REPREZENTAREA NUMERELOR SI ARITMETICA LA PROCESOARELE DE SEMNAL

1. Generalităţi

Fără a avea informaţii legate de convenţia folosită la reprezentarea numerelor binare în calculator nu se poate atribui o valoare exactă unui număr binar, acesta reprezentând o înşiruire de biţi. De exemplu, ce valoare zecimală are 10111011B = X ? Este pozitiv sau negativ ? Aceasta depinde de reprezentarea folosită.

La alegerea unui Procesor Digital de Semnal (DSP) pentru o anumită aplicaţie una dintre caracteristicile importante la luarea deciziei o constituie modul de reprezentare (binar) al datelor de către procesor. Din acest punct de vedere DSP-urile de pe piaţă se pot clasifica după cum arată fig. 1.

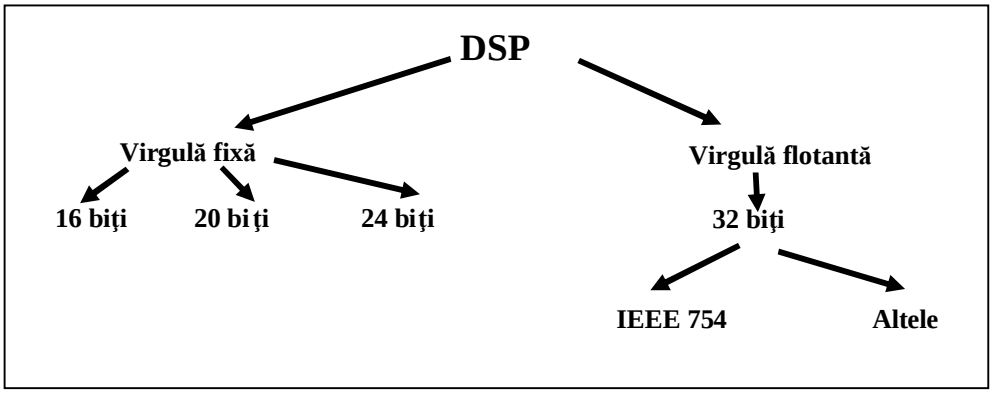


Fig. 1 Reprezentarea numerelor la DSP-urile uzuale

Aritmetica în virgulă fixă a fost folosită de către primele DSP-uri şi la ora actuală mai este încă predominantă la multe dintre DSP-urile uzuale.

DSP-urile în virgulă fixă pot reprezenta numerele fie ca:

- întregi (aritmetica întreagă) – fiind folosită de DSP la operaţii de control, calcul de adrese sau alte operaţii care nu implică semnale, sau
- fracţionar (aritmetica fracţionară) cu valori între -1 şi +1 utilă în calculele legate de semnale.

Algoritmii şi hard-ul folosit la implementarea aritmeticii fracţionare sunt virtual identici cu cei folosiţi la aritmetica întreagă. Diferenţa principală între cele două aritmetici apare la modul de folosire a rezultatelor operaţiilor de înmulţire. Majoritatea DSP-urilor în virgulă fixă acceptă cele două aritmetici.

Intrucât sistemele numerice folosesc pentru reprezentarea numerelor un cuvânt de o anumită lungime operaţiile aritmetice se vor executa cu o anumită precizie (limitată). Numerele vor fi astfel reprezentate pe un cerc (inel) în loc de axa reală infinită, situaţie în care poate să apară depăşirea aritmetică.

În cazul **DSP-urilor în virgulă mobilă**, valorile sunt reprezentate de o mantisă şi de un exponent conform relaţiei $mantisa*2^{exponent}$. Mantisa este în general o fracţie în intervalul -1.0 şi +1.0, în timp ce exponentul este un întreg care reprezintă numărul de poziţii cu care trebuie deplasat stânga punctul binar (termen definit analog cu punctul zecimal) pentru a obţine valoarea reprezentată.

Procesoarele în virgulă mobilă sunt mai uşor de programat decât cele în virgulă fixă, dar sunt şi mai scumpe. Acest lucru se datorează complexităţii sporite a circuitului care se traduce într-o suprafaţă mai mare a chip-ului. Uşurinta programării acestor procesoare este dată şi de faptul că programatorul nu mai trebuie să gestioneze cazurile de depăşire (overflow) ale acumulatorului ca şi în cazul procesoarelor în virgulă fixă (această gestionare înseamnă scalarea periodică a rezultatului în diverse faze ale prelucrării).

Majoritatea aplicaţiilor de cost redus utilizează procesoare în virgulă fixă. Programarea în acest caz nu are în general nici o dificultate, deoarece în faza de simulare numerică a algoritmului (care precede faza de dezvoltare) se pot detecta toate situaţiile în care este necesară corecţia rezultatului pentru a se evita saturarea. Pentru o mai bună înţelegere a acestor fenomene de overflow, vom prezenta două situaţii care se întalnesc în cazul procesoarelor DSP sau a celor ce suportă tehnologia MMX, şi anume: wraparound (modulo 2^n) şi saturarea.

Pentru exemplificare, se prezintă în cele ce urmează o aplicaţie pe imagini: având imaginea originală pe nivele de gri (Fig. 2a) se poate obţine efectul de wraparound (Fig. 2b) şi de saturare (Fig. 2c) folosind următoarele prelucrări:

- dacă rezultatul adunării între pixelişi o valoare produce depăşire, rezultatul este trunchiat, luându-se în considerare doar biţii cei mai puţin semnificativi (efectul wraparound), datorită limitării la n biţi;
- dacă rezultatul adunării produce o depăşire, apare situaţia de saturare şi astfel rezultatul este limitat la valoarea maximă a domeniului;

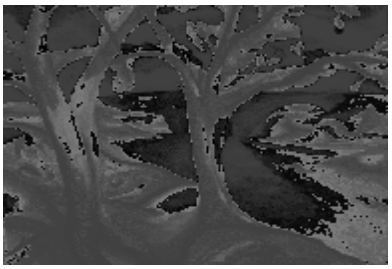




Fig2. a) Imaginea originală pe nivele de gri; b) Influenţa operaţiei modulo 2^n; c) Influenţa saturării.

Numerele mici reprezintă zone cu nivele de gri închise la culoare (negru), în timp ce numerele mari reprezintă arii de culoare deschisă (alb). Folosind 8 biţi pentru reprezentarea valorilor pixelilor din imagine, se va obţine gama numerelelor [0; 255], cu valoarea 0 pentru negru şi 255 pentru alb.

Pentru a ilumina imaginea originală (Fig. 2a) se poate aduna un număr pozitiv întreg (de exemplu $64_{10}=40_h$) la fiecare pixel din imagine.

În cazul efectului de wraparound (modulo 2^8), dacă există depăşire (adică valoarea unui pixel depăşeşte pragul maxim de 255), rezultatul final va fi trunchiat a.î. doar biţii cei mai puţin semnificativi vor fi reţinuţi. De exemplu, dacă adunăm 64_{10} la 250_{10} (aproape alb), vom obţine:

```
   250 zecimal      1111.1010 binar
 +  64 zecimal    + 0100.0000 binar
-------------    ------------------
 = 314 zecimal   = 10011.1010 binar  - se produce depasire (overflow)
 =  58 zecimal   =  0011.1010 binar  - se retin doar cei mai putin semnificativi 8 biti
```

Rezultatul este 58_{10}, producând o zonă închisă la culoare (aproape negru) în locul uneia mai deschise la culoare, cum era de aşteptat. S-a obţinut un efect invers celui dorit, deci o inversare de nuanţă, deoarece valorile de nuanţă deschisă au devenit zone de nuanţă închisă.

În cazul efectului de saturare, prin adăugarea unei valori fiecărui pixel din imaginea originală, zonele deschise la culoare vor deveni zone pur albe. Pragul de saturare în acest caz este valoarea maximă reprezentabilă pe 8 biţi, 255.

0x00	0x00	0x57	0x7F	0xAB	0xAB	oxC8	0xFF
+	+	+	+	+	+	+	+
0x40	0x40	0x40	0x40	0x40	0x40	0x40	0x40
=	=	=	=	=	=	=	=
0x40	0x40	0x97	0xBF	0xEB	0XEB	0x08	0x3F

Fig. 3 a) obţinerea valorilor pixelilor în situaţia modulo 2^8

0x00	0x00	0x57	0x7F	0xAB	0xAB	oxC8	0xFF
+	+	+	+	+	+	+	+
0x40	0x40	0x40	0x40	0x40	0x40	0x40	0x40
=	=	=	=	=	=	=	=
0x40	0x40	0x97	0xBF	0xEB	0XEB	0xFF	0xFF

b) obţinerea valorilor pixelilor în situaţia de saturare

2. Reprezentarea întreagă

Pe un cuvânt de n biţi se pot reprezenta 2^n numere plasate în gama numerelor egal distanţate cu pasul de cuantizare q=1.

Reprezentarea binară fără semn, pentru un număr reprezentat pe n biţi este: $B_2 = b_{n-1}\ b_{n-2}\ \ldots\ldots b_1\ b_0$ cu valoarea zecimală: $B_{10} = b_{n-1}*2^{n-1}+\ldots+b_1*2 + b_0$ aflat în gama $[0, 2^n-1]$. Se poate vedea în Fig. 4 "roata numerelor" pe care se pot verifica proprietăţile adunării pentru numerele reprezentate pe un număr finit de biţi, în acest caz n=4.

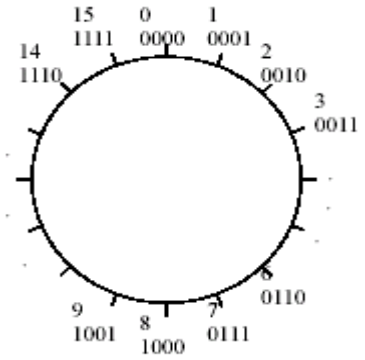


Fig. 4 "Roata numerelor" fără semn reprezentate pe 4 biţi

Exemplu. Folosind reprezentarea pe 4 biţi fără semn se incrementează cel mai mare număr reprezentabil astfel:

$1111_2 = 15_{10}$
$1_2 = 1_{10}$
_____
$1|0000_2 = 0_{10}$, numărul 10000_2 nu mai este pe 4 biţi , a apărut depăşire!

Bit de transport

Numerele binare fără semn pe n biţi reprezintă un sistem modulo 2^n!

Pentru a reprezenta **numerele binare negative** (deci cu semn) se utilizează reprezentarea în complement faţă de 2. Un număr în complement faţă de 2 pe n biţi este: $B_2 = b_{n-1}\ b_{n-2}\ \ldots\ldots b_1\ b_0$, cu valoarea zecimală:
$B_{10} = -b_{n-1}*2^{n-1}+\ldots+b_1*2 + b_0$ aflat în gama $[-2^{n-1}, 2^{n-1}-1]$.

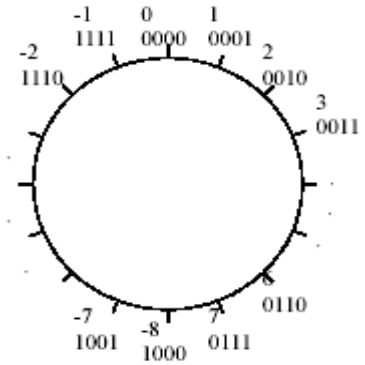


Fig.5. "Roata numerelor" cu semn reprezentate pe 4 biţi

Observaţie. La reprezentarea zecimală numerele 100_{10} şi 0100_{10} sunt identice, dar la reprezentarea în complement faţă de 2 pentru numerele 1010_2 şi 01010_2 nu putem presupune acelaşi lucru. De ce? La reprezentarea zecimală se foloseste pentru semn un caracter special (-), pe când la reprezentarea binară semnul este dat de bitul de semn. De aceea, valoarea zecimală a numărului 1010_2 depinde de lungimea pe care este reprezentat. Dacă este pe 5 biţi are valoarea 10_{10} , iar dacă este pe 4 biţi are valoarea -6_{10}.

Proprietăţi

- Inmulţirea a două numere binare întregi, în virgulă fixă, reprezentate pe n biţi dă un rezultat pe 2n biţi
- Adunarea a două numere reprezentate pe n biţi poate da un rezultat pe n+1 biţi

Multiplicarea a doi întregi în virgulă fixă duce cu mare probabilitate la depăşire.

La N adunări succesive pentru a nu avea depăşire sunt necesari $\log_2(N)$ biţi suplimentari.

Exemplu. Folosind reprezentarea în virgulă fixă pe 4 biţi să se facă înmulţirea : 3_{10} cu 2_{10} .
Observaţie. Rezultatul este pe 8 biţi şi a nu se uita extensia semnului la lungimea cuvântului rezultatului.

```
     1101   = -3 10
     0010   =  2 10
-------------------------------------
00000000
1111101
000000
00000
-------------------------------------
11111010   = -6 10
```

Dacă se revine la reprezentarea pe 4 biţi rezultatul va fi $1010_2 = -6_{10}$, deci este corect.

Exemplu. Să se facă înmulţirea în virgulă fixă pe 4 biţi a numerelor -3_{10} şi 6_{10} .

```
    1101  = -3 10
    0110  =  6 10
-------------------------------------
00000000
1111101
111101
00000
-------------------------------------
11101110  = -18 10
```

Dacă se revine la reprezentarea pe 4 biţi, rezultatul va fi $1110_2 = -2_{10}$, deci nu este corect.

Proprietăţi. Orice secvenţă de operaţii a cărui rezultat final se poate reprezenta corect în gama dată se poate calcula corect chiar dacă apar depăşiri la rezultatele intermediare.

Exemplu. Folosind reprezentarea pe 4 biţi cu semn (gama [-8,7)) să se calculeze expresia: 7+6-8 = 5

```
0111+     7+
0110      6
------------------------
1101 +   -3+ (depăşire)
1000     -8
-------------------------
0101      5  (corect)
```

3. Reprezentarea fracţionară

Pentru a se evita problema depăşirilor în cazul înmulţirii, reprezentarea fracţionară în virgulă fixă limitează sau normalizează numerele în gama [-1.0, 1.0), caz în care problema este rezolvată.

Excepţie. (-1)*(-1) = +1 ! care nu aparţine gamei [-1,+1).

La adunare şi scădere depăşirea este totuşi posibilă!

Observaţii.
1. Rezultatul multiplicării este trunchiat pierzând din precizie prin suprimarea biţilor mai puţin semnificativi.
2. Reprezentarea fracţionară pe n biţi se obţine prin deplasarea punctului cu n-1 poziţii spre stânga, adică reprezentarea conţine bit de semn, punct radix şi n-1 biţi fracţionari, deci: $B_{2fractionar} = b_0\, b_1 \ldots\ldots b_{n-2}\, b_{n-1}$, cu valoarea zecimală $B_{10} = -b_0 * 2^0 + b_1 * 2^{-1} + \ldots.. + b_{n-1} * 2^{-(n-1)}$.
3. Pasul de cuantizare este $q=2^{-(n-1)}$

Reprezentarea fracţionară este cunoscută şi sub numele de reprezentarea Qx, unde x este numărul de biţi fracţionari. Numărul total de biţi ai reprezentării fiind x+1. Reprezentarea fracţionară uzuală la DSP–urile în virgulă fixă (16 biţi) este Q15.

Localizarea punctului nu este neapărat precizată aceasta fiind un instrument de programare. Astfel folosind reprezentarea pe 16 biţi gama numerelor poate fi:

[0, 65535] - la reprezentarea întreagă fără semn
[-32768, +32767] - la reprezentarea întreagă cu semn
[-1, +0.999969] - la reprezentarea fracţionară

Toate aceste variante sunt utilizate la aplicaţiile cu DSP.

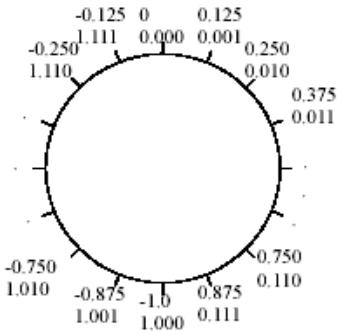


Fig. 6. "Roata numerelor" pentru reprezentarea fracţionară pe 4 biţi

Observaţie. La adunarea a două numere binare reprezentarea lor trebuie făcută în aceiaşi convenţie.

Exemple :

Să se reprezinte în format:

a) Q3, numerele –0.375 şi 0.75

La DSP în virgulă fixă punctul radix (zecimal) nu este disponibil, aceasta cade în sarcina programatorului. La operaţiile aritmetice cu date fracţionare sau cu semn se foloseşte aceeaşi unitate aritmetică şi logică (UAL). Reprezentarea fracţionară se obţine scalând valoarea cu 2^x, unde x reprezintă numărul de biţi fracţionari. Deci:

$-0.375_{10} = 1.101_2 * 2^3 = 1101_2 = -3_{10}$
$0.75_{10} = 0.110_2 * 2^3 = 0110_2 = 6_{10}$

b) Q15, numărul X_{10} = - 8.9969653E-003

Se calculează $X*2^{15}$ = -294.8 (~ -295), care în reprezentarea în complement faţă de 2 este X_{Q15} = 0FED9h .

Să se înmulţească folosind reprezentarea fracţionară pe 4 biţi:

c) -0.5_{10} cu 0.75_{10}

$1.100 = -0.5_{10}$
$0.110 = 0.75_{10}$

00000000
1111100
111100
00000

$11.101000_2 = -0.375_{10}$

↖ Bit de semn suplimentar

Observaţie. Atenţie la extensia de semn.
Rezultatul va avea 6 biţi fracţionari din 8.

Revenim la reprezentarea pe 4 biţi :

$1|1.101|000_2 = 1.101_2 = -0.375_{10}$ care reprezintă un rezultat corect.

d)-0.5_{10} cu 0.625_{10}

$1.100 * 0.101 = 11.101100_2 = -0.3125_{10}$

Revenim la reprezentarea pe 4 biţi:

$1|1.101|100_2 = 1.101_2 = -0.375_{10}$ care este incorect datorită trunchierii.

e) -0.25_{10} cu 0.5_{10} în format Q15.

```
0.100 0000 0000 0000 *
1.110 0000 0000 0000
 ------------------------------------------------------
11.11 1000  0000 0000    0000 0000  0000  0000 format Q30
```

Trecem la format Q15, eliminăm bitul de semn suplimentar şi rezultă:

$1.1110\ 0000\ 0000\ 0000_2 = -0.125_{10}$, corect.

Avantaj: Utilizând fracţii binare se obţine o viteză mai mare în calculele în buclă închisă.
Dezavantaj: Rezultatul poate să nu fie exact. După cum rezultă de mai sus în memorie se obţine valoarea (-4/16). Biţii cuprinşi între 2^{-4} şi 2^{-6} au fost trunchiaţi. Rezultatul corect este (-3/16) !
Este de menţionat că operaţia de înmulţire pe 4 biţi nu se face la capacitatea reală a C28x care operează pe 32 de biţi. În acest caz trunchierea va afecta biţii cuprinşi între 2^{-32} şi 2^{-64}. În cele mai multe cazuri se trunchiază numai zgomotul. Cu toate acestea, în unele aplicaţii cu reacţie (de exemplu filtrele IIR) erorile mici pot adăuga şi conduce la un anumit nivel de instabilitate. Este responsabilitatea proiectantului de a recunoaşte această potenţială sursă de eşec în folosirea fracţiilor binare.

4. Operaţii cu numere mai mari ca 1

a) Se scalează toţi coeficienţii din algoritm astfel încât să intre în gama [-1, 1). Ca efect rezultă o atenuare a semnalului la ieşire păstrând răspunsul în frecvenţă.

b) Se utilizează proprietăţile înmulţirii întregi cu 1: $A*B = (A-1)*B+B$

Exemplu. Să se înmulţească eşantionul $X_i = 0.625$ cu coeficientul 1.375. Se presupune reprezentarea pe 4 biţi.

```
      1.375*0.625=(0.375+1)*0.625 = 0.375*0.625 + 0.625 =0.859375
      0.101 *      0.625_10*
      0.11         0.375_10
    ----------------------------------
   00.001111 +       0.234375_10  +
    0.101000         0.625 _10
-------------------------------------
```

00.110111 **0.859375 $_{10}$** Revenim la reprezentarea pe 4 biţi (Q3) :
$0.110_2 = 0.75_{10}$ Rotunjirea datorată numărului limitat de biţi.

c) Utilizarea proprietăţii înmulţirii întregi cu 2: $A*B = A/2 * B + A/2 * B$

Exemplu. $0.625 * 1.375 = 0.859375$, reprezentaţi în Q3.

$1.375/2=0.6875_{10} = 0.1011_2$ care în format Q3 rotunjit este $0.110_2 = 0.75_{10}$
$0.625_{10} * 0.75_{10} = 0.46875_{10}$
$0.46875_{10} + 0.46875_{10} = 0.9375_{10} \approx 0.875_{10}$

$0.101_2 * 0.110_2 = 00.011110_2$

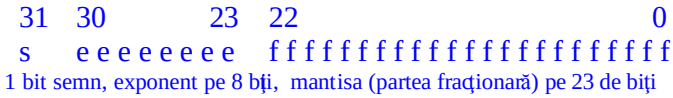
$00.011110_2+00.011110_2 = 00.111100_2 \approx 0.111_2$ (în format Q3).

5. Reprezentarea în virgulă mobilă

Nucleul unui procesor în virgulă mobilă e o unitate aritmetică ce suportă operaţii în virgulă mobilă în conformitate cu standardul IEEE 754/85. Un exemplu tipic pentru această clasă e familia x86 de la Intel începând cu procesoarele 486. Procesoarele în virgulă mobilă sunt foarte eficiente când se operează cu date în virgulă mobilă şi permit efectuarea unei game mari de calcule numerice. Aceste procesoare nu sunt aşa de eficiente în cazul controlului sarcinilor (manipularea biţilor, controlul intrărilor şi ieşirilor, răspunsul la întreruperi) şi în plus sunt destul de scumpe.
Formatul IEEE 754/85 cuprinde numere finite ce pot fi în baza 2 (binare) sau în baza 10 (zecimale). Valoarea numerică a numărului finit va fi dat de formula: $(-1)^s \times f \times b^e$, unde b reprezintă baza numărului. Fiecare număr este descris prin cei 3 parametrii: s - semnul (zero sau unu), f – mantisa şi e – exponentul. De exemplu, dacă semnul este 1 (un nr negativ), mantisa este 12345, exponentul este -3 şi se consideră baza 10, atunci numărul va fi: -12.345.

Toate valorile finite posibile de reprezentare într-un anumit format sunt determinate de: baza de numeraţie, numărul maxim de digiţi din mantisă (ce definesc precizia p) şi valoarea maximă a exponentului, emax. Mantisa trebuie să fie un număr întreg în domeniul [0; b^p-1]. Exponentul trebuie să fie un întreg a.î. 1-emax≤ q+p-1≤emax.

Standardul IEEE 754/85 defineşte 5 formate de bază: trei formate binare (pot coda numere pe 32, 64 sau 128 biţi) şi două zecimale (pot coda numere pe 64 sau 128 biţi). Formatul binar pe 32 biţi pentru precizie simplă în virgulă mobilă descrie numerele în virgulă mobilă astfel:



S= bit de semn
e…e= 8 biţi ce reprezintă exponentul
f…f= 23 de biţi ce reprezintă mantisa (biţi fracţionari)

Avantaj: exponentul oferă o gamă dinamică mare de reprezentare a numerelor
Dezavantaj: precizia de reprezentare a numerelor depinde de exponent

Reprezentarea în virgulă mobilă - Obţinerea biţilor:

- Bitul de semn:
  Negativ: bitul 31 = 1
  Pozitiv : bitul 31 = 0
- Mantisa:

$$M = 1+m_1 2^{-1}+m_2 2^{-2}+\ldots=1+\sum_{i=1}^{23} m_i \times 2^{-i} \quad \text{, unde } 1 \le M < 2$$

- Exponentul:
  O valoare pe 8 biţi cu semn, memorată cu offset-ul „+127”

Astfel, calcularea unei valori v se face astfel:

$$z = (-1)^S \times M \times 2^{E-OFFSET}$$

Exemple :

Să se scrie în format virgulă mobilă următoarele numere:

1) 0x 3FE0 0000h = 0011 1111 1110 0000 0000 0000 0000 0000 B
S = 0
E = 0111 1111 = 127
M = (1).11000 = 1 + 0.5 + 0.25 = 1.75
Z = $(-1)^0$ * 1,75 * $2^{127-127}$ = 1.75

2) **Z = -2.5**
S = 1
2.5 = 1.25 * 2^1
1 = E – OFFSET
E = 128
M = 1.25 = (1).01 = 1 + 0.25
Binar: 1100 0000 0010 0000 0000 0000 0000 0000 B = **0x C020 0000h**

Reprezentarea în virgulă mobilă are şi dezavantaje:

Exemplul *:

```
   x = 10.0        (0x41200000)
 + y =  0.000000238 (0x347F8CF1)
```

`z = 10.000000238` Greşit! Nu se poate reprezenta valoarea `10.000000238` cu numere în virgulă mobilă în simplă precizie.

```
0x412000000 = 10.000000000
              10.000000238  ⇐ nu se poate reprezenta!
0x412000001 = 10.000000950
```

Astfel, numărul `10.000000238` va fi rotunjit în jos la numărul `10.000000000`.

Proprietăţi

- Adunarea numerelor reprezentate în virgulă mobilă presupune reprezentarea lor cu acelaşi exponent; rezultatul obţinut este apoi rotunjit şi normalizat dacă este cazul
- Inmulţirea a două numere reprezentate în virgulă mobilă se realizează după regula: exponentul rezultatului se va scrie pe un număr de biţi egal cu suma numărului de biţi ai exponenţilor numerelor ce se adună, iar rezultatul obţinut este apoi rotunjit şi normalizat

6. Formatul „IQ”

Până acum s-a prezentat cazul folosirii numerelor fracţionare la care punctul fracţionar se afla după MSB. În general acest punct se poate pune oriunde în reprezentarea binară. Astfel se poate obţine o rezoluţie mai mare.
"I" – reprezintă partea întreagă (integer)
"Q" – reprezintă partea fracţionară (quotient)

Avantaj: aceeaşi precizie pentru toate numerele
Dezavantaj: domeniul dinamic este limitat comparativ cu reprezentarea în virgulă mobilă

```
31                              0
s iiiiiii.qqqqqqqqqqqqqqqqqqqqqqq
ß------------------------------>
```

mantisă pe 23 de biţi

$-2^{I} + 2^{I-1} + ... + 2^{1} + 2^{0} . 2^{-1} + 2^{-2} + ... + 2^{-Q}$

Exemplul 1: Formatul I1Q3 3 0
s. qqq

Cel mai negativ număr zecimal: -1.0	=	1. 000
Cel mai pozitiv număr zecimal: +0.875	=	0. 111
Cel mai mic număr IQ zecimal negativ: $-1*2^{-3}$	=	1. 111
Cel mai mic număr IQ zecimal pozitiv: 2^{-3}	=	0. 001

Domeniul: -1.0 ... 0.875
Rezoluţia: 2^{-3}

Exemplul 2: Formatul I3Q1 3 0
sii . q

Cel mai negativ număr zecimal: -4.0	=	100. 0
Cel mai pozitiv număr zecimal: +3.5	=	011. 1
Cel mai mic număr IQ zecimal negativ: $-1*2^{-1}$	=	111. 1
Cel mai mic număr IQ zecimal pozitiv: 2^{-1}	=	000. 1

Domeniul: -4.0 ... +3.5
Rezoluţia: 2^{-1}

Exemplul 3: Formatul I1Q31

s.qqq qqqq qqqq qqqq qqqq qqqq qqqq qqqq

Cel mai negativ număr zecimal: -1.0	=	1.000 0000 0000 0000 0000 0000 0000 0000 B
Cel mai pozitiv număr zecimal: $1-1/2^{31}$	=	0.111 1111 1111 1111 1111 1111 1111 1111 B
Cel mai mic număr IQ zecimal negativ: $-1*2^{-31}$	=	1.111 1111 1111 1111 1111 1111 1111 1111 B
Cel mai mic număr IQ zecimal pozitiv: 2^{-31}	=	0.000 0000 0000 0000 0000 0000 0000 0001 B

Domeniul: [-1.0 …. +1.0)
Rezoluţia: 2^{-31}

Dacă transpunem Exemplul * în formatul IQ, vom avea:

```
   x = 10.0         (0x0A000000)
 + y =  0.000000238 (0x00000004)
_____________________________
   z = 10.000000238 (0x0A000004),
```

număr care se poate reprezenta în formatul IQ !

Folosind formatul IQ în locul reprezentării în virgulă mobilă se poate obţine o precizie mai mare în reprezentarea datelor.

7. Concluzii

Numerele întregi versus numerele fracţionare:

Domeniul:
- numerele întregi au un domeniu maxim determinat de numărul de biţi pe care se reprezintă numărul
- numerele fracţionare pot aparţine doar domeniului [-1;+1]

Precizia:
-numerele întregi au o precizie de maxim 1
-numerele fracţionare au precizia determinată de numărul de biţi

Aritmetica în virgulă fixă versus aritmetica în virgulă mobilă:
Aritmetica în virgulă mobilă este mai flexibilă decât cea în virgulă fixă. Principalul avantaj constă în accesul la o gamă dinamică mai mare a datelor reprezentate.

Exemple de procesoare:
- cu virgulă mobilă: Seria Intel Pentium, Texas Instruments C67xxDSP
- cu virgulă fixă: Motorola HC68x, Infineon C166, Texas Instruments TMS430, TMS320C5000, C2000

Procesoarele în virgulă fixă:
Pot reprezenta numere:
- întregi (aritmetica întreagă): pentru control, calcul adrese (nu sunt implicate semnale)
- fracţionare: pentru prelucrări cu semnale

Prezintă preţuri scăzute
Programare uşoară: în faza de simulare se pot detecta toate situaţiile în care este necesară corecţia rezultatului pentru a se evita saturarea.

Procesoarele în virgulă mobilă:
Numerele au mantisă şi exponent: $mantisa*2^{exponent}$
Prezintă o tehnologie foarte flexibilă
Au o gamă dinamică largă de reprezentare a numerelor
Programare uşoară: programatorul nu mai trebuie sa gestioneze cazurile de umplere (overflow) ale acumulatorului
Dezavantaj: prezintă costuri mari

8. Teme

8.1 Se dau mai jos cei 15 coeficienţi ai unui filtru FIR trece sus. Să se convertească coeficienţii în format Q15.

(h1=h15=8.99696E-3; h2=h14=-7.86406E-3; h3=h13=-3.34992E-2; h4= h12= -6.53486E-2; h5=h11=-9.8897E-2; h6=h10=-0.12856; h7=h9=-0.14897; h8=0.84375)

8.2 După modelul celor 3 exemple prezentate la formatul IQ, specificaţi valorile corespunzătoare formatului I8Q24.

8.3 Să se scrie în format virgulă mobilă numărul : **`0x BFB0 0000h`** pe 32 biţi.

8.4 TEME de studiu :

Studiaţi următoarele exemple pentru adunarea, scăderea şi înmuţirea numerelor în virgulă mobilă:

Exemplul 1. Să se efectueze adunarea numerelor 123456.7 şi 101.7654

$$123456.7 = 1.234567 * 10^{5}$$
$$101.7654 = 1.017654 * 10^{2} = 0.001017654 * 10^{5}$$

In detaliu:

```
   e=5;      z=1.234567    (123456.7)
+  e=2;      z=1.017654    (101.7654)
-------------------------------------

   e=5;     z=1.234567
+  e=5;     z=0.001017654   (după shiftare)
-------------------------------------

   e=5;   suma=1.235584654   ( suma reală este: 123558.4654)
```

Dacă se revine la reprezentarea cu 7 digiţi rezultatul va fi: `e=5; suma=1.235585` (adică suma finală va fi: `123558.5`), deci rezultatul a fost rotunjit si normalizat. Cei 3 digiţi 654 s-au pierdut, aceasta fiind eroarea de rotujire.

Exemplul 2. Să se efectueze scăderea numerelor 123457.1467 şi 123456.659:

$$123457.1467 = 1.234571 * 10^5$$
$$123456.659 = 1.234567 * 10^5 = 0.000004 * 10^5$$

In detaliu:

```
   e=5;      z=1.234571    (123457.1467)
-  e=5;      z=1.234567    (123456.659)
-------------------------------------

   e=5;      d=0.000004   ( diferenţa reală este: 0.000004877)
```

Dacă se revine la reprezentarea cu 7 digiţi rezultatul va fi: `e=-1; s=4.000000` (adică rezultatul final va fi: `0.4`), deci rezultatul a fost rotunjit si normalizat. Eroarea de rotujire este în acest caz de aprox. 20%.

Exemplul 3. Să se efectueze înmulţirea numerelor 4734.612 şi 541724.2:

```
   e=3;      z=4.734612    (4734.612)
×  e=5;      z=5.417242    (541724.2)
-------------------------------------

   e=8;     z=1.234567
×  e=8;     z=0.001017654   (după shift-are)
-------------------------------------

   e=8;   prod=25.648538980104   ( produsul real este: 2564853898.0104 )
   e=8;   prod=25.64854          (după rotunjire)
   e=9;   prod=2.564854          (după normalizare)
```