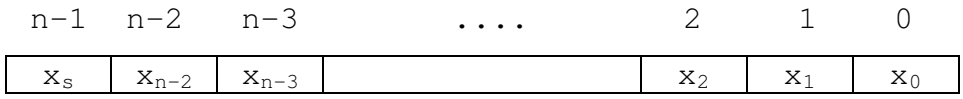
2.2 Reprezentarea numerelor in virgula fixa

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:



pozitia punctului zecimal pentru un numar subunitar

pozitia punctului zecimal pentru un numar intreg

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:

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

In practica se utilizeaza trei coduri de virgula fixa: codul direct, codul invers si codul complementar.

Reprezentarea in cod direct

Codul direct permite reprezentarea explicita a numerelor prin semn si modul.

$$[x]_d = \begin{cases} 0x_{n-2}x_{n-3}...x_1x_0 & daca\ x \geq 0 \\ 1x_{n-2}x_{n-3}...x_1x_0 & daca\ x < 0 \end{cases}$$

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

$[x]_d = 010101$

$[y]_d$ = 110100

Reprezentarea in cod invers (sau in complement fata de 1)

$$[x]_i = \begin{cases} 0x_{n-2}x_{n-3}...x_1x_0 & daca\ x > 0 \\ 0000...000 & \\ \quad sau & daca\ x = 0 \\ 1111...111 & \\ 1\bar{x}_{n-2}\bar{x}_{n-3}...\bar{x}_1\bar{x}_0 & daca\ x < 0 \end{cases}$$

unde $\bar{x}_k = 1 - x_k$. Se poate stabili o relatie importanta pentru numerele strict negative reprezentate in cod invers. Astfel, daca $x<0$, atunci:

$$|x| + [x]_i = 0x_{n-2}x_{n-3}...x_1x_0 + 1\bar{x}_{n-2}\bar{x}_{n-3}...\bar{x}_1\bar{x}_0 = 1111...111 = 2^n - 1$$

Exemple. Sa se reprezinte in cod invers numerele $x = 21$ si $y = -27$ pe 6 biti.

$[x]_d$ = 010101

Reprezentarea numerelor pozitive in cod invers este identica cu reprezentarea in cod direct. Numerele negative se reprezinta insa diferit. Pentru y se scrie mai intai reprezentarea in cod direct, apoi se inverseaza (se complementeaza fata de 1) fiecare bit al modulului.

$[y]_d$ = 110100
$[y]_i$ = 101011

Reprezentarea in cod complementar (in complement fata de 2)

$$[x]_c = \begin{cases} 0x_{n-2}x_{n-3}...x_1x_0 & daca\ x \geq 0 \\ 1\tilde{x}_{n-2}\tilde{x}_{n-3}...\tilde{x}_1\tilde{x}_0 & daca\ x < 0 \end{cases}$$

unde, prin definitie, $1\tilde{x}_{n-2}\tilde{x}_{n-3}...\tilde{x}_1\tilde{x}_0 = 2^n - |x|$. Se poate stabili o relatie importanta pentru reprezentarea numerelor negative in cod complementar. Daca $x<0$. atunci:

$$|x|_c = 1\tilde{x}_{n-2}\tilde{x}_{n-3}...\tilde{x}_1\tilde{x}_0 = 2^n - |x| = |x| + [x]_i + 1 - |x| = [x]_i + 1$$

Acest rezultat reprezinta si o metoda pentru determinarea reprezentarii numerelor negative in cod complementar.

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

$[x]_d = 010101$

Din nou, reprezentarea numerelor pozitive in cod complementar este identica cu reprezentarea in cod direct. Pentru a obtine reprezentarea unui numar negativ in cod complementar, se scrie mai intai numarul in cod direct, apoi se inverseaza bitii modulului pentru a obtine codul invers si in final se aduna o unitate, in rangul cel mai putin semnificativ (c.m.p.s.), conform relatiei stabilite mai sus.

$$\begin{array}{l} [y]_d = 110100 \\ [y]_i = 101011+ \\ \qquad\qquad\ 1 \\ \hline [y]_c = 101100 \end{array}$$

Comparand reprezentarea in cod complementar a unui numar negativ cu reprezentarea in cod direct, rezulta urmatoarea regula practica pentru scrierea codului complementar (numai pentru numerele negative !): *se parcurge reprezentarea numarului in cod direct de la dreapta spre stanga, se lasa toate zerourile neschimbate pana la prima unitate, care de asemenea ramane neschimbata, iar apoi se inverseaza toti ceilalti biti ai modulului.*

2.3 Adunarea si scaderea in virgula fixa

Adunarea si scaderea in cod direct

In cazul operatiilor in cod direct 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).

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 $z_s = \overline{y}_s$.

Adunarea si scaderea in cod invers

Adunarea si scaderea in cod invers se reduc la operatia de adunare (in cazul operatiei de scadere, se aduna la descazut opusul scazatorului). Operanzii se prelucreaza impreuna cu semnele lor. Astfel, se aduna cei doi operanzi bit cu bit, inclusiv bitii de semn, iar eventualul transport care rezulta de la rangul de semn se aduna in rangul c.m.p.s (reprezinta o corectie) si se obtine rezultatul corect in cod invers. Se disting urmatoarele cazuri:

1) $x>0$, $y>0$, $x+y<2^{n-1}$ (pentru a nu avea depasire). Operatia de adunare este asemanatoare cu cea din cod direct, dar in plus se aduna si bitii de semn (0+0).

$[x]_i+[y]_i = |x|+|y| =$ (pentru ca operanzii au acelasi semn)
$= |x+y| = [x+y]_i$

2) $x>0$, $y<0$, $|x|>|y|$. La adunare apare un transport de la rangul de semn, care se aduna in rangul c.m.p.s. (corectie).

$[x]_i+[y]_i = |x|+2^n -1- |y| =$ (2^n si -1 se reduc, corectie)
$= |x|-|y| =$ (semne diferite si $|x|>|y|$)
$= |x+y| = [x+y]_i$

3) $x>0$, $y<0$, $|x|<|y|$. La adunare nu apare transport de la rangul de semn.

$$[x]_i+[y]_i = |x|+2^n -1- |y| = 2^n -1-(|y|-|x|) = \text{(semne diferite si } |x|<|y|)$$
$$= 2^n -1-|y+x| = \ (x+y \text{ este negativ})$$
$$= \ [x+y]_i$$

4) $x<0$, $y<0$, $|x+y|<2^{n-1}$. La adunare apare un transport de la rangul de semn, care se aduna in rangul c.m.p.s. (corectie).

$$[x]_i+[y]_i = 2^n -1- |x|+2^n -1- |y| = (2^n \text{ si } -1 \text{ se reduc, corectie})$$
$$=2^n -1- |x| - |y| = (x \text{ si } y \text{ au acelasi semn})$$
$$= 2^n -1- |x+y| = \ (x + y \text{ este negativ})$$
$$= [x+y]_i$$

Adunarea si scaderea in cod complementar

Adunarea si scaderea in cod complementar se reduc la operatia de adunare (in cazul operatiei de scadere, se aduna la descazut opusul scazatorului). Operanzii se prelucreaza impreuna cu semnele lor. Se aduna cei doi operanzi bit cu bit, inclusiv bitii de semn, iar eventualul transport care rezulta de la rangul de semn se neglijeaza (reprezinta o corectie) si se obtine rezultatul corect in cod complementar. Se observa urmatoarele cazuri:

1) $x>0$, $y>0$, $x+y<2^{n-1}$ (pentru a nu avea depasire). Operatia de adunare este asemanatoare cu cea din cod direct, dar in plus se aduna si bitii de semn (0+0).

$$[x]_c+[y]_c = |x|+|y| = \text{(pentru ca operanzii au acelasi semn)}$$
$$= |x+y| = [x+y]_c$$

2) $x>0$, $y<0$, $|x|>|y|$. La adunare apare un transport de la rangul de semn, care se neglijeaza (corectie).

$$[x]_c+[y]_c = |x|+2^n - |y| = (2^n \text{ se neglijeaza, corectie})$$
$$= |x|-|y| = \text{(semne diferite si } |x|>|y|)$$
$$= |x+y| = [x+y]_c$$

3) $x>0$, $y<0$, $|x|<|y|$. La adunare nu apare transport de la rangul de semn.

$$[x]_c+[y]_c = |x|+2^n - |y| = 2^n -(|y|-|x|) = \text{(semne diferite si } |x|<|y|)$$
$$= 2^n -|y+x| = \ (x+y \text{ este negativ})$$
$$= \ [x+y]_c$$

4) $x<0$, $y<0$, $|x+y|<2^{n-1}$. La adunare apare un transport de la rangul de semn, care se neglijeaza (corectie).

$$[x]_c+[y]_c = 2^n - |x|+2^n - |y| = (2^n \text{ se neglijeaza, corectie})$$
$$=2^n - |x| - |y| = (x \text{ si } y \text{ au acelasi semn})$$
$$= 2^n - |x+y| = \ (x + y \text{ este negativ})$$
$$= [x+y]_c$$

2.4 Inmultirea in virgula fixa

Inmultirea in cod direct

Pentru inmultirea a doua numere reprezentate in virgula fixa cod direct ($x\bullet\ y$ = 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.

Inmultirea in cod invers

Se va studia mai intai modul in care se face deplasarea unui numar negativ in cod invers (un numar pozitiv in cod invers avand aceeasi reprezentare ca in cod direct, se deplaseaza la fel spre stanga / dreapta, prin introducerea de zerouri prin dreapta / stanga numarului).

Fie $x<0$ (se va considera un numar subunitar, pe 6 biti, ca in exemplele din acest paragraf):

$[x]_d = 1.x_{-1}\ x_{-2}\ x_{-3}\ x_{-4}\ x_{-5}$

iar acelasi numar pe lungime dubla, cum sunt produsele partiale de la inmultire:

$[x]_d = 1.x_{-1}\ x_{-2}\ x_{-3}\ x_{-4}\ x_{-5}00000$

trecut apoi in cod invers:

$$[x]_i = 1.\overline{x}_{-1}\overline{x}_{-2}\overline{x}_{-3}\overline{x}_{-4}\overline{x}_{-5}11111$$

in continuare, *x* deplasat stanga o pozitie (echivalent cu o inmultire cu 2), in cod direct:

$$[x \cdot 2]_d = 1.x_{-2}x_{-3}x_{-4}x_{-5}000000$$

trecut acest numar in cod invers:

$$[x \cdot 2]_i = 1.\overline{x}_{-2}\overline{x}_{-3}\overline{x}_{-4}\overline{x}_{-5}111111$$

Se considera *x* impartit la 2 (echivalent cu o deplasare spre dreapta) in cod direct:

$$\left[x \cdot 2^{-1}\right]_d = 1.0x_{-1}x_{-2}x_{-3}x_{-4}x_{-5}0000$$

scris apoi in cod invers:

$$\left[x \cdot 2^{-1}\right]_i = 1.1\overline{x}_{-1}\overline{x}_{-2}\overline{x}_{-3}\overline{x}_{-4}\overline{x}_{-5}1111$$

Se compara $[x]_i$ cu $[x \bullet 2]_i$, respectiv cu $[x \bullet 2^{-1}]_i$. Rezulta urmatoarea regula: numerele negative reprezentate in cod invers se deplaseaza spre stanga / dreapta cu introducerea de unitati prin dreapta / stanga numarului.

La inmultirea numerelor in cod invers este necesar ca inmultitorul sa fie pozitiv, in caz contrar se schimba semnele celor doi operanzi. Operanzii sunt prelucrati impreuna cu semnele lor, caci este necesar sa se adune produsele partiale, care sunt scrise tot in cod invers, iar la operatia de adunare se aduna si bitii de semn.

Inmultirea in cod complementar

La fel ca la inmultirea in cod invers, se va studia mai intai modul in care se face deplasarea unui numar negativ in cod complementar (un numar pozitiv in cod complementar avand aceeasi reprezentare ca in cod direct, se deplaseaza la fel spre stanga / dreapta, prin introducerea de zerouri prin dreapta / stanga numarului).

Fie $x<0$ (se va considera un numar subunitar, pe 6 biti, ca in exemplele din acest paragraf):

$$[x]_d = 1.x_{-1}\ x_{-2}\ x_{-3}\ x_{-4}\ x_{-5}$$

iar acelasi numar pe lungime dubla, cum sunt produsele partiale de la inmultire:

$$[x]_d = 1.x_{-1}\ x_{-2}\ x_{-3}\ x_{-4}\ x_{-5}00000$$

trecut apoi in cod complementar:

$$[x]_c = 1.\tilde{x}_{-1}\tilde{x}_{-2}\tilde{x}_{-3}\tilde{x}_{-4}\tilde{x}_{-5}00000$$

in continuare, x deplasat stanga o pozitie (echivalent cu o inmultire cu 2), in cod direct:

$$[x \cdot 2]_d = 1.x_{-2}x_{-3}x_{-4}x_{-5}000000$$

trecut acest numar in cod complementar:

$$[x \cdot 2]_c = 1.\tilde{x}_{-2}\tilde{x}_{-3}\tilde{x}_{-4}\tilde{x}_{-5}000000$$

Se considera x impartit la 2 (echivalent cu o deplasare spre dreapta) in cod direct:

$$\left[x \cdot 2^{-1}\right]_d = 1.0x_{-1}x_{-2}x_{-3}x_{-4}x_{-5}0000$$

scris apoi in cod complementar:

$$\left[x \cdot 2^{-1}\right]_c = 1.1\tilde{x}_{-1}\tilde{x}_{-2}\tilde{x}_{-3}\tilde{x}_{-4}\tilde{x}_{-5}0000$$

Se compara $[x]_c$ cu $[x \bullet 2]_c$, respectiv cu $[x \bullet 2^{-1}]_c$. Rezulta urmatoarea regula: numerele negative reprezentate in cod complementar se deplaseaza spre stanga / dreapta cu introducerea de zerouri / unitati prin dreapta / stanga numarului.

La inmultirea numerelor in cod complementar este de asemenea necesar ca inmultitorul sa fie pozitiv, in caz contrar se schimba semnele celor doi operanzi. Operanzii sunt prelucrati impreuna cu semnele lor, caci

este necesar sa se adune produsele partiale, care sunt scrise tot in cod complementar, iar la operatia de adunare se aduna si bitii de semn.

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 (pentru cod direct), metoda refacerii restului partial (de asemenea pentru cod direct), metoda fara refacerea restului partial (pentru cod complementar). In continuare se va prezenta pe scurt metoda comparatiei.

Metoda comparatiei

Aceasta metoda se aplica pentru numere reprezentate in virgula fixa cod direct, 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
|     qs = xs ⊕ ys
|     rs = xs
|     |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):

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

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}, inlocuidu-se succesiv fiecare $r^{(k)}$ din relatia precedenta in functie de $r^{(k-1)}$. Se obtine:

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

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

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

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