1. Bazele aritmetice al calculatoarelor numerice

1.1. Sisteme de numeraţie

Un *sistem de numeraţie (SN)* este format din totalitatea regulilor de reprezentare a numerelor cu ajutorul unor simboluri numite *cifre*.

SN sunt de două tipuri: poziţionale şi nepoziţionale. Pentru un sistem poziţional ponderea unei cifre este dată atât de valoarea ei intrinsecă cât şi de poziţie. Un sistem nepoziţional, este acela în care ponderea nu este influenţată de poziţia cifrei.

Datorită simplităţii de reprezentare şi efectuare a calculelor, în sistemele numerice se folosesc în exclusivitate sistemele poziţionale, un asemenea sistem fiind caracterizat prin *bază* care reprezintă numărul total de simboluri (cifre).

Exemple de baze uzuale:

Sistemul zecimal: *b=10*, simboluri: *0,1,2,3,4,5,6,7,8,9*;

Sistemul binar: *b=2*, simboluri: *0,1*;

Sistemul octal: *b=8*, simboluri: *0,1,2,3,4,5,6,7*;

Sistemul hexazecimal: *b=16*, simboluri: *0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F*.

Pentru un număr întreg $N \geq 0$, reprezentarea în baza b este secvenţa de simboluri $x_{m-1}\ x_{m-2} \ldots x_2\ x_1\ x_0$ care verifică următoarele condiţii:

a. $$0 \leq x_i < b,\ \ i = m-1, m-2, \ldots, 2, 1, 0;\ \ x_{m-1} \neq 0; \qquad (1.1)$$

b. $$N = x_{m-1}b^{m-1} + x_{m-2}b^{m-2} + \ldots + x_2 b^2 + x_1 b^1 + x_0 b^0. \qquad (1.2)$$

Numerele reale au o reprezentare asemănătoare, însă conţin punctul fracţionar (sau virgula) care separă partea întreagă de cea fracţionară.

Pentru un număr real $r \geq 0$, reprezentarea în baza b este secvenţa de simboluri $x_{m-1} \ldots x_1\ x_0 . x_{-1}\ x_{-2} \ldots$ care verifică următoarele condiţii:

a. $$0 \leq x_i < b,\ \ i = m-1, m-2, \ldots, 2, 1, 0, -1, -2, \ldots;\ \ x_{m-1} \neq 0; \qquad (1.3)$$

b. $$r = x_{m-1}b^{m-1} + \ldots + x_2 b^2 + x_1 b^1 + x_0 b^0 + x_{-1} b^{-1} + x_{-2} b^{-2} + \ldots. \qquad (1.4)$$

Pornind de la faptul că la baza realizării unui sistem numeric de calcul stau dispozitivele cu două stări stabile, rezultă că SN binar (care necesită numai două cifre, *0* şi *1*) este cel mai potrivit pentru prelucrarea, codificarea şi transmiterea informaţiei în aceste echipamente. SN ale căror baze reprezintă puteri ale lui 2 prezintă de asemenea proprietăţile sistemului binar, motiv pentru care sunt frecvent utilizate în tehnica de prelucrare automată a datelor (în special SN octal şi SN hexazecimal). În ceea ce priveşte SN zecimal acesta este cu precădere utilizat în anumite faze ale operaţiilor de intrare- ieşire.

1.2. Conversia unui număr dintr-o bază în alta

Existenţa şi utilizarea mai multor SN ridică problema conversiei dintr-un sistem în altul. Una din metodele frecvent utilizate o reprezintă împărţirea / înmulţirea numărului cu noua bază.

- *Vom examina pentru început conversia numerelor întregi.*

Fie numărul N exprimat în baza α :

$$(N)_\alpha = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + \ldots + a_1\alpha^1 + a_0\alpha^0, \tag{1.5}$$

pentru care se doreşte exprimarea în baza β, respectiv

$$(N)_\alpha = x_{n-1}\beta^{n-1} + x_{n-2}\beta^{n-2} + \ldots + x_1\beta^1 + x_0\beta^0, \tag{1.6}$$

Algoritmul de conversie presupune determinarea coeficienţilor x_i , una din metode fiind cea a *împărţirilor succesive a numărului la noua bază,* conform următorului procedeu:

$$(N)_\alpha/\beta = \underbrace{x_{n-1}\beta^{n-2} + x_{n-2}\beta^{n-3} + \ldots + x_1\beta^0}_{\text{Câtul}\,(N_1)_\alpha} + \underbrace{x_0/\beta}_{\text{Restul}} \Rightarrow x_0\,, \tag{1.7}$$

$$(N_1)_\alpha/\beta = \underbrace{x_{n-1}\beta^{n-3} + x_{n-2}\beta^{n-4} + \ldots + x_2\beta^0}_{\text{Câtul}\,(N_2)_\alpha} + \underbrace{x_1/\beta}_{\text{Restul}} \Rightarrow x_1\,, \tag{1.8}$$

……..

$$(N_k)_\alpha/\beta = \underbrace{x_{n-1}\beta^{n-k-2} + \ldots + x_{k+1}\beta^0}_{\text{Câtul}(N_{k+1})_\alpha} + \underbrace{x_k/\beta}_{\text{Restul}} \Rightarrow x_k\,, \tag{1.9}$$

…….

După cum se observă în urma primei împărţiri rezultă x_0, (*cifra cea mai puţin semnificativă a rezultatului - CCMPS*), apoi x_1 ş.a.m.d. Procedeul continuă până când câtul devine mai mic decât noua bază β, ultimul rest fiind coeficientul x_{n-1} , (*cifra cea mai semnificativă a rezultatului - CCMS*).

- *În cazul numerelor reale de forma* $r = r_I + r_S$ partea întreagă r_I se converteşte potrivit procedurii prezentate, iar partea subunitară r_S prin înmulţiri succesive cu noua bază.

Fie numărul r_S exprimat în baza α :

$$(r_S)_\alpha = a_{-1}\alpha^{-1} + a_{-2}\alpha^{-2} + \ldots + a_{-m}\alpha^{-m}, \tag{1.10}$$

pentru care se doreşte exprimarea în baza β , respectiv

$$(r_S)_\alpha = x_{-1}\beta^{-1} + x_{-2}\beta^{-2} + \ldots + a_{-n}\alpha^{-n} + \ldots, \tag{1.11}$$

Coeficienţii x_{-i} se obţin prin înmulţiri succesive în (1.11) cu β, respectiv

$$(r_S)_\alpha \cdot \beta = x_{-1} + \underbrace{x_{-2}\beta^{-1} + \ldots + a_{-n}\beta^{-n+1} + \ldots}_{(r_{S1})_\alpha}, \Rightarrow x_{-1}; \tag{1.12}$$

$$(r_{S1})_\alpha \cdot \beta = x_{-2} + \underbrace{x_{-3}\beta^{-1} + \ldots + a_{-n}\beta^{-n+2} + \ldots}_{(r_{S2})_\alpha}, \Rightarrow x_{-2}; \quad (1.13)$$

……..

$$(r_{S\,k-1})_\alpha \cdot \beta = x_{-k} + \underbrace{x_{-k-1}\beta^{-1} + \ldots + a_{-n}\beta^{-n+k} + \ldots}_{(r_{S\,k})_\alpha}, \Rightarrow x_{-k}; \quad (1.14)$$

…….

Cifra x_{-1} reprezintă *CCMS* iar x_{-k} *CCMPS* a rezultatului. Procedura se opreşte în momentul partea fracţionară $r_{S\,k}$ a rezultatului unei înmulţiri devine zero, sau dacă s-a atins precizia de conversie specificată prin numărul de cifre al reprezentării.

Din prezentarea efectuată rezultă că numărul de poziţii binare pentru partea întreagă se poate determina apriori, în timp ce pentru partea fracţionară acest număr trebuie impus. Astfel numărul *n* de poziţii binare care se obţin la conversia unui număr zecimal *N* este:

$$n = \begin{cases} \log_2 N + 1 & \text{dacă } N = 2^k,\ k \in N \\ [\log_2 N] + 1 & \text{dacă } N \neq 2^k, k \in N \end{cases} \quad (1.15)$$

- *Conversiile binar - octal ; binar - hexazecimal* se realizează simplu datorită faptului că bazele acestora sunt puteri ale lui 2 aspect evidenţiat în tabelul 1.1.

Tabelul 1.1

Octal	Binar	Hexaze cimal	Binar	Hexaze cimal	Binar
0	000	0	0000	8	1000
1	001	1	0001	9	1001
2	010	2	0010	A	1010
3	011	3	0011	B	1011
4	100	4	0100	C	1100
5	101	5	0101	D	1101
6	110	6	0110	E	1110
7	111	7	0111	F	1111

Cifrele sistemului octal pot fi reprezentate prin combinaţii de câte trei biţi denumite *triade* iar ale sistemului hexazecimal prin combinaţii de câte patru biţi numite *tetrade.*

Regula de conversie: fiind dată reprezentarea în binar a unui număr real, reprezentarea în octal se obţine grupând câte trei cifrele binare începând de la separatorul fracţionar (punct, virgulă), spre stânga şi spre dreapta. După ce grupele extreme se completează (dacă este cazul cu zerouri nesemnificative), fiecare triadă se substituie cu echivalentul său octal. Aceiaşi regulă se aplică şi la conversia *binar-hexazecimal,* cu observaţia că în loc de *triade* se operează cu combinaţii de 4 biţi (*tetrade).*

Conversia inversă *octal / hexazecimal* $\rightarrow$ *binar* se face prin înlocuirea fiecărei cifre *octale / hexazecimale* cu *triada / tetrada* corespunzătoare.

1.3. Reprezentarea numerelor în calculator

Uzual un echipament de calcul numeric preia datele şi oferă rezultatele într-o formă accesibilă utilizatorului. În ceea ce priveşte prelucrarea, aceasta presupune exprimarea datelor într-o formă specifică procesorului. În consecinţă există două formate de reprezentare a numerelor în calculator şi anume *formatul intern* şi *formatul extern.*

a) *Reprezentarea numerelor întregi* . Acestea se reprezintă de regulă în format cu virgulă fixă. De la început trebuie făcută precizarea că virgula (punctul zecimal) nu este prezentă în mod explicit în reprezentare. Sintagma *virgulă fixă* are în vedere o convenţie care presupune un loc fix al acesteia şi un acelaşi număr de cifre pentru număr.

Numărul de biţi (n) utilizaţi pentru o reprezentare determină numărul de valori reprezentabile (2^n). În tabelul 2.2 sunt prezentate valori tipice pentru n şi 2^n.

Domeniul finit de valori pentru numerele întregi, reprezentate în formatul cu virgulă fixă este:

$$D = [V_{\min}, V_{\max}] \cap Z , \tag{1.16}$$

unde V_{min} *şi* V_{max} sunt cea mai mică respectiv cea mai mare valoare ce se pot reprezenta pe n biţi. Cele 2^n valori distincte pot constitui reprezentări ale unor numere întrgi *pozitive sau negative.*

Observaţie importantă.

Indiferent de format numărul de biţi pe care se reprezintă un număr este *finit şi fix,* stabilindu-se în faza de proiectare a calculatorului. Din acest motiv în calculator nu se poate reprezenta decât un număr finit de valori, interpretate diferit în cele două formate. *Numerele care se pot reprezenta în calculator se numesc* ***numere cu precizie finită (NPF)*** *şi au proprietăţi diferite faţă de numerele din matematică.*

În cazul acestora poate apărea depăşirea capacităţii de memorare deoarece *NPF* au un domeniu finit de valori, în sensul că nu pot exista numere oricât de mari sau oricât de mici. Depăşirile pot fi detectate hardware sau software.

Numerele întregi fără semn se reprezintă prin corespondentul lor binar *(codul direct)* numărul de biţi pentru reprezentarea unui număr N fiind

$$2^{n-1} \le N < 2^n . \tag{1.17}$$

Pentru numerele cu semn există trei reprezentări mai des utilizate şi anume:

a1semn – mărime;
a2complement faţă de 1;
a3complement faţă de 2.

a1) Reprezentarea numerelor în semn mărime

Considerând reprezentarea pe *n+1* biţi primul bit din stânga este asociat *semnului*, iar restul de *n* biţi conţin *mărimea* numărului egală cu reprezentarea binară a valorii $|N|$- figura 1.1.

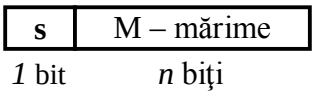

Fig. 1.1. Reprezentarea *semn – mărime* pe *n+1* biţi.

În privinţa semnului *s* care ocupă bitul *cel mai semnificativ* (***MSB –Most Significant Bit***) convenţia este următoarea:

$S = 0$, dacă $N \geq 0$;

$S = 1$, dacă $N < 0$.

a2) Reprezentarea numerelor în complement faţă de 1

Pentru numere pozitive $N \geq 0$ reprezentarea în complement faţă de *1* este identică cu reprezentarea *semn – mărime.*

Dacă N < 0, atunci s = 1 şi mărimea

$$M = C1(|N|) = 2^n - 1 - |N|, \tag{1.18}$$

unde *n este* numărul de biţi utilizaţi pentru reprezentarea mărimii.

Calculul complementului *C1* se poate face prin două metode şi anume:

1- utilizând definiţia, (1.18) respectiv

$S = 1 \quad M = 2^n - 1 - |N|.$

2 - prin *inversarea biţilor* reprezentării cu semn a valorii absolute $|N|$ a numărului *N.*

a3) Reprezentarea numerelor în complement faţă de 2

Pentru numere pozitive $N \geq 0$ reprezentarea în complement faţă de 2 este identică cu reprezentările *semn – mărime* şi în *cod complementar faţă de 1.*

Dacă $N < 0$, atunci $s = 1$ şi mărimea

$$M = C2(|N|) = 2^n - |N|, \tag{1.19}$$

unde *n este* numărul de biţi utilizaţi pentru reprezentarea mărimii.

Calculul *C2* se poate face prin trei metode şi anume:

1- *utilizând definiţia,* respectiv

$S = 1 \quad M = 2^n - |N|.$ *s=1.*

2 - prin *inversarea biţilor* reprezentării cu semn a valorii absolute $|N|$ a numărului *N* la care se adaugă *1*;

3 - prin *analiza* de la dreapta la stânga a reprezentării cu semn a valorii absolute $|N|$ a numărului *N*. Primii biţi de *0* şi primul bit de *1* se lasă neinversaţi, apoi se inversează toţi biţii, inclusiv bitul de semn (dacă primul bit întâlnit este *1* se lasă neschimbat numai acesta).

Observaţie

Reprezentarea în *complement faţă de 2* este cea mai utilizată pentru numerele algebrice datorită faptului că elimină ambiguităţile legate de reprezentarea numărului zero.

Reprezentările *semn – mărime şi C1* oferă două reprezentări distincte ale acestui număr ***(+0 şi -0)*** după cum urmează.

Semn - mărime	*C1*	*C2*
+0 = 0000 0000	*+0 = 0000 0000*	*+0 = 0000 0000*
- 0 = 1000 0000	*- 0 = 1111 1111*	*-0 = 0000 0000*

b. <u>*Reprezentarea numerelor reale.*</u> Numerele reale se pot reprezenta în *format virgulă fixă sau virgulă mobilă.*

b1) Reprezentarea în format virgulă fixă este asemănătoare reprezentării numerelor întregi. Se impun *n1* biţi pentru partea întreagă, respectiv *n2* biţi pentru cea fracţionară, rezultând o poziţie fix[pentru separatorul zecimal (punct sau virgulă) potrivit figurii 1.2.

s	M1 – mărime partea întreagă	M2 – mărime partea fracţionară
1	*n1* biţi	*n2* biţi

Fig. 1.2. Reprezentarea *format cu virgulă fixă* pentru numere reale.

b2) Reprezentarea în format virgulă fixă mobilă utilizează reprezentarea ştiinţifică

$$r = m \cdot 10^{E}, \tag{1.20}$$

căreia îi sunt asociate două componente:

• *E - exponentul* - indică ordinul de mărime al numărului;
• *m - mantisa* - arată mărimea exactă a numărului într-un anumit domeniu.

Considerăm că pentru reprezentarea unui număr în virgulă mobilă se utilizează *n* biţi din care *e* pentru exponent (care determină intervalul de valori) şi *m* biţi pentru mantisă (care determină precizia reprezentării). După cum se ştie cu *n* biţi se pot reprezenta 2^n numere reale.

Diferenţa faţă de formatul cu virgulă fixă constă în modul în care sunt interpretate aceste valori. Între numere reale din matematică şi numerele reale reprezentate în formatul cu virgulă mobilă există diferenţe legate de domeniul finit şi mulţimea discretă de valori. Domeniul finit de valori este determinat de capacitatea fizică limitată a memoriei calculatorului.

O reprezentare a valorilor limite pe axa reală permite evidenţierea situaţiilor in care se produce depăşire. Determinarea valorilor V_{min} şi V_{max} permite identificarea pe axa reală, figura 2.3, a domeniilor de valori reprezentabile.

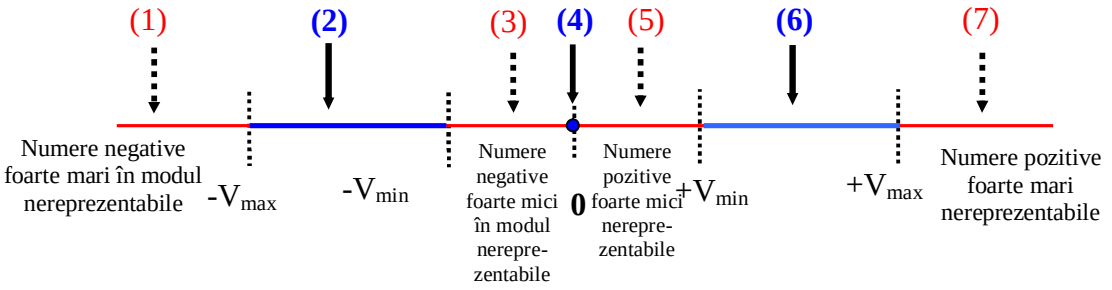

Fig. 1.3. Domeniile valorilor reale reprezentabile.

Zonele marcate cu cifre in figura 1.3 reprezintă următoarele categorii de numere:

1. numere negative foarte mari în modul nereprezentabile ;
2. numere negative reprezentabile
3. numere negative foarte mici în modul, nereprezentabile;
4. *numărul zero*;
5. numere pozitive foarte mici în valoare absolută nereprezentabile;
6. numere pozitive reprezentabile *;*
7. numere pozitive foarte mari in valoare absolută, nereprezentabile.

Domeniul de reprezentare va fi format din zonele (*2 – numere negative*) şi (*6 - numere pozitive*) la care se adaugă zona *4* reprezentată de numărul zero. Dacă în urma unei operaţii aritmetice rezultă o valoare în zonele (*1*) sau (*7*) apare *depăşirea flotantă* (inferioară dacă *rezultat* < $-V_{min}$ sau superioară dacă *rezultat* > V_{max}). Dacă rezultatul se situează în zonele (*3*) sau (*5*) se semnalizează imposibilitatea reprezentării pentru ca numerele sunt prea mici (negative – zona *3*, pozitive – zona *5*).

Caracterul discret al reprezentării în virgulă mobilă este dat de faptul că mulţimile din zonele (*2*) şi (*6*) ale schemei din figura 1.3 sunt mulţimi *finite*. Având în vedere că mulţimea numerelor reale este o mulţime *continuă* (între orice două numere reale se găseşte o infinitate de asemenea numere) rezultă că nu există o corespondenţă biunivocă între această mulţime şi mulţimea numerelor reprezentabile în virgulă mobilă.

Numărul de cifre al mantisei determina *precizia de reprezentare* în domeniile *2* şi *6* in timp ce numărul de cifre al exponentului afectează *mărimea aceloraşi* domenii.

Mantisa se reprezintă uzual in forma normalizată (*prima cifră din stânga diferită de zero)* si intr-una din bazele *2, 4, 8, 16***.**

Pentru a nu folosi exponenţi negativi se introduce noţiunea de *caracteristică***.** Aceasta este egală cu exponentul cu semn deplasat, astfel încât să ia valori intr-o mulţime de numere pozitive. De exemplu pentru un *exponent* in domeniul [-64, +63]∩Z, o deplasare cu +64 va conduce la o *caracteristică* în domeniul [0, +127]∩Z.

Reprezentarea standard în VM. Î anul 1985 s-a adoptat standardul *IEEE 754* referitor la reprezentarea numerelor în virgulă mobilă, standard acceptat de majoritatea firmelor producătoare de microprocesoare.

Standardul defineşte trei formate: *simpla precizie, dubla precizie, precizie extinsă,* tabelul 1.2 fiind prezentate caracteristicile primelor două formate.

Tabelul 1.2

Format	Biti	Semn	Exp.	Mantisa	Exces
Simplă precizie	32	1	8	23	127
Dublă precizie	64	1	11	52	1023

1.4. Operaţii aritmetice

În cele ce urmează se vor prezenta câteva elemente ce privesc realizarea operaţiilor aritmetice în cod binar.

Efectuarea oricărei astfel de operaţii se reduce la adunarea şi / sau scăderea numerelor binare conform regulilor următoare:

0+0 = 0 0-0 = 0
0+1 = 1 1-0 = 1
1+0 = 1 10-1 = 1+b
1+1 = 10=0+c 1-1 = 0

unde c (*carry*) este transportul la rangul superior, iar b (*borrow*) este împrumutul de la rangul superior.

Adunarea şi scăderea în reprezentarea *semn - mărime*. Cele două operaţii vor fi tratate unitar conform relaţiei:

$$op = x_S \oplus y_S \oplus s_{OP} \qquad (1.21)$$

unde: *op* reprezintă operaţia efectivă ce se va efectua între cei doi operanzi;
x_S, y_S - semnele celor doi operanzi (*1* pt. -, *0* pt. +);
s_{OP} - operaţia ce se doreşte a fi efectuată (*1* pt. -, *0* pt. +).

În ceea ce priveşte operaţia "⊕" (*sau exclusiv*) aceasta este definită astfel:
$0 \oplus 0 = 0$;
$0 \oplus 1 = 1$;
$1 \oplus 0 = 1$;
$1 \oplus 1 = 0$.

Pe baza celor prezentate, valoarea operatorului *op* se determină cu ajutorul tabelului 1.3.

Tabelul 1.3

x_s	y_s	S_{op}	Op	x_s	y_s	S_{op}	Op
0	0	0	0	+	+	+	+
0	0	1	1	+	+	-	-
0	1	0	1	+	-	+	-
0	1	1	0	+	-	-	+
1	0	0	1	-	+	+	-
1	0	1	0	-	+	-	+
1	1	0	0	-	-	+	+
1	1	1	1	-	-	-	-

Dacă *op=0*, se adună modulele celor doi operanzi semnul rezultatului fiind dat de semnul primului operand. Rezultatul este corect dacă nu se depăşeşte valoarea maximă pentru numărul respectiv de biţi.

Dacă *op=1*, se va efectua scăderea modulelor celor doi operanzi, semnul rezultatului fiind dat de semnul numărului mai mare în modul.

Adunarea şi scăderea în *C1*. Aceste operaţii se reduc la operaţia de adunare a numerelor reprezentate în C1. Cei doi operanzi se adună bit cu bit, inclusiv biţii de semn. Eventualul transport care rezultă la *rangul de semn* se va aduna la rangul cel mai puţin semnificativ. Dacă bitul de semn al rezultatului are valoarea 1 atunci rezultatul este în C1 (altfel este *semn – mărime*).

Adunarea şi scăderea în C2. Operaţiile se reduc la adunarea numerelor reprezentate în C2. Cei doi operanzi se adună bit cu bit inclusiv biţii de semn iar eventualul transport care rezultă la bitul de semn se neglijează. Dacă bitul de semn al rezultatului are valoarea 1 atunci rezultatul este în C2 (altfel este *semn – mărime*).

Înmulţirea numerelor binare. Majoritatea metodelor de înmulţire a numerelor binare se bazează pe procedeul adunării repetate. Vom exemplifica pentru produsul a două numere *x* şi *y* exprimate în *semn – mărime,* pentru care se parcurg următoarele etape:

a) se determină semnul produsului $s_P = s_x + s_Y$ unde s_P este semnul produsului iar s_x şi s_Y semnele celor doi factori $s_P, s_x, s_Y \in \{0, 1\}$ conform regulii:

$0 + 0 = 0$; $(+) \bullet (+) = (+)$
$0 + 1 = 1$; $(+) \bullet (-) = (-)$
$1 + 0 = 1$; $(-) \bullet (+) = (-)$
$1 + 1 = 0$; $(-) \bullet (-) = (+)$

b) se calculează modulul produsului prin însumarea produselor parţiale, respectiv

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

unde (1.22)

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

O categorie specială de înmulţire o reprezintă înmulţirea cu puteri ale bazei *2* respectiv cu 2^k, care presupune deplasări după cum urmează:

k>0 $\rightarrow$ deplasare *stânga* cu ***k*** poziţii (se adaugă zerouri în poziţiile nesemnificative din dreapta rămase libere);

k<0 $\rightarrow$ deplasare *dreapta* cu ***k*** poziţii (se adaugă zerouri în poziţiile semnificative rămase libere).

Împărţirea numerelor binare. Împărţirea numerelor binare are la bază regulile *0 : 1 = 0, 1 : 1 = 1* (împărţirea cu zero nu are sens). După cum s-a văzut, operaţia de înmulţire poate fi redusă la o serie de adunări. În mod similar operaţia de împărţire poate fi redusă la o serie de scăderi.

Adunarea şi scăderea numerelor reprezentate în format virgulă mobilă. Aceste operaţii se execută în mai multe etape şi anume:

a) dacă cei doi exponenţi (caracteristici) nu coincid se aduc numerele la acelaşi cel mai mare - cel mai mare (în acest scop numărul cu exponentul mai mic va fi deplasat cu $D = E_{max} - E_{min}$ poziţii la dreapta);

b) se adună mantisele în codul în care sunt reprezentate;

c) se normalizează mantisa obţinută în urma adunării.

Fie două numere în *VM* baza 16

$A = (\pm 0.M_1)_{16} \cdot 16^{E1}$ şi $B = (\pm 0.M_2)_{16} \cdot 16^{E2}$ unde M_1 şi M_2 sunt mantisele iar E_1 şi E_2 (unde $E_1 > E_2$) sunt caracteristicile celor două numere.

Suma respectiv diferenţa celor două numere se calculează astfel:

$$A \pm B = ((\pm 0.M_1)_{16} \pm (\pm 0.M_2)_{16} \cdot 16^{D}) \cdot 16^{E1} \qquad (1.23)$$

Înmulţirea şi împărţirea numerelor reprezentate în format cu virgulă mobilă. Această operaţie presupune adunarea (scăderea) exponenţilor (caracteristicilor) şi înmulţirea (împărţirea mantiselor).

Dacă cele două numere sunt:

$A = (\pm 0.M_1)_{16} \cdot 16^{E1}$ şi $B = (\pm 0.M_2)_{16} \cdot 16^{E2}$ atunci produsul şi câtul vor fi :

$$A \cdot B = ((\pm 0.M_1)_{16} \cdot (\pm 0.M_2)_{16}) \cdot 16^{E1+E2}$$

$$A : B = ((\pm 0.M_1)_{16} : (\pm 0.M_2)_{16}) \cdot 16^{E1-E2}$$

Produsul mantiselor va determina un număr de lungime dublă faţă de cele două mantise din care se vor reţine numai jumătate din cifre (cele mai semnificative) după care se face rotunjirea.

1.5. Coduri numerice şi alfanumerice

Pentru stocarea, prelucrarea şi transmiterea informaţiei se utilizează diverse codificări care elimină erorile de reprezentare permiţând detecţia şi corecţia erorilor.

1.5.1. Coduri numerice (binar – zecimale)

Cifrele zecimale pot fi reprezentate atât în binar cât şi în alte codificări care prezintă diverse avantaje în utilizare cum ar fi: **BCD**, codurile ponderate **8421** şi **2421** şi neponderate **Gray** şi **Exces 3.**

- **Codul BCD.** Fiecare cifră zecimală a unui număr este reprezentată prin codul său binar codificarea numindu-se *zecimal codificat binar (BCD – Binary Coted Decimal).* Această codificare se deosebeşte evident de codificarea binară, următorul exemplu fiind relevant.

*Operaţii aritmetice în cod BCD***.** Din modul în care se efectuează codificarea în BCD rezultă că sunt valide numai combinaţiile cuprinse între *0000* şi *1001,* celelalte (între *1010* şi *1111)* fiind invalide. Se pune problema efectuării de calcule aritmetice cu numere reprezentate în BCD, dar utilizând o UAL care lucrează binar. De exemplu 36+94 ş 130_{10} ş 1100 1010_{BCD} unde apar combinaţiile interzise 1100 şi 1010. Eliminarea acestora şi revenirea în cod BCD se face prin procedeul de *ajustare zecimală* care constă în:

a. dacă în urma adunării rezultă o cifră hexa $S \geq A$ atunci se adună 6 (respectiv *0110* şi se obţine cifra BCD;

b. dacă în urma adunării rezultă cifra $S \leq 3$, dar cu transport spre rangul superior, atunci se adună 6 (respectiv *0110* şi se obţine cifra BCD.

Această corecţie va permite efectuarea adunării în bază 10 *şi nu* în bază 16 cum s-ar efectua dacă s-ar lua în consideraţie toate combinaţiile posibile de câte 4 biţi.

În ceea ce priveşte celelalte operaţii acestea se reduc la adunare sau utilizează în mod repetat adunarea.

O menţiune pentru înmulţirea (împărţirea) cu 10^k ,echivalente cu deplasarea numărului cu *k* tetrade spre dreapta sau spre stânga faţă de poziţia punctului zecimal, după cum *k* este un întreg pozitiv, respectiv negativ.

- **Coduri ponderate şi neponderate.** *Un cod ponderat* asociază fiecărei cifre zecimale o tetradă binară, iar ponderea fiecărui bit din tetradă este egală cu valoarea cifrei din denumirea codului. Valoarea cifrei zecimale codificate se obţine prin însumarea biţilor cuvântului de cod, ponderaţi cu valoarea corespunzătoare din denumirea codului.

La *codurile neponderate* nu este o legătură directă între poziţia şi ponderea unui anumit bit..

In tabelul 1.4 se prezintă câte două coduri din fiecare categorie menţionată.

Tabelul 1.4

Cifră zecimală	Coduri ponderate		Coduri neponderate	
	8421	2421	Gray	Exces 3
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0010	0011	0101
3	0011	0011	0010	0110
4	0100	0100	0110	0111
5	0101	1011	0111	1000
6	0110	1100	0101	1001
7	0111	1101	0100	1010
8	1000	1110	1100	1011
9	1001	1111	1101	1100

Referitor la codurile prezentate în tabelul 1.4 se pot formula observaţiile de mai jos.

a) La codul *2421* codurile primelor 4 cifre sunt identice cu ale codului 8421. Codurile cifrelor a căror sumă este 9 sunt complementare exemplu 2 cu 7, 3 cu 6, 4 cu 5 etc).

b) Codul *Gray* are proprietatea de adiacenţă în sensul că trecerea de la o cifră la următoarea se face prin modificarea unui singur bit.

c) Un cuvânt al codului *EXCES 3* se obţine din cuvântul corespunzător din 8421 la care se adaugă 0011 (adică 3 în binar). Avantajul acestui cod este că se poate face distincţie între absenţa informaţie şi codul cifrei zero.

Cel mai răspândit cod este 8421 care stă la baza codificării *zecimal – binare (BCD).*

1.5.2. Coduri alfanumerice

În afara numerelor întregi sau reale informaţia prelucrată de calculator mai poate conţine text format din caractere, adrese de memorie, informaţie de stare etc.

Textul reprezintă una din formele cele mai utilizate pentru manevrarea şi memorarea informaţiei. În prezent calculatorul este utilizat într-o mare varietate de aplicaţii care nu necesită neapărat calcule matematice cum ar fi: editare, redactare, grafică etc. Tot cu ajutorul caracterelor se introduc în memoria calculatorului programele sursă şi se obţin rezultatele.

În general, în mulţimea caracterelor alfanumerice intră cifre, litere şi caractere speciale. Caracterele pot fi tipăribile sau nu, în ultima categorie intrând:

- caractere de control pentru comunicaţia între dispozitive;
- caractere de control pentru deplasarea cursor;
- caractere cu semnificaţie nedefinită.

În prezent marea majoritate a platformelor de calcul utilizează codul IBM **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange). Codul ASCII se prezintă în variantele restrâns şi extins. Codul ASCII restrâns asociază fiecărui caracter câte o formaţie de 7 biţi iar cel extins câte una de 8 biţi. Rezultă că cele două coduri permit codificarea a 128 respectiv a 256 de caractere. Exemple de coduri ASCII: A: 41h, 2:32h, *:2Ah, ?:3Fh, etc.

1.5.3. Coduri pentru detectarea şi corectarea erorilor

Relativ la erorile care pot apare la transmisia sunt de menţionat două categorii de probleme

a. detectarea erorilor;
b. corectarea erorilor.

La rândul său detectarea erorilor comportă două tipuri de operaţii:

a1. detectarea stării de eroare;
a2. diagnoza erorii.

Prin detectarea *stării de eroare* se specifică existenţa unei erori fără a se preciza exact eroarea. *Diagnoza* presupune determinarea erorii, respectiv a biţilor care au fost transmişi eronat.

Importantă în detectarea erorilor de transmisie este *distanţa Hamming (DH)* definită ca numărul de poziţii binare prin care diferă două cuvinte de cod. De exemplu cuvintele binare $x = 01100$ şi $y = 00101$ diferă prin două poziţii binare, deci distanţa Hamming este $d = 2$. Fiind vorba de sesizarea lipsei coincidenţei, d se poate determina cu ajutorul funcţiei *SAU – EXCLUSIV*, respectiv $d = x \oplus y$. Semnificaţia *DH* este că sunt necesare d erori de un singur bit

pentru a converti un cuvânt în altul. De asemenea pentru a elimina *d* erori este nevoie de un cod cu *DH = d + 1*.

Cea mai simplă soluţie de semnalare a stării de eroare este adăugarea bitului de *paritate (bP).* Dacă de exemplu la codul 8421 pe 4 biţi se adaugă un bit de paritate se obţine un cod detector de erori pe 5 biţi. *DH* a acestui cod este 2 deoarece la transmisia greşită a unui bit este afectat şi bitul de paritate. Se poate utiliza convenţia de paritate *pară (PP) sau impară (PI)*. La *PI* numărul total de biţi **1** din codul 8421 împreună cu bitul de paritate este impar, iar la *PP* acest număr este par.

În practică este larg utilizată şi metoda *codului polinomial* cunoscut şi sub denumirea de *cod cu redundanţă ciclică* sau cod ***CRC (Cyclic Redundancy Code)***.

La transmiterea datelor între cele mai utilizate polinoame generatoare sunt următoarele :

$CRC\text{-}12 = x^{12} + x^{11} + x^3 + x^2 + x + 1$

$CRC\text{-}16 = x^{16} + x^{15} + x^2 + 1$

$CRC\text{-}CCITT = x^{16} + x^{12} + x^5 + 1$

Corectarea erorii presupune înlocuirea biţilor detectaţi ca fiind transmişi eronat. Un cod este **cod corector de erori** (CCE) atunci când cuvântul de cod corect poate fi întotdeauna dedus din cuvântul eronat. Între CCE un loc aparte revine *codului Hamming*, care în cazul unei corecţii singulare are distanţa **3.** Numărul minim de biţi de control pentru a asigura această distanţă se determină cu relaţia

$$2^k \geq m + k + 1, \qquad (1.24)$$

unde *m* este numărul de biţi de informaţie, iar *k* - numărul de biti de control.