↓   ↓   ↓     ↓   ↓   ↓
 3   1   7  .  6   5   4
```

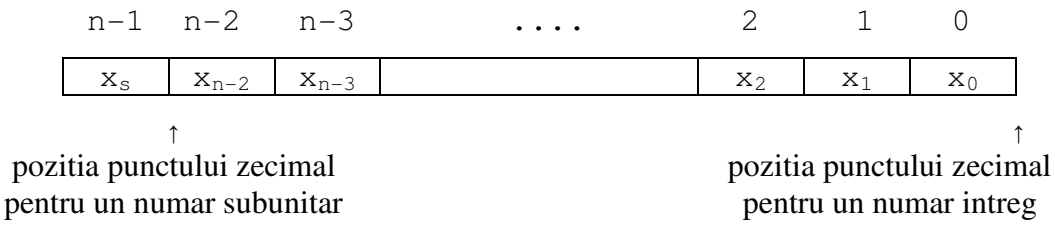
S-a obtinut astfel rezultatul (exact !) $N = 317.654_{(8)}$, fara sa se utilizeze operatii de impartire si inmultire.

2.2 Reprezentarea numerelor in virgula fixa

In general, sistemele de calcul prelucreaza o mare cantitate de informatie numerica. Numerele se pot reprezenta intern prin doua mari

metode: in virgula fixa (numere intregi sau numere subunitare) si in virgula mobila (numere reale). In continuare se va discuta reprezentarea numerelor in virgula fixa.

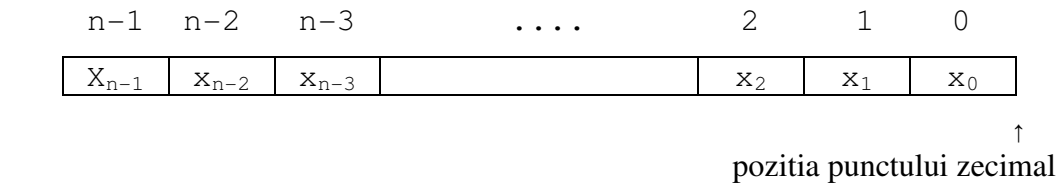
Se considera un numar x reprezentat in calculator pe n biti, sub forma:



unde x_s reprezinta bitul de semn, 0 pentru numar pozitiv si 1 pentru numar negativ, iar ceilalti biti reprezinta modulul numarului. Astfel, valoarea in zecimal a numarului se poate calcula dupa relatia:

$$|x| = \begin{cases} \sum_{k=0}^{n-2} x_k 2^k & \textit{pentru numar intreg} \\ \sum_{k=0}^{n-2} x_k 2^{k-n+1} & \textit{pentru numar subunitar} \end{cases}$$

Varianta: un numar intreg x fara semn reprezentat in calculator pe n biti, sub forma:



toti ce n biti reprezinta valoarea numarului. Astfel, valoarea in zecimal a numarului se poate calcula dupa relatia:

$$x = \sum_{k=0}^{n-1} x_k 2^k$$

Exemple. Sa se reprezinte in cod direct numerele $x = 21$ si $y = -20$ pe 6 biti.

```
[x] = 010101
[y] = 110100
```

2.3 Adunarea si scaderea in virgula fixa

In cazul operatiilor in virgula fixa pentru numere cu semn se trateaza separat semnele si separat modulele operanzilor. Se considera operatia *x op y = z*, unde se codifica cu *op* operatia de realizat, 0 pentru adunare si 1 pentru scadere. Operatia finala de efectuat intre modulele celor doi operanzi este data de relatia :

$$opfin = x_s \oplus y_s \oplus op$$

unde x_s si y_s sunt bitii de semn ai celor doi operanzi. Aceasta relatie poate fi obtinuta daca se studiaza toate cele opt combinatii posibile la adunarea / scaderea a doua numere pozitive / negative. Se disting doua cazuri:

1) *opfin = 0*. In acest caz se aduna modulele celor doi operanzi, iar semnul rezultatului este dat de semnul primului operand. Este necesar sa se verifice ca la adunarea modulelor nu rezulta un transport de la rangul cel mai semnificativ (c.m.s.), o astfel de situatie insemnand depasire, deci rezultatul obtinut nu este corect (rezultatul este prea mare in modul pentru a se putea reprezenta pe lungimea respectiva de biti).

Exemplu. Sa se efectueze operatia *x-y=z,* unde *x* = 11, *y* = -14, pe 6 biti (in continuarea exemplele vor fi de asemenea cu numere reprezentate in virgula fixa, pe 6 biti si nu se va mai preciza acest lucru).

```
[x] = 001011
[y] = 101110
```

Se calculeaza $opfin = x_s \oplus y_s \oplus op = 0 \oplus 1 \oplus 1 = 0$ (s-a considerat *op*=1, pentru scadere). Semnul rezultatului este z_s=x_s=0, iar modulul se calculeaza:

```
|x|+       01011+
|y|        01110
———    =>  ——————
|z|        11001
```

deci, s-a obtinut [*z*] = 011001 (*z* = 25).

2) *opfin = 1*. In acest caz se scade modulul operandului mai mic din modulul operandului mai mare, iar semnul rezultatului este dat de semnul operandului mai mare in modul. *Exceptie*: in cazul *op* = 1 (scadere) si |*y*|>|*x*|, semnul rezultatului este $z_s = \bar{y}_s$.

Exemplu. Sa se efectueze operatia *x*+*y*=*z,* unde *x* = -29, *y* = 17.

```
[x] = 111101
[y] = 010001
```

Se calculeaza $opfin = x_s \oplus y_s \oplus op = 1 \oplus 0 \oplus 0 = 1$ (s-a considerat *op*=0, pentru adunare). Deoarece |*x*|>|*y*|, semnul rezultatului este z_s=x_s=1, iar modulul se calculeaza:

```
|x|-        11101-
|y|         10001
———   =>    ——————
|z|         01100
```

deci, s-a obtinut [*z*] = 101100 (*z* = -12).

2.4 Inmultirea in virgula fixa

Pentru inmultirea a doua numere reprezentate in virgula fixa (*x*• *y* = z) se trateaza separat semnele si separat modulele. Se parcurg urmatoarele etape:

1) *Determinarea semnului rezultatului.* Se utilizeaza relatia: $z_s = x_s \oplus y_s$, care rezulta din regula semnelor de la aritmetica.

2) *Calcularea modulului rezultatului.* Pentru aceasta se inmultesc modulele celor doi operanzi. Aici se disting doua cazuri:

a) *Numere intregi.*

$$|x| \cdot |y| = |x| \cdot \sum_{k=0}^{n-2} y_k \cdot 2^k = \sum_{k=0}^{n-2} |x| \cdot y_k \cdot 2^k$$

Termenul general al sumei reprezinta un produs partial si este:

$$|x| \cdot y_k \cdot 2^k = \begin{cases} 0 & daca\ y_k = 0 \\ |x| \cdot 2^k & daca\ y_k = 1 \end{cases}$$

sau, $|x|2^k$ reprezinta $|x|$ deplasat spre stanga k pozitii.

b) *Numere subunitare.*

$$|x| \cdot |y| = |x| \cdot \sum_{k=1}^{n-1} y_{-k} \cdot 2^{-k} = \sum_{k=1}^{n-1} |x| \cdot y_{-k} \cdot 2^{-k}$$

Termenul general al sumei reprezinta un produs partial si este:

$$|x| \cdot y_{-k} \cdot 2^{-k} = \begin{cases} 0 & daca\ y_k = 0 \\ |x| \cdot 2^{-k} & daca\ y_k = 1 \end{cases}$$

sau, $|x|2^{-k}$ reprezinta $|x|$ deplasat spre dreapta k pozitii.

3) *Trunchierea si rotunjirea rezultatului.* Rezultatul inmultirii se obtine pe lungime dubla si pentru a-l reprezenta pe lungime simpla (la fel ca cei doi operanzi care s-au inmultit) este necesar sa se trunchieze rezultatul prin neglijarea celor n-1 biti c.m.p.s. in cazul numerelor subunitare sau prin neglijarea celor n-1 biti c.m.s. in cazul numerelor intregi (se verifica daca a aparut depasire). Pentru a obtine un rezultat cat mai aproape de cel corect la numerele subunitare se poate efectua si o rotunjire dupa urmatoarea regula: daca bitul c.m.s. care se indeparteaza la trunchiere este 1, atunci se aduna o unitate in rangul c.m.p.s. care ramane.

Exemplu. Sa se efctueze inmultirea $x \bullet y = z$ in virgula fixa, unde x =20/32 si y = -19/32.

Asa cum s-a discutat la inceputul acestui capitol numerele subunitare de cele mai multe ori nu se convertesc exact dintr-o baza in alta (din baza 10 in baza 2). Pentru a elimina eroarea de conversie, in acest exemplu (ca si in cele care urmeaza) s-au ales numere subunitare care se pot converti exact din zecimal in binar: este cazul numerelor care se pot scrie sub forma unei sume de puteri negative ale lui 2. Chiar daca punctul zecimal nu se

reprezinta explicit in calculator in cadrul exemplelor cu numere subunitare se va scrie si punctul zecimal, pentru mai multa claritate.

```
[x] = 0.10100
[y] = 1.10011
```

Se parcurg etapele prezentate mai sus:

1) Semnul rezultatului:
$z_s = x_s \oplus y_s = 0 \oplus 1 = 1$

2) Modulul rezultatului:
|z| = |x| • |y|

```
      .10100•
      .10011
  ____________
.0000010100+   |x|•y-5•2^-5=|x|•2^-5
.0000101000    |x|•y-4•2^-4=|x|•2^-4
.0000000000    |x|•y-3•2^-3=0
.0000000000    |x|•y-2•2^-2=0
.0101000000    |x|•y-1•2^-1=|x|•2^-1
  ____________
.0101111100
```

Acesta este rezultatul exact si tinand cont si de bitul de semn rezulta [z] = 1.0101111100. Valoarea in zecimal se obtine cu relatia:

$z = - (0•2^{-1}+1•2^{-2}+0•2^{-3}+1•2^{-4}+1•2^{-5}+1•2^{-6}+1•2^{-7}+1•2^{-8}+0•2^{-9}+0•2^{-10}) =$
$= - (0•2^{9}+1•2^{8}+0•2^{7}+1•2^{6}+1•2^{5}+1•2^{4}+1•2^{3}+1•2^{2}+0•2^{1}+0•2^{0})/2^{10} =$
$= - (256+64+32+16+8+4)/1024 = - 380/1024$

(rezultat corect !)

3) Trunchierea si rotunjirea modulului rezultatului:

```
.01011 11100
    ↓
.01011+
     1
  _______
.01100
```

Rezultatul aproximativ este [z] = 1.01100, deci z = -12/32 (s-a obtinut rezultatul -0.375, fata de cel exact 0.37109375).

2.5 Impartirea in virgula fixa

Metodele directe de impartire in virgula fixa sunt complicate si nu sunt implementate in unitatile aritmetice. In practica se utilizeaza anumite metode eficiente, cum sunt: metoda comparatiei, metoda refacerii restului partial, metoda fara refacerea restului partial. In continuare se va prezenta pe scurt metoda comparatiei.

Metoda comparatiei

Aceasta metoda se aplica pentru numere reprezentate in virgula fixa, cand se trateaza separat semnele si separat modulele operanzilor. Pentru efectuarea operatiei de impartire este necesar ca deimpartitul < impartitorul. Se considera operatia $x : y$, care furnizeaza catul q si restul r. Cei n-1 biti ai modulului fiecarui operand se noteaza cu indici negativi –1, –2, –3, ... $–m$ ($m = n–1$), pentru ai pune in corespondenta cu puterile negative ale bazei 2 reprezentand ponderile. Algoritmul este descris in continuare, in limbaj pseudocod:

```
citeste x,y
daca |x| ≥ |y| atunci
|     scrie „Eroare!”
altfel
|     q_s = x_s ⊕ y_s
|     r_s = x_s
|     |r| =|x|  //initializare rest partial
|     pentru i=1,m executa
|     |     |r| = |r|•2  //deplaseaza o pozitie stanga
|     |     daca |r| ≥ |y| atunci
|     |     |     q_-i = 1  //bitul curent al catului
|     |     |     |r| = |r| – |y|
|     |     altfel
|     |     |     q_-i = 0  //bitul curent al catului
|     |     |     |r| = |r|
|r| = |r|•2^-m  //rest final
scrie q, r
```

Se observa ca pentru furnizarea restului final s-a efectuat o corectie reprezentata printr-o inmultire cu 2^{-m}. Acest lucru rezulta din urmatoarele considerente (s-a notat prin $r^{(k)}$ restul partial la pasul k):

$$
\begin{aligned}
&|r^{(0)}| = |x| \\
&|r^{(1)}| = 2\bullet|r^{(0)}| - q_{-1}\bullet|y| \\
&|r^{(2)}| = 2\bullet|r^{(1)}| - q_{-2}\bullet|y| \\
&\ldots\ldots\ldots \\
&|r^{(m)}| = 2\bullet|r^{(m-1)}| - q_{-m}\bullet|y|
\end{aligned}
$$

unde $q_{-k} = \begin{cases} 1 & daca\ 2\cdot\left|r^{(k-1)}\right| \geq |y| \\ 0 & altfel \end{cases}$. In ultima relatie de mai sus se inmultesc ambii membri cu 2^{-m}, inlocuindu-se succesiv fiecare $r^{(k)}$ din relatia precedenta in functie de $r^{(k-1)}$. Se obtine:

$$
\begin{aligned}
&2^{-m}\bullet|r^{(m)}| = \\
&\quad = -2^{-m}\bullet q_{-m}\bullet|y| + 2^{-m}\bullet 2\bullet(2\bullet\ldots\bullet(2\bullet|x| - q_{-1}\bullet|y|)\ldots - q_{-m+1}\bullet|y|) = \\
&\quad = 2^{-m}\bullet 2^{m}\bullet|x| - |y|\bullet(q_{-1}\bullet 2^{-1} + q_{-2}\bullet 2^{-2} + \ldots + q_{-m}\bullet 2^{-m}) = \\
&\quad = |x| - |y|\bullet|q|
\end{aligned}
$$

deci, s-a obtinut relatia de baza de la impartire:

$$|x| = |y|\bullet|q| + 2^{-m}\bullet|r^{(m)}|$$

ceea ce inseamna ca restul corect este $2^{-m}\bullet|r^{(m)}|$.

Exemplu. Sa se efectueze impartirea $x : y$, unde $x = 20/32$ si $y = 25/32$.

Se reprezinta cei doi operanzi pe 6 biti:

$$
\begin{aligned}
[x] &= 0.10100 \\
[y] &= 0.11001
\end{aligned}
$$

Impartirea se executa in cinci (n-1) pasi (1,2,...5):

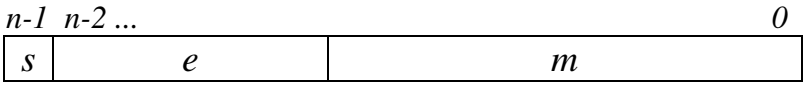
Pas	$\lvert y\rvert$	$\lvert r^{(k)}\rvert$	q
0	.11001	.10100	$\lvert r^{(0)}\rvert < \lvert y\rvert$ => OK!
1	.11001	1.01000 .11001 .01111	$\lvert r^{(1)}\rvert \geq \lvert y\rvert$ => $q_{-1}=1$

2	.11001	.11110- .11001 .00101	$\|r^{(2)}\| \geq \|y\|$	=> $q_{-2}=1$
3	.11001	.01010	$\|r^{(3)}\| < \|y\|$	=> $q_{-3}=0$
4	.11001	.10100	$\|r^{(4)}\| < \|y\|$	=> $q_{-4}=0$
5	.11001	1.01000- .11001 .01111	$\|r^{(5)}\| \geq \|y\|$	=> $q_{-5}=1$

S-a obtinut urmatorul rezultat: $[q]$ = 0.11001 (q = 25/32) si $[r]$ = 0.0000001111 (r = 15/1024). Se verifica: $x = q \bullet y + r$ (20/32 = 25/32 • 25/32 + 15/1024).

2.6 Reprezentarea numerelor in virgula mobila

Reprezentarea in virgula mobila (virgula flotanta) este utilizata pentru numere reale, permitand o precizie ridicata si o plaja larga de valori. Un numar este reprezentat in virgula mobila printr-un cuvant de n biti, continand urmatoarele campuri:



unde s este bitul de semn al numarului, cu aceeasi conventie ca pentru numere in virgula fixa (s=0 numar pozitiv si s=1 numar negativ), m este mantisa numarului, reprezentand cifrele semnificative, iar e este exponentul (puterea la care trebuie ridicata o valoare numita baza si care inmulteste cifrele semnificative ale numarului). Valoarea numarului real reprezentat in virgula mobila este data de expresia:

$$\text{valoare} = (-1)^s \bullet m \bullet \text{baza}^e$$

Ca baza se utilizeaza valorile 2, 10 sau 16 (baza 2 este cea mai utilizata).

Mantisa trebuie sa indeplineasca conditia: $1 > m \geq 1/\text{baza}$, fiind un numar subunitar in virgula fixa, reprezentat in cod complementar. Pentru

baza 2, mantisa indeplineste conditia $1 > m \geq ½$, ceea ce inseamna ca primul bit, c.m.s. este 1.

Exponentul este un numar intreg cu semn, ceea ce complica algoritmii de calcul in virgula mobila. In practica se reprezinta in locul exponentului o valoare modificata, numita caracteristica, care este o valoare fara semn si permite conservarea plajei de valori, conform relatiei:

$$\text{caracteristica} = \text{exponent} + 2^{\text{numar de biti exponent} - 1}$$

Exemplu. Se considera pentru o reprezentare in virgula mobila 7 biti pentru exponent, ceea ce inseamna ca se pot reprezenta pentru acesta 128 valori distincte cu semn:

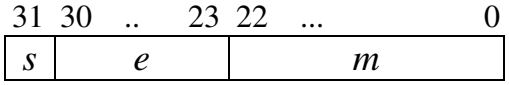
$$-64 \leq \text{exponent} \leq +63$$

Reprezentand in locul exponentului caracteristica, valoare intreaga fara semn, plaja de valori contine tot 128 de valori, astfel:

$$\text{caracteristica} = \text{exponent} + 2^{7-1}$$

$$\text{caracteristica} = \text{exponent} + 64 \Rightarrow 0 \leq \text{caracteristica} \leq 127$$

Exemplu. Standardul IEEE 754 (IEEE – „Institute of Electrical and Electronics Engineers") pentru reprezentarea numerelor reale in virgula mobila pe lungime (precizie) simpla (32 de biti):



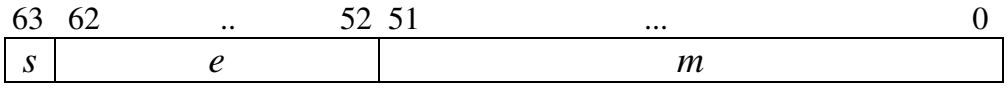
Valoarea numarului fiind data de relatiile:

daca $0 < e < 255 \Rightarrow$ valoarea $= (-1)^s \bullet 1.m \bullet 2^{e-127}$

daca $e = 0$ si $m = 0 \Rightarrow$ valoarea $= 0$

daca $e = 0$, $m \neq 0$ sau $e = 255 \Rightarrow$ eroare

Reprezentarea pe lungime (precizie) dubla (64 de biti):



Valoarea numarului fiind data de relatiile:

daca $0 < e < 2047$ => valoarea = $(-1)^s \bullet 1.m \bullet 2^{e-1023}$
daca $e = 0$ si $m = 0$ => valoarea = 0
daca $e = 0$, $m \neq 0$ sau $e = 2047$ => eroare

2.7 Metode eficiente de impartire

In practica se utilizeaza metode eficiente de impartire in virgula mobila, dintre care vor fi prezentate metoda inversarii impartitorului si metoda factorilor succesivi.

Metoda inversarii impartitorului

Se considera ecuatia *f(x)*=0. Pentru gasirea unei radacini a ecuatiei se poate utiliza metoda iterativa Newton-Raphson: daca functia *f* este continua si derivabila pe un interval care include radacina cautata, aceasta poate fi gasita utilizand relatia de recurenta de mai jos, pornind de la o valoare initiala x_0 bine aleasa:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Daca se alege functia $f(x) = \frac{1}{x} - w$, cu radacina 1/*w*, $f'(x) = -\frac{1}{x^2}$, iar relatia de recurenta devine:

$$x_{i+1} = x_i - \frac{1/x_i - w}{-1/x_i^2} = x_i \cdot (2 - w \cdot x_i)$$

relatie utilizata pentru a calcula pe 1/*w*. Astfel operatia *A:B* se poate transforma in operatia *A•(*1*/B)*, ceea ce necesita aflarea lui 1/*B* prin metoda iterativa de mai sus. Sunt necesare resurse rapide pentru operatiile de inmultire si scadere. Un exemplu de circuit care implementeaza aceasta metoda este Advanced Micro Devices AM29C325.

Metoda factorilor succesivi

Este utilizata pentru calcularea catului mantiselor (*Q=D:*I) de la impartirea in virgula mobila. Se poate genera direct rezultatul impartirii

prin inmultiri succesive atat la numarator cat si la numitor cu o serie de factori bine alesi $f_0, f_1,..., f_k$,

$$Q = \frac{D}{I} = \frac{D \cdot f_0 \cdot f_1 \cdot f_2 \cdot f_3 \cdot ....}{I \cdot f_0 \cdot f_1 \cdot f_2 \cdot f_3 \cdot ....}$$

astfel incat numitorul fractiei sa tinda catre 1, iar in acest caz rezultatul impartirii este egal cu numaratorul fractiei. Se noteaza $I=1-x$, unde x este subunitar deoarece si I este subunitar, considerat fara semn. Rezulta:

$$f_0 = 1 + x = 1 + (1 - I) = 2 - I$$
$$I \bullet f_0 = (1 - x) \bullet (1 + x) = 1 - x^2$$

care este mai aproape de 1 decat I, iar f_0 este complementul fata de 2 al lui I.

$$f_1 = 1 + x^2 = 1 + (1 - I \bullet f_0) = 2 - I \bullet f_0$$
$$I \bullet f_0 \bullet f_1 = (1 - x^2) \bullet (1 + x^2) = 1 - x^4$$

care este mai aproape de 1 decat $I \bullet f_0$, iar f_1 este complementul fata de 2 al lui $I \bullet f_0$.... Astfel fiecare f_k se obtine din complementul fata de 2 al valorii curente a numitorului. De obicei se considera un numar fix de pasi. Pentru a asigura convergenta mai buna a algoritmului in locul factorului initial f_0 se considera o valoare modificata, generata dintr-o memorie ROM. Schema bloc a unei unitati de impartire care utilizeaza aceasta metoda este prezentata in figura urmatoare (fig.2.10.1):

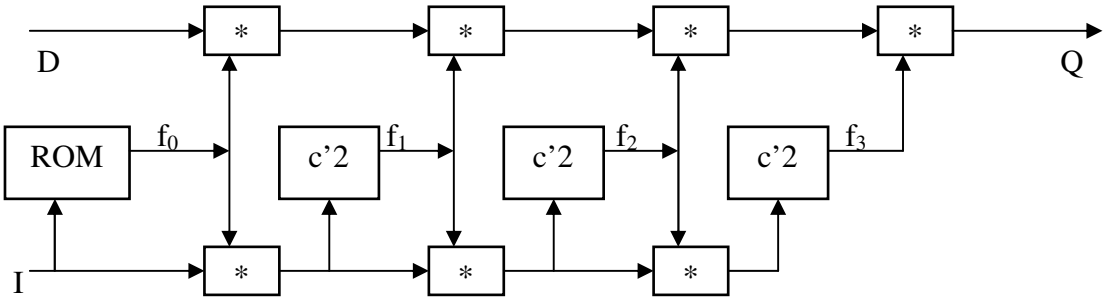


Fig.2.10.1 Unitate de impartire prin metoda factorilor succesivi.

In cadrul schemei s-a notat prin (*) unitate rapida de inmultire si prin (c'2) unitate de complementare fata de 2.

2.8 Alte coduri numerice

In sistemele digitale (calculatoare si alte dispozitive numerice) se utilizeaza si alte coduri pentru reprezentarea numerelor, pe langa cele prezentate mai inainte.

Codul BCD

Codul BCD (”Binary Coded Decimal”) sau ZCB (zecimal codificat binar) utilizeaza 4 biti pentru reprezentarea explicita a fiecarei cifre zecimale, conform tabelului urmator:

cifra	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

De remarcat faptul ca grupurile de biti 1010,1011,...,1111 nu reprezinta cifre BCD corecte. In anumite dispozitive numerice se utilizeaza echipamente de introducere de valori prin cifre zecimale, care sunt reprezentate in BCD, iar apoi convertite in binar pentru a fi prelucrate (caci algoritmii de calcul in binar sunt mai eficienti). Rezultatele binare obtinute in urma prelucrarilor sunt convertite din nou in BCD pentru a fi afisate in exterior (de exemplu pe dispozitive cu 7 segmente). Rezulta astfel in unele aplicatii necesitatea conversiilor din BCD in binar si din binar in BCD.

In continuare va fi discutata pe scurt conversia din binar in BCD. Algoritmul de conversie rezulta din echivalenta dintre o deplasare la stanga a unei valori binare si inmultirea cu 2. Astfel, se considera doua grupuri de biti, corespunzatoare la doua registre in cazul unei implementari hardware: grupul binar si grupul BCD (impartit in decade de cate 4 biti corespunzatoare cifrelor zecimale). Se initializeaza grupul binar cu valoarea binara de convertit, iar grupul BCD cu 0. Se executa deplasari succesive cu cate o pozitie spre stanga atat in grupul binar cat si in grupul BCD, cu trecerea cate unui bit din grupul binar in grupul BCD, incepand cu bitul c.m.s. Inainte de fiecare deplasare se executa corectii in acele decade din grupul BCD care au o valoare > 4 (prin deplasare stanga, echivalent cu o inmultire cu 2, se obtine o valoare > 9, deci care nu este cifra BCD). Corectia consta din adunarea valorii 3 in fiecare decada cu valoare > 4

inainte de deplasare. Conversia se incheie in momentul in care toti bitii din grupul binar au fost transferati in grupul BCD.

Necesitatea conversiei rezulta din urmatoarele considerente:

a) Daca decada $\in \{0, 1, 2, 3, 4\}$ atunci prin deplasare stanga o pozitie se obtine tot cifra BCD, deci nu este necesara nici o corectie.

b) Daca decada $\in \{5, 6, 7\}$ prin deplasare stanga rezulta:

$$\text{decada} \bullet 2 = (5 + (\text{decada} - 5)) \bullet 2 = 10 + (\text{decada} - 5) \bullet 2$$

ceea ce inseamna ca se scade 5 din decada inainte de deplasare (operatia este in interiorul parantezei care se inmulteste cu 2) si se aduna 10 dupa deplasare, sau o unitate la decada urmatoare (operatie echivalenta cu fortarea in 1 a bitului c.m.s. al decadei curente, sau adunarea valorii 8, inainte de deplasare). Deci s-a scazut 5 si s-a adunat 8 la decada curenta, echivalent cu adunarea valorii 3 inainte de deplasare.

c) Daca decada $\in \{8, 9\}$ prin deplasare stanga bitul c.m.s. de pondere 8 va trece in pozitia c.m.p.s. din decada urmatoare. Valoarea relativa la decada curenta a acestui bit creste de la 8 la 10 prin deplasare (inmultire cu 2), in loc sa creasca de la 8 la 16. Corectia se face prin adunarea valorii 6 dupa deplasare sau prin adunarea valorii 3 inainte de deplasare la decada curenta.

Exemplu. Se considera numarul in binar $N_{(2)} = 111110010$. Sa se converteasca in BCD prin metoda deplasarilor succesive prezentata mai sus.

```
   Grup BCD          Grup binar
0000 0000 0000       111110010
0000 0000 0001       11110010
0000 0000 0011       1110010
0000 0000 0111+      110010
            11
           ____
0000 0000 1010       110010
0000 0001 0101+      10010
            11
           ____
0000 0001 1000       10010
0000 0011 0001       0010
0000 0110+0010       010
       11
      ____
0000 1001+0010       010
0001 0010 0100       10
0010 0100 1001+      0
            11
```

```
          ____
0010 0100 1100        0
0100 1001 1000
```

S-a obtinut rezultatul $498_{(10)}$ in BCD pe 3x4 biti.

Conversia BCD - binar are la baza echivalenta dintre deplasarile succesive spre dreapta si impartirile la 2. In acest caz se initializeaza grupul BCD cu valoarea BCD de convertit, iar grupul binar cu 0. Se executa deplasari succesive spre dreapta cu o pozitie atat in grupul BCD cat si in grupul binar, efectuand corectii la acele decade in care, dupa deplasare, se obtine bitul c.m.s egal cu 1. Corectia consta in scaderea valorii 3 (echivalent cu adunarea complementului sau, 13) din fiecare decada cu valoarea ≥ 8 (care a primit prin deplasare in pozitia c.m.s un bit 1). Necesitatea corectiei rezulta din faptul ca prin deplasare la dreapta cand un bit 1 trece din pozitia c.m.p.s. a unei decade in pozitia c.m.s. a decadei inferioare alaturate, valoarea sa, relativ la decada inferioara, scade de la 10 la 8, in loc sa scada de la 10 la 5 (prin impartire la 2). Deci se executa corectia la decada inferioara in care a aparut bitul 1 in pozitia c.m.s., scazand valoarea 3 (sau adunand 13).

Exemplu. Sa se reprezinte in BCD numarul $N_{(10)}$=503 si sa se converteasca in binar prin metoda deplasarilor succesive.

```
   Grup BCD           Grup binar
0101 0000 0011        000000000
0010 1000+0001        1
     1101
     ____
0010 0101 0001        1
0001 0010 1000+       11
          1101
          ____
0001 0010 0101        11
0000 1001+0010        111
     1101
     ____
0000 0110 0010        111
0000 0011 0001        0111
0000 0001 1000+       10111
          1101
          ____
0000 0001 0101        10111
0001 0000 1010+       110111
          1101
          ____
```

```
0000 0000 0111        110111
          .......  (nu se modifica configuratia de biti)
0000 0000 0000        111110111
```

S-a obtinut valoarea $N_{(2)} = 111110111$ ($503_{(10)}$).

Codul Gray

Codul Gray are proprietatea ca oricare doua valori succesive (consecutive) difera prin valoarea unui singur bit. Acest cod este util in acele aplicatii unde o eroare de un bit este acceptabila si nu modifica foarte mult valoarea numerica respectiva. In tabelul urmator se prezinta echivalenta dintre codurile binar si Gray pe 4 biti:

binar	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Coduri nenumerice

Intr-un sistem de calcul o parte a informatiei prelucrate este nenumerica. Pentru reprezentarea acesteia se utilizeaza diferite coduri, dintre care cel mai utilizat este codul ASCII (”American Standard Code for Information Interchange”). Codul ASCII standard utilizeaza 7 biti pentru reprezentarea literelor mari si mici ale alfabetului latin, cifrelor zecimale, caracterelor speciale si a cateva caractere de control (in total 128 de

caractere). Caracterele sunt grupate astfel (codurile fiind furnizate in hexazecimal):

0-1Fh : caractere de control, ca LF (0Ah), CR (0Dh), BEL (07h),...;

20h-2Fh : caractere speciale, ca spatiu, !, ”, #, $,...;

30h-39h : cifrele zecimale 0(30h), 1(31h),...,9(39h);

3Ah-40h : caractere speciale :, ;, <, =, ...;

41h-5Ah : literele mari ale alfabetului A(41h), B(42h), ..., Z(5Ah);

5Bh-60h : caractere speciale [, \, ...;

61h-7Ah : literele mici ale alfabetului a(61h), b(62h), ..., z(7Ah);

7Bh-7Fh : caractere de control si speciale, cum sunt DEL(7Fh), {(7Bh), }(7Dh),...

Pe langa acest cod se utilizeaza si codul ASCII extins, in care caracterele sunt reprezentate pe 8 biti (deci se reprezinta in total 256 de caractere).