Capitolul al IV-lea
Reprezentări interne

IV.1.
INTRODUCERE

- Reprezentările interne elementare constituie un element al **arhitecturii** oricărui calculator
    - Resursă accesibilă direct programatorilor
- Structurile de date mai complicate se definesc pornind de la reprezentările interne elementare

Reprezentări elementare

- Date numerice
  - Anumite submulţimi finite ale mulţimilor numerelor întregi, respectiv raţionale
- Date "alfa-numerice" / logice
  - Caractere; valori de tip boolean
- Instrucţiuni
  - În limbaj maşină
  - Singurele reprezentări interne elementare ne-standardizate
    - Şi, evident, neportabile

Importanţa studiului reprezentărilor

- Eficienţa şi siguranţa (fiabilitatea) reprezentărilor interne
- Numerice:

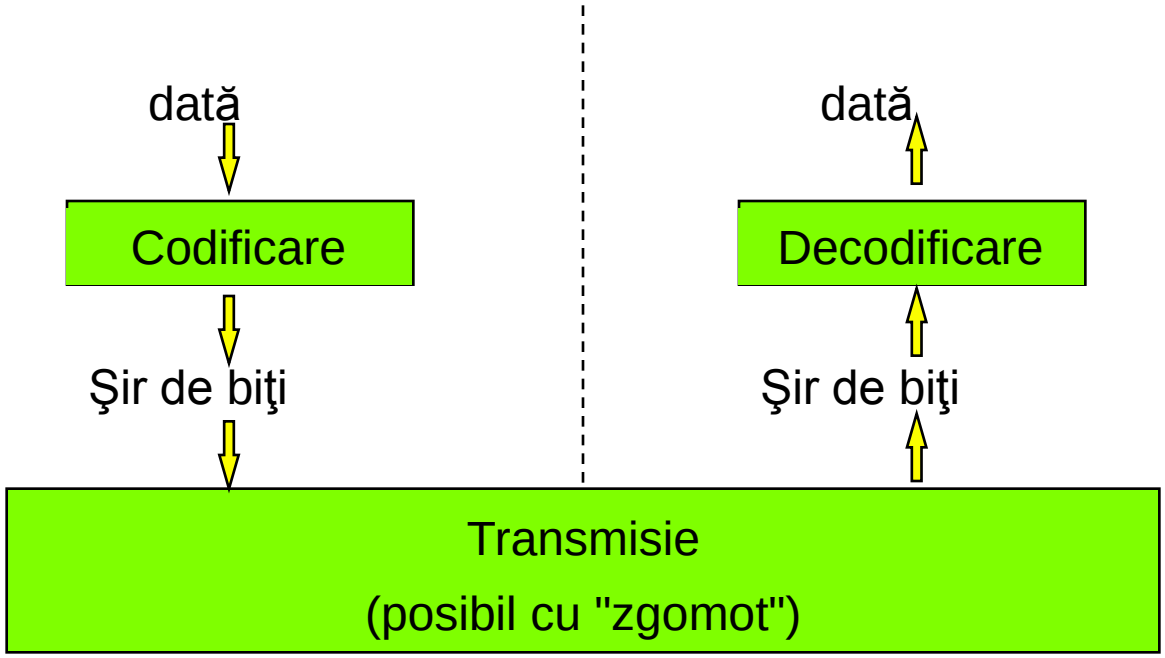
$$r(n_1)\ op\ r(n_2)\ ?=?\ r(n_1\ \mathbf{op}\ n_2)$$

- Erori inevitabile şi efectul lor
  - Mulţimi de cardinalităţi diferite
    - » $R \cap [a,b]$ şi $Q_m \cap [a,b]$)
  - Aproximări, depăşiri
  - Tratarea cazurilor de excepţie

IV.2. CODURI DETECTOARE ŞI CODURI CORECTOARE DE ERORI

Detectarea de erori

- Fiabilitatea transmisiei şi prelucrării reprezentărilor de date



Moduri de detectare/corectare

- Paritate: bit suplimentar
  - *detecţie*
  - paritate (im)pară: număr (im)par de 1
- Cod Hamming
  - *corecţie*
  - 4 biţi de informaţie, 3 biţi de paritate
  - detectarea/corecţia mai multor erori simultan

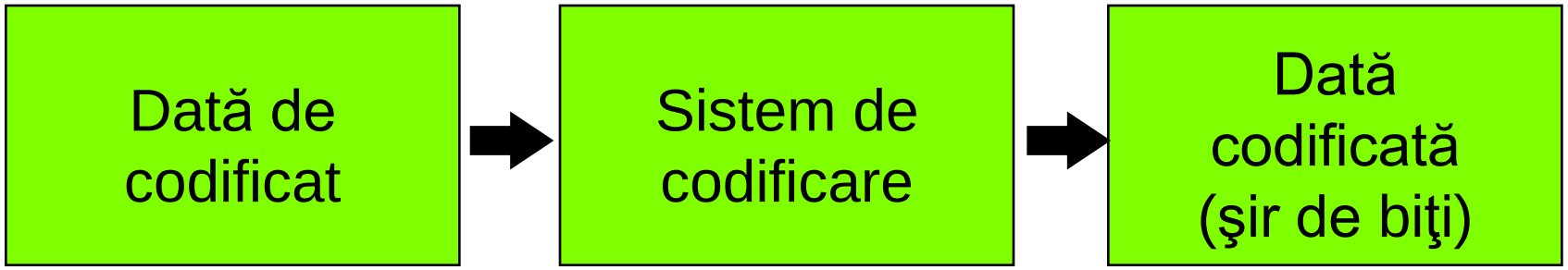
Exemplu: "paritate impară"

- Emiţător:
  - are de trimis valoarea $(110)_2$
  - generează bitul de "paritate impară" *P=1*
  - trimite $(110\mathbf{\underline{1}})_2$
- Receptor:
  - primeşte $(1101)_2$
  - verifică imparitatea numărului de 1 din şir
  - dacă nu detectează erori, elimină bitul de paritate, pentru a obţine valoarea transmisă: $(110)_2$

IV.3.
CODIFICĂRI ALFANUMERICE

Codificări alfanumerice

- Reprezentări binare ale datelor alfanumerice
  - Alfabetice, numerice, simbolurile pentru operatori, separatori etc.



Coduri alfanumerice

- ASCII
  - American Standards Committee for Information Interchange code
  - Un caracter se reprezintă pe 7 biţi plus un bit de paritate
- EBCDIC (8 biţi)
  - Extended Binary Coded Decimal Interchange Code
    - » Extinde codul binar pentru cifre zecimale
- ISO 8859-1 (Latin-1)
  - 8 biţi
  - Include şi, spre exemplu, litere cu accent
  - Ê = $CA_{(16)}$
- Supraîncărcare a şirurilor de biţi
  - Unicode
    - caractere non-latine
  - UCS
    - Universal Character Set

Codul ASCII - exemple

- 1000001 → A
- 1000010 → B
- …
- 1011010 → Z
- 1100001 → a
- 1100010 → b
- …
- 1111010 → z
- Ordine lexicografică → comparatorul binar
  - pentru 7 biţi – bitul de paritate se ignoră

IV.4. REPREZENTAREA INTERNĂ A NUMERELOR

IV.4.1.
SCRIEREA POZIȚIONALĂ

Scrierea poziţională

- Este tot o reprezentare!
  - 72018 nu este un număr, ci reprezentarea unui număr
- Inventată de arabi/indieni
  - Scrierea romană nu permite algoritmi eficienţi de calcul
- Factor implicit ataşat fiecărei poziţii din reprezentare
- Esenţială în arhitectura calculatoarelor
  - Exemplu: sumatorul serial din sumatoare complete

Baze de numeraţie

- Orice număr natural d>1
  - Cu un singur simbol nu se pot folosi factori impliciţi, ci juxtapunere + numărare
- Mulţimea cifrelor în baza d: {0,1,…,d-1}
- Calculatorul lucrează în baza d=2
  - Estimări analitice şi probabiliste: bazele în care se pot face cel mai rapid calcule sunt 2 şi 3
  - Tehnic: 2 cifre cel mai uşor de realizat
  - Teoretic: baza 2 se poate "scufunda" în logica booleană
    - ca simboli şi ca operaţii

Limite

Dacă s-ar reprezenta numerele în baza 2 fără semn (pozitive), atunci:

- Numărul maxim reprezentabil pe un octet ar fi $255 = 2^8-1$
- Numărul maxim reprezentabil pe doi octeţi ar fi $65535 = 2^{16}-1$
- Numărul maxim reprezentabil pe patru octeţi ar fi $4294967295 = 2^{32}-1$

Scrierea poziţională

- Baza d, $d \in N^* - \{1\}$:

  pentru $a_i \in \{0,1,\ldots,d-1\}$, $i=-m,\ldots,n-1$

  $$\pm (a_{n-1}a_{n-2}\ldots a_1a_0,a_{-1}\ldots..a_{-m})_{(d)} =$$
  $$\pm \sum_{i=-m}^{n-1} (a_i \times d^i)_{(10)}$$

  - a_i = valoarea celei de-a i+1[a] cifre de la stânga virgulei
    - » i=0..n-1
  - a_{-j} = valoarea celei de-a j[a] cifre de la dreapta virgulei (j>0)
    - » j=1..m
- d^i este factorul implicit pentru poziţia i
  - Se ridică la puterea i
    - d^{+1} pentru partea întreagă
    - d^{-1} pentru partea fracţionară

Baza d → baza 10

- Formula de mai sus este şi formula trecerii din baza d în baza 10
- Partea întreagă de n-1 cifre
- Partea fracţionară de m cifre

Un exemplu

- $FA2,B_{(16)} = 15\times16^{2} + 10\times16^{1} + 2\times16^{0} + 11\times16^{-1} = 3840 + 60 + 2 + 11 / 16 = 4002,6875_{(10)}$

Baza 10 → baza d

- $813,65_{(10)} = 1100101101,10(1001)_{(2)}$
  - » 813 / 2 = 406 + 1 / 2 → 1 (LSB_i)
  - » 406 / 2 = 203 + 0 / 2 → 0
  - » 203 / 2 = 101 + 1 / 2 → 1
  - » 101 / 2 = 50 + 1 / 2 → 1
  - » 50 / 2 = 25 + 0 / 2 → 0
  - » 25 / 2 = 12 + 1 / 2 → 1
  - » 12 / 2 = 6 + 0 / 2 → 0
  - » 6 / 2 = 3 + 0 / 2 → 0
  - » 3 / 2 = 1 + 1 / 2 → 1
  - » 1 / 2 = 0 + 1 / 2 → 1 (MSB_i)
  - » 0,65 / 2^{-1} = 1 + 0,3 → 1 (MSB_f)
  - » 0,3 / 2^{-1} = 0 + 0,6 → 0
  - » 0,6 / 2^{-1} = 1 + 0,2 → 1
  - » 0,2 / 2^{-1} = 0 + 0,4 → 0
  - » 0,4 / 2^{-1} = 0 + 0,8 → 0
  - » 0,8 / 2^{-1} = 1 + 0,6 → 1 (LSB_f)
  - » …. (perioadă)

Aproximarea reprezentării

- Dacă numărul are mai multe cifre la partea fracţionară decât admite codificarea, atunci există o aproximare
  - de cel mult 2^{-k}, dacă m=k
  - dacă există la partea întreagă mai multe cifre decât se pot reprezenta, atunci se produce **depăşire**

Conversii între baze care sunt puteri ale acelaşi număr

- $d_1 = 8 = 2^3$; $d_2 = 16 = 2^4$
- $703{,}102_{(8)} =$

  $= 111\ 000\ 011\ ,\ 001\ 000\ 010\ _{(2)} =$

  $= 0001\ 1100\ 0011\ ,\ 0010\ 0001\ 0000\ _{(2)} =$

  $= 1C3{,}21_{(16)}$

IV.4.2. REPREZENTĂRILE BCD ŞI ÎN EXCES

- Coduri poziţionale
  - Sisteme de numeraţie bazate pe scrierea poziţională
    - de exemplu, codul BCD
- Coduri non-poziţionale
    - De exemplu, codul Excess-k
        - » k=3 etc.
        - » în general, $k=2^p-1$
- Pentru aplicaţii tip business, numerele se pot reprezenta ca şiruri de cifre în baza 10, fiecare cifră fiind reprezentată pe 4 biţi
    - BCD, Excess

Codurile BCD și Excess-3

- Zecimal BCD Excess-3

0	0000	0011
1	0001	0100
2	0010	0101
...	...	...
7	0111	1010
8	1000	1011
9	1001	1100

- $1413_{(10)} = 0001\ 0100\ 0001\ 0011_{(BCD)}$

Adunarea BCD

$5 = 0101 +$

$3 = 0011$

$8_{(10)} = 1000 = 8_{(BCD)}$

$5 = 0101 +$

$8 = 1000$

$13_{(10)} = 1101$

$\neq 13_{(BCD)} =$ 0001 0011

- Problemele apar atunci când suma cifrelor depăşeşte 9

Adunarea BCD

Soluţie: se adună 6 (0110) atunci când suma depăşeşte 9.

Temă: De ce?

$5 = 0101 +$

$8 = 1000$

1101

$6 = 0110 +$

$1\ 0011 = 1\ 3_{(BCD)}$

$9 = 1001 +$

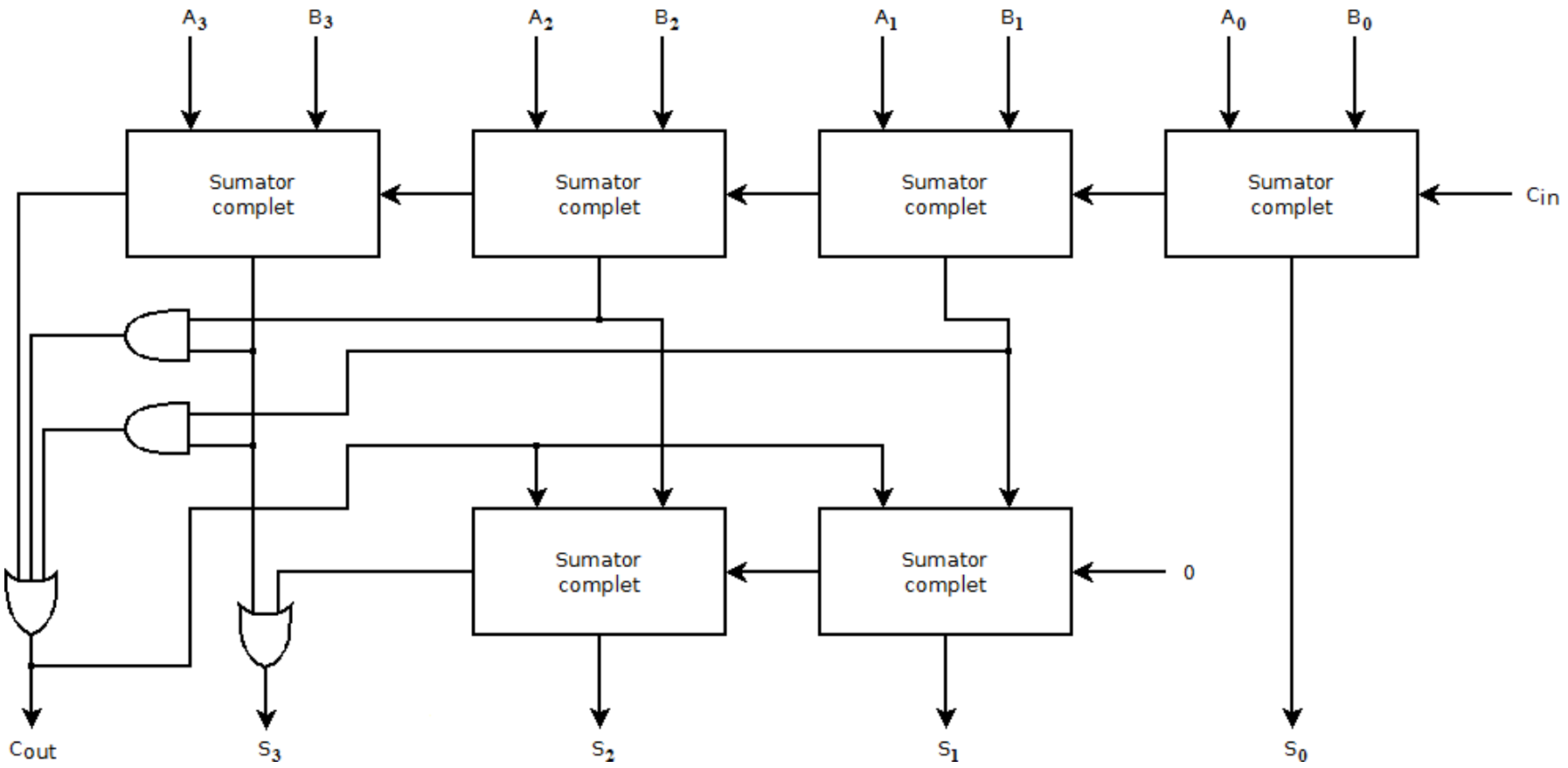
$7 = 0111$

$16_{(10)} = 1\ 0000_{(2)} \neq 16_{(BCD)}$

$6 = 0110$

$1\ 0110 = 1\ 6_{(BCD)}$

Sumator BCD



Se adună 0110 la sumă dacă ea depăşeşte 1001 (11XX sau 1X1X)

IV.4.3.
Reprezentarea numerelor întregi:
aritmetica în virgulă fixă

- Omogenitate
  - Nu se reprezintă semne speciale ("+", "-", ".", ",")
- Semnul este dat de **un bit**
  - nu de codificarea pe mai mulţi biţi a unui caracter special (+ sau -)
  - 0 pentru plus
  - 1 pentru minus
  - excepţie: reprezentarea "cu exces"
- Pentru virgulă se ştie **poziţia**
  - dar nu se reprezintă caracterul
  - aceeaşi poziţie a virgulei pentru toate numerele: **virgulă fixă** ("aritmetică întreagă")
  - poziţie diferită de la număr la număr: **virgulă mobilă** ("aritmetică flotantă")
- Esenţial pentru portabilitate: standardizarea aritmeticii (atât întreagă, cât şi flotantă)
  - În toate implementările
    - » Funcţiile elementare definite şi calculate la fel
    - » Cazurile de excepţie tratate la fel

Codificări în virgulă fixă

- $d = 2 \rightarrow a_i \in B = \{0,1\}$
- Obiectiv: eficienţa calculului
- Codificare / decodificare facilă (omogenitate)
- Algoritmi eficienţi
  - În particular, un singur algoritm pentru adunare şi scădere
- Codificările în virgulă fixă se fac pe **n+m** biţi
  - $m \geq 0$; m=0 – numere întregi
  - $n \geq 1$; n=1 – numere subunitare
- Codificări în virgulă fixă
  - Cantitate şi semn
  - Complement faţă de 1
    - » în general, faţă de cifra maximă
  - Complement faţă de 2
    - » în general, faţă de bază

Codificări redundante

- Reprezentarea numerelor pozitive coincide la cele trei codificări.
- O codificare se numeşte *redundantă* dacă există numere care au două reprezentări diferite.
- În codificările în virgulă fixă folosite, singurul număr ce poate avea două reprezentări este 0.

IV.4.4.
Reprezentarea prin modul şi semn

Reprezentarea prin modul şi semn

- A+S
- $Val_{A+S}^{n,m}(a_{n-1}a_{n-2}\ldots.a_1a_0a_{-1}\ldots..a_{-m}) =$

$$\begin{cases} a_{n-2} \times 2^{n-2} + \ldots + a_{-m} \times 2^{-m}, & \text{dacă } a_{n-1} = 0 \\ -(a_{n-2} \times 2^{n-2} + \ldots + a_{-m} \times 2^{-m}), & \text{dacă } a_{n-1} = 1 \end{cases}$$

- Coincide cu scrierea în baza 2
  - dar semnul este un bit şi virgula implicită

Reprezentarea prin modul şi semn

- Pe n+m biţi există 2^{n+m} reprezentări diferite (şiruri diferite de biţi)
- Ele corespund la 2^{n+m} -1 numere diferite
  - Redundantă, căci $0 = val_{A+S}^{n,m}(00\ldots0) = val_{A+S}^{n,m}(10\ldots0)$
- Cel mai mic număr reprezentabil este

  $$min_{A+S}^{n,m} = val_{A+S}^{n,m}(11\ldots1) = -(2^{n-1}-2^{-m})$$
- Cel mai mare număr reprezentabil este

  $$Max_{A+S}^{n,m} = val_{A+S}^{n,m}(01\ldots1) = 2^{n-1}-2^{-m}$$
- Intervalul pe care se află numerele reprezentabile este $[-(2^{n-1}-2^{-m}); +(2^{n-1}-2^{-m})]$

Reprezentarea prin modul şi semn

- Numerele reprezentabile **exact** sunt cele începând cu min = $-(2^{n-1}-2^{-m})$, cu pasul 2^{-m}
- Celelalte numere din interval se reprezintă aproximativ, cu eroare de cel mult 2^{-m}
- Precizia reprezentării este 2^{-m}
  - pentru numere întregi, precizia este 1
- Pentru n+m fixat
  - creşterea magnitudinii duce la aproximare mai slabă
  - precizie mai bună duce la magnitudine scăzută

Exemple

- $Val_{A+S}^{8,0}(00110011) = 51$

$00110011 \rightarrow + (2^0 + 2^1 + 2^4 + 2^5) = 51$

- $Val_{A+S}^{6,2}(00110011) = 12{,}75 = 51 : 2^2$

$00110011 \rightarrow + (2^{-2} + 2^{-1} + 2^2 + 2^3) = 12{,}75$

- $Val_{A+S}^{4,4}(00110011) = 3{,}1875 = 51 : 2^4$

$00110011 \rightarrow + (2^{-4} + 2^{-3} + 2^0 + 2^1) = 3{,}1875$

Exemple

- $Val_{A+S}^{8,0}(10110011) = -51$

$10110011 \rightarrow -(2^0 + 2^1 + 2^4 + 2^5) = -51$

- $min_{A+S}^{8,0} = val_{A+S}^{8,0}(11111111) = -(2^7-2^0) = -(128-1) = -127$
- $max_{A+S}^{8,0} = val_{A+S}^{8,0}(01111111) = 2^7-2^0 = 128-1 = 127$
- [-127; 127] → 255 numere, din 1 în 1
- $Val_{A+S}^{4,4}(10110011) = -3,1875$

  $10110011 \rightarrow -(2^{-4} + 2^{-3} + 2^0 + 2^1) = -3,1875$
- $min_{A+S}^{4,4} = val_{A+S}^{4,4}(11111111) = -(2^3-2^{-4}) = -7,9375$
- $max_{A+S}^{4,4} = val_{A+S}^{4,4}(01111111) = 2^3-2^{-4} = 8-0,0625 = 7,9375$
- [-7,9375; 7,9375] → 255 numere din 0,0625 în 0,0625

Operaţii A+S

- Algoritmi de complexitate relativ mare
  - Adunare / scădere
    - Stabilirea semnului rezultatului (comparaţie lexicografică)
    - Implementarea algoritmilor uzuali de adunare / scădere manuală
      - Incluzând "împrumuturi" etc.
  - Înmulţirea / împărţirea – analog celor manuale

IV.4.5. Reprezentarea în complement faţă de 1

Reprezentarea în complement faţă de 1

- Complement faţă de 1: C_1
- $Val_{C1}^{n,m}(a_{n-1}a_{n-2}\ldots a_1a_0a_{-1}\ldots a_{-m}) =$

$$\begin{cases} a_{n-2} \times 2^{n-2} + \ldots + a_{-m} \times 2^{-m}, & \text{dacă } a_{n-1} = 0 \\ (a_{n-2} \times 2^{n-2} + \ldots + a_{-m} \times 2^{-m}) - (2^{n-1} - 2^{-m}), & \text{dacă } a_{n-1} = 1 \end{cases}$$

- Temă: negativ pentru $a_{n-1}=1$
  - Deci a_{n-1} reprezintă semnul

Reprezentarea în complement faţă de 1

- Cele 2^{n+m} reprezentări diferite (şiruri diferite de biţi) corespund la $2^{n+m}-1$ numere diferite
  - Redundantă: 0 poate fi reprezentat şi ca număr negativ
- Cel mai mic număr reprezentabil este

$$\min_{C1}^{n,m} = val_{C1}^{n,m}(10\ldots0) = -(2^{n-1} - 2^{-m})$$

- Cel mai mare număr reprezentabil este

$$Max_{C1}^{n,m} = val_{C1}^{n,m}(01\ldots1) = 2^{n-1}-2^{-m}$$

- Intervalul pe care se află numerele reprezentabile este deci $[-(2^{n-1}-2^{-m}) ; +(2^{n-1}-2^{-m})]$

Reprezentarea în complement faţă de 1

- $Val_{C1}^{8,0}(00110011) = 51$

$00110011 \rightarrow + (2^0 + 2^1 + 2^4 + 2^5) = 51$

- $Val_{C1}^{6,2}(00110011) = 12{,}75 = 51 : 2^2$

$00110011 \rightarrow + (2^{-2} + 2^{-1} + 2^2 + 2^3) = 12{,}75$

- $Val_{C1}^{4,4}(00110011) = 3{,}1875 = 51 : 2^4$

$00110011 \rightarrow + (2^{-4} + 2^{-3} + 2^0 + 2^1) = 3{,}1875$

Reprezentarea în complement faţă de 1

- $Val_{C1}^{8,0}(10110011) = -76$

$10110011 \rightarrow (2^0 + 2^1 + 2^4 + 2^5) - (2^7 - 2^0) = 51 - 127 = -76$

- $min_{C1}^{8,0} = val_{C1}^{8,0}(10000000) = 0 - (2^7 - 2^0) = 0 - 127 = -127$
- $max_{C1}^{8,0} = val_{C1}^{8,0}(01111111) = 2^7 - 2^0 = 128 - 1 = 127$
- [-127; 127] → 255 numere, din 1 în 1.
- $Val_{C1}^{4,4}(10110011) = -4,75 = -76 : 2^4$

$10110011 \rightarrow (2^{-4} + 2^{-3} + 2^0 + 2^1) - (2^3 - 2^{-4}) = 3,1875 - 7,9375 =$
$= -4,75$

- $min_{C1}^{4,4} = val_{C1}^{4,4}(10000000) = 0 - (2^3 - 2^{-4}) = -7,9375 = -127 : 2^4$
- $max_{C1}^{4,4} = val_{C1}^{4,4}(01111111) = 2^3 - 2^{-4} = 8 - 0,0625 = 7,9375 = 127 : 2^4$
- [-7,9375; 7,9375] → 255 numere din 0,0625 în 0,0625

C_1 - complementare

- Dată reprezentarea lui q, se poate afla automat reprezentarea lui –q?
  - Dacă da, atunci scăderea p-q devine adunare, după generarea automată a reprezentării lui –q: p - q = p + (-q)
- Reprezentarea lui –q: complementul faţă de 1 al reprezentării lui q
- Exemplu: q = -76 = $Val_{C1}^{8,0}$ (10110011)
  q = 76 = $Val_{C1}^{8,0}$ (01001100) = 64+8+4
- Din cauza redundanţei şi a adunării preciziei (în sumă algebrică, de două ori), algoritmii de calcul în C_1 sunt mai puţin eficienţi decât cei în C_2
- De aceea, reprezentarea cvasi-general utilizată este C_2