Reprezentarea în complement faţă de 2

- cerințe
  - reprezentare neredundantă
    - o singură reprezentare pentru 0
  - adunarea a două numere se poate realiza cu un singur sumator
    - la fel ca la numere fără semn
    - beneficiu - o singură operație de adunare implementată în procesor pentru tipuri de date cu semn și fără semn

Complement faţă de 2

- notație: C_2

$$val_{C_2}^{n,m}(a_{n-1}a_{n-2}\ldots a_1a_0a_{-1}\ldots a_{-m}) =$$

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

- temă: demonstrați că valoarea este negativă pentru $a_{n-1} = 1$
  - deci a_{n-1} reprezintă semnul

Complement față de 2 - limite

- pe n+m biți sunt 2^{n+m} reprezentări diferite
  - și 2^{n+m} numere diferite
  - 00...0 - singura reprezentare pentru 0
- valorile extreme reprezentabile

$$\max_{A+S}^{n,m} = val_{A+S}^{n,m}(01\ldots1) = 2^{n-1} - 2^{-m}$$

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

  - deci numerele reprezentabile sunt în intervalul $[-2^{n-1}; +(2^{n-1}-2^{-m})]$ - asimetric

Complement față de 2 - precizie

- numerele reprezentabile exact încep cu $min=-2^{n-1}$
  - și continuă cu pasul 2^{-m}
- celelalte numere din interval - aproximare
  - eroarea - cel mult 2^{-m}
- deci precizia reprezentării este 2^{-m}
- pentru n+m fixat
  - numere mai mari = precizie mai slabă și invers

Complementare (1)

- există o relație între reprezentarea numărului *q* și cea a numărului -*q*?
- da: reprezentarea lui -*q* este complementul faţă de 2 al reprezentării lui *q*
  - se neagă toți biții și se adună 0...01
  - la fel ca la C_1, operația este comutativă - se poate aplica indiferent de semnul *q*

Complementare (2)

- exemplu

  $q = 77$ are reprezentarea 01001101 în $C_2^{8,0}$

  $-q = -77$ are reprezentarea 10110010 + 00000001 = 10110011

- temă
  - reprezentarea în C_2 pe N biţi a numărului întreg negativ q este de fapt reprezentarea pe N biţi a numărului $q + 2^N = 2^N - |q|$

Exemple (1)

$$\mathrm{val}_{C_2}^{8,0}(00110011) = 2^5 + 2^4 + 2^1 + 2^0 = 51$$

$$\mathrm{val}_{C_2}^{6,2}(00110011) = 2^3 + 2^2 + 2^{-1} + 2^{-2} = 12{,}75$$

sau

$$\mathrm{val}_{C_2}^{6,2}(00110011) = \mathrm{val}_{C_2}^{8,0}(00110011) : 2^2 = 51 : 4 = 12{,}75$$

$$\mathrm{val}_{C_2}^{4,4}(00110011) = 2^1 + 2^0 + 2^{-3} + 2^{-4} = 3{,}1875$$

sau

$$\mathrm{val}_{C_2}^{4,4}(00110011) = \mathrm{val}_{C_2}^{8,0}(00110011) : 2^4 = 51 : 16 = 3{,}1875$$

Exemple (2)

$$val_{C_2}^{8,0}(11001101) = (2^6 + 2^3 + 2^2 + 2^0) - 2^7 = 77 - 128 = -51$$

$$val_{C_2}^{4,4}(11001101) = (2^2 + 2^{-1} + 2^{-2} + 2^{-4}) - 2^3 = -3{,}1875$$

sau

$$val_{C_2}^{4,4}(11001101) = val_{C_2}^{8,0}(11001101) : 2^4 = -51 : 16 = -3{,}1875$$

$$\min_{C_2}^{8,0} = val_{C_2}^{8,0}(10000000) = 0 - 2^7 = -128$$

$$\min_{C_2}^{4,4} = val_{C_2}^{4,4}(10000000) = 0 - 2^3 = -8$$

sau

$$\min_{C_2}^{4,4} = \min_{C_2}^{8,0} : 2^4 = -128 : 16 = -8$$

Exemple (3)

$$\max_{C_2}^{8,0} = \mathrm{val}_{C_2}^{8,0}(01111111) = 127$$

$$\max_{C_2}^{4,4} = \mathrm{val}_{C_2}^{4,4}(01111111) = 7{,}9375$$

sau

$$\min_{C_2}^{4,4} = \min_{C_2}^{8,0} : 2^4 = 127 : 16 = 7{,}9375$$

- intervale reprezentabile
  - $C_2^{8,0}$: [-128; 127] $\rightarrow$ 256 numere, din 1 în 1
  - $C_2^{4,4}$: [-8; 7,9375] $\rightarrow$ 256 numere, din 0,0625 în 0,0625 (=1:16)

Concluzii

- C_2 este reprezentarea utilizată cel mai des
  - neredundantă
  - adunarea/scăderea - implementate la fel ca la numere fără semn
- în practică - tipuri de date întregi din limbajele de programare
  - caz particular (m=0)
  - pentru numere reale se utilizează reprezentările în virgulă mobilă

IV.5. Depăşiri pentru operaţii cu reprezentări în virgulă fixă

Depășiri

- nu sunt suficienți biți la partea întreagă pentru numărul de reprezentat
  - numărul este în afara intervalului reprezentabil
- problemă
  - având două numere reprezentabile, rezultatul unei operații efectuate asupra lor poate să nu fie reprezentabil - *depășire*
  - când se întâmplă și cum detectăm așa ceva?

Trecerea la reprezentări mai lungi

- având un număr reprezentat pe *n* biți, cum obținem reprezentarea sa pe *n*+*k* biți?
  - adăugare de cifre nesemnificative la partea întreagă; partea fracționară nu e afectată
- A+S: se adaugă *k* zerouri imediat la dreapta cifrei semn
- C_1, C_2: se repetă cifra semn de *k* ori imediat la dreapta cifrei semn

Exemple

	număr	
codificare	51	-51
$A+S^{8,0}$	00110011	10110011
$A+S^{16,0}$	0000000000110011	1000000000110011
$C_1^{8,0}$	00110011	11001100
$C_1^{16,0}$	0000000000110011	1111111111001100
$C_2^{8,0}$	00110011	11001101
$C_2^{16,0}$	0000000000110011	1111111111001101

Trecerea la reprezentări mai scurte

- folosim aceste rezultate pentru a răspunde la problema inversă
- având un număr reprezentat pe *n* biți, poate fi reprezentat pe *n*-*k* biți?
  - da, dacă și numai dacă cei *k* biți de la dreapta celui de semn au valorile ca mai înainte
  - în acest caz, ei pot fi eliminați din reprezentare

Operaţii în C_2

- în continuare vom discuta doar cazul C_2
  - reprezentarea cel mai des folosită
- restricții impuse de calculator asupra operațiilor cu reprezentări
  - termenii sumei şi rezultatul se reprezintă pe acelaşi număr de biţi
  - termenii înmulțirii se reprezintă pe acelaşi număr de biţi, iar rezultatul pe număr dublu de biți

Definiția depășirii

- fie o reprezentare dată şi *op* o operaţie cu numere
- pe n+m biți, numerele reprezentabile sunt în intervalul [min; max]
- fie două numere a, b $\in$ [min; max]
- operaţia *op* aplicată numerelor a şi b produce depăşire dacă

$$a \; op \; b \notin [min; max]$$

Exemple (1)

- în continuare vom folosi reprezentarea C_2 cu n=4, m=0

1111 + 1111 = 11110 → 1110

– bitul "suplimentar" este ignorat (doar 4 biți)

– de fapt este bitul de transport

$$val_{C_2}^{4,0}(1111) = -1$$

$$val_{C_2}^{4,0}(1110) = -2$$

– rezultatul este corect - nu avem depășire

Exemple (2)

$0111 + 0111 = 1110 \rightarrow 1110$

- nu avem bit "suplimentar" (transportul este 0)

$\mathrm{val}_{C_2}^{4,0}(0111) = 7$

$\mathrm{val}_{C_2}^{4,0}(1110) = -2$

- rezultat incorect - se produce depășire

- concluzie - bitul de transport nu oferă informații privind depășirea
  - trebuie căutată altă formă de detecție

Condiţia de depăşire

- nu putem folosi direct definiția depășirii
  - numerele nu sunt disponibile
- constatare
  - depăşire se poate produce la adunare doar când ambii operanzi au acelaşi semn
    - iar reprezentarea rezultatului indică semn opus
- temă: nu se poate produce depășire la adunarea a două numere de semn opus

Suma algebrică în C_2 (1)

- Teorema 1

  dacă numerele a și b sunt reprezentabile în $C_2^{n,m}$, atunci a ± b sunt reprezentabile în $C_2^{n+1,m}$

- Lemă

dacă $a = \mathrm{val}_{C_2}^{n+1,m}\left(\alpha_n \alpha_{n\text{-}1} \ldots \alpha_1 \alpha_0 \alpha_{\text{-}1} \ldots \alpha_{\text{-}m}\right)$ și $\alpha_n = \alpha_{n\text{-}1}$

atunci $a = \mathrm{val}_{C_2}^{n,m}\left(\alpha_{n\text{-}1} \ldots \alpha_1 \alpha_0 \alpha_{\text{-}1} \ldots \alpha_{\text{-}m}\right)$

Suma algebrică în C_2 (2)

- fie reprezentările

$\alpha = \alpha_{n-1}\alpha_{n-2}\ldots\alpha_1\alpha_0\alpha_{-1}\ldots\alpha_{-m}$

$\beta = \beta_{n-1}\beta_{n-2}\ldots\beta_1\beta_0\beta_{-1}\ldots\beta_{-m}$

- definim suma lor formală $\gamma = \alpha + \beta$ ca

$\gamma = \gamma_n\gamma_{n-1}\gamma_{n-2}\ldots\gamma_1\gamma_0\gamma_{-1}\ldots\gamma_{-m}$

adică

$$\sum_{i=-m}^{n}\left(\gamma_i \times 2^i\right) = \sum_{i=-m}^{n-1}\left(\left(\alpha_i + \beta_i\right) \times 2^i\right)$$

Suma algebrică în C_2 (3)

- Teorema 2

  dacă suma algebrică a numerelor reprezentate de α şi β nu produce depăşire, atunci reprezentarea rezultatului este

  $$\gamma_{n-1}\gamma_{n-2}\cdots\gamma_1\gamma_0\gamma_{-1}\cdots\gamma_{-m}$$

- Teorema 3

  suma algebrică a numerelor reprezentate de α şi β nu produce depăşire dacă cifrele transport C_{n-1} şi C_n ale rezultatului coincid

Consecințe

- adunarea a două numere în C_2 se poate realiza utilizând un sumator "clasic"
  - biții de semn se adună la fel ca toți ceilalți biți
- testarea depășirii în C_2 se poate realiza adăugând la sumator o poartă NXOR
  - în care intră biții transport C_{n-1} şi C_n
  - nu este deci necesar să fie cunoscute numerele

IV.4. Reprezentări în virgulă mobilă

Probleme cu reprezentările în virgulă fixă

- lungimea totală *n*+*m* este fixată prin hardware
- dar, în virgulă fixă, atât *n* cât și *m* sunt la rîndul lor fixate
  - deci magnitudinea și precizia sunt prestabilite și nu pot fi modificate
  - dacă dorim o precizie mai bună și suntem dispuși să reducem magnitudinea (sau invers)?

Notaţia ştiinţifică

- utilă pentru numere cu multe cifre

  $745194.2623 \times 10^{-32}$ în loc de

  0.000000000000000000000000007451942623
- același număr - mai multe forme de scriere

  $571.42 \times 10^{2} = 5.7142 \times 10^{4} = 571420 \times 10^{-1} = \ldots$
- scriere normalizată - unică
  - exact o cifră semnificativă înainte de virgulă

  5.7142×10^{4}

Notaţia ştiinţifică în binar

- cifra semnificativă dinaintea virgulei poate fi doar 1
  - deci în practică nu este necesară memorarea sa
- excepție - reprezentarea numărului 0
  - doar cifre de 0
- scriere normalizată (număr nenul)

  $1.xx...x \times 2^{y}$
  - baza 2 este implicită - nu trebuie memorată

Reprezentări în virgulă mobilă

- componente
  - semnul (S): 0 sau 1 (+ sau -)
  - mantisa (M): 1.*xx*...*x*
    - de obicei se folosește partea fracționară (f)

    M = 1 + f; f = 0.*xx*...*x*
  - caracteristica
    - reprezentarea exponentului în exces

$$N = (-1)^S \times 1.f \times 2^{C\text{ - exces}}$$

Limite

- numărul de biți al caracteristicii este fixat
  - deci există o valoare minimă și una maximă pentru exponent
- depășire superioară - exponent prea mare
  - numărul este considerat $\pm\infty$
- depășire inferioară - exponent prea mic
  - numărul este considerat 0
- tipul depășirii nu depinde de semn

Standardizare

- esențială pentru portabilitate
- standardul IEEE 754/1985
  - elaborat între 1977 și 1985
  - prima implementare comercială: Intel 8087
- 2 variante principale
  - simplă precizie (32 biți)
  - dublă precizie (64 biți)
  - au fost proiectate și unele extensii

Simplă precizie

31	30 ... 23	22 ... 0
S	C = exponent + 127	f = partea fracţionară a mantisei

- corespunde tipului *float* din C/C++
- limite în baza 10
  - minim: $\approx 1.2 \times 10^{-38}$
    - orice număr mai mic în modul va fi considerat 0
  - maxim: $\approx 3.4 \times 10^{38}$
    - orice număr mai mare în modul va fi considerat $\pm\infty$

Dublă precizie

63	62 ... 52	51 ... 0
S	C = exponent + 1023	f = partea fracţionară a mantisei

- corespunde tipului *double* din C/C++
- limite în baza 10
  - minim: $\approx 1.7 \times 10^{-308}$
  - maxim: $\approx 1.7 \times 10^{308}$
- magnitudine mai mare
- precizie superioară

Structură

- de fapt, reprezentarea în virgulă mobilă este formată din două reprezentări în virgulă fixă
  - semnul și mantisa - reprezentare modul-semn
  - caracteristica - reprezentare în exces
- de ce câmpurile sunt în ordinea S,C,f?
  - pentru a compara două reprezentări, câmpurile trebuie luate în considerare în această ordine

Caracteristici IEEE 754/1985

	Simplă precizie	Dublă precizie
Biți semn+mantisă	24	53
Exponent maxim	128	1024
• numere finite	127	1023
Exponent minim	-127	-1023
• numere normalizate	-126	-1022
Exces caracteristică	127	1023

Exemplu 1

- fie numărul -23.25
  - cum se reprezintă în simplă precizie?
- semnul: 1 (negativ)
- scriere în baza 2: $-23.25_{(10)} = -10111.01_{(2)}$
- normalizare: $10111.01 = 1.011101 \times 2^4$
- caracteristica: $4 + 127 = 131 = 10000011_{(2)}$
- reprezentarea

  $1\ 10000011\ 0111010...0_{(2)} = C1BA0000_{(16)}$

Exemplu 2

- ce număr corespunde reprezentării $42D80000_{(16)}$ (simplă precizie)?

$42D80000_{(16)} = 0\ 10000101\ 10110000...0_{(2)}$

$S = 0 \rightarrow$ număr pozitiv

$C = 10000101_{(2)} = 133_{(10)} \Rightarrow e = 133 - 127 = 6$

$M = 1 + 0.1011 = 1.1011$

- numărul: $+1.1011 \times 2^6 = 1101100_{(2)} = 108_{(10)}$

Aritmetica extinsă

- în plus față de aritmetica numerelor reale
  - reprezentarea numărului ∞ și definirea regulilor elementare de calcul cu acesta
    - x / ∞, $x \times \infty$, $\infty \pm \infty$
  - reprezentare pentru rezultatul operaţiilor nedefinite (NaN - Not a Number) şi definirea regulilor de propagare a acestuia
    - NaN *op* x = NaN, $\forall$*op*
- utilizare - bibliotecile de funcţii matematice

Exemplu

- calculul funcției arccos cu formula

$$\arccos(x) = 2 \cdot \arctan\sqrt{(1-x)/(1+x)}$$

- care este valoarea arccos(-1)?

$$x = -1 \Rightarrow (1-x)/(1+x) = 2/0 = \infty \Rightarrow$$

$$\Rightarrow \arctan\sqrt{(1-x)/(1+x)} = \pi/2$$

- răspuns: arccos (-1) = π
  - nu ar fi fost posibil de obținut fără aritmetica extinsă

Tipuri de valori în virgulă mobilă

tip valoare	exponent (e)	f	valoare
normalizată	$e_{min} < e < e_{max}$	orice valoare	$(-1)^S \times 1.f \times 2^e$
denormalizată	$e = e_{min}$	$f \neq 0$	$(-1)^S \times 0.f \times 2^e$
zero	$e = e_{min}$	$f = 0$	S 0 (= 0)
infinit	$e = e_{max}$	$f = 0$	S ∞ ($\pm\infty$)
NaN	$e = e_{max}$	$f \neq 0$	NaN

Depășiri

- depăşire inferioară
  - în forma normalizată, exponentul negativ nu poate fi reprezentat în câmpul caracteristicii
    - numărul va fi considerat 0
- depăşire superioară
  - în forma normalizată, exponentul pozitiv nu poate fi reprezentat în câmpul caracteristicii
    - numărul va fi considerat ±∞, după caz

Reprezentări denormalizate

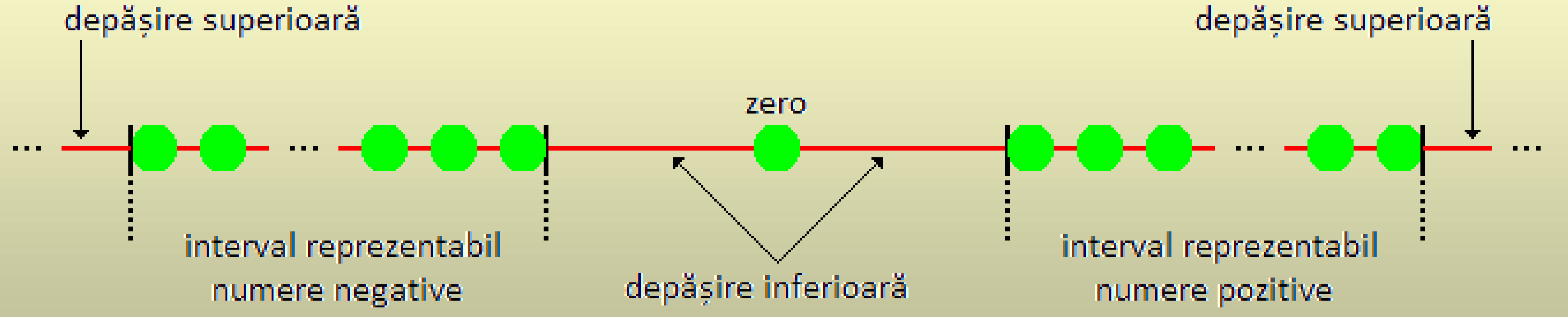
- număr mai mic în modul decât cea mai mică reprezentare normalizată nenulă
  - se renunţă la normalizare
  - exponentul are valoarea minimă
    - simplă precizie: -127
    - dublă precizie: -1023
  - în acest caz, mantisa va fi 0.f, în loc de 1.f

Aproximări (1)

- depășirea inferioară este de fapt aproximare
  - un număr nenul foarte mic este considerat 0
- care este precizia reprezentării în virgulă mobilă?
  - depinde de exponent
  - simplă precizie: 2^{e-23}
  - dublă precizie: 2^{e-52}
- exponent foarte mare - precizie foarte slabă

Aproximări (2)

- exemplu: e = 123
  - diferența între două numere consecutive reprezentabile exact este $2^{123-23} = 2^{100} \approx 10^{30}$
- ce putem reprezenta exact?
  - numere raționale (nu reale) - doar o parte



Aritmetica în virgulă mobilă

- fie două numere

$$x = m_x \cdot 2^{e_x}$$

$$y = m_y \cdot 2^{e_y}$$

- operațiile aritmetice elementare

$$x + y = \left(m_x \cdot 2^{e_x - e_y} + m_y\right) \cdot 2^{e_y}, \text{dacă } e_x \le e_y$$

$$x - y = \left(m_x \cdot 2^{e_x - e_y} - m_y\right) \cdot 2^{e_y}, \text{dacă } e_x \le e_y$$

$$x \cdot y = \left(m_x \cdot m_y\right) \cdot 2^{e_x + e_y}$$

$$x : y = \left(m_x : m_y\right) \cdot 2^{e_x - e_y}$$

Adunarea în virgulă mobilă

- compararea exponenților
  – se egalizează prin deplasarea unei mantise
- adunarea mantiselor
  – în complement față de 2
- normalizarea sumei
  – dacă apare depășire - oprire
- rotunjirea mantisei rezultat la numărul permis de biți

Înmulţirea în virgulă mobilă

- adunarea exponenților
- înmulțirea mantiselor
- normalizarea produsului
  - dacă apare depășire - oprire
- rotunjirea mantisei rezultat la numărul permis de biți
- determinarea semnului rezultatului

Temă

- urmăriți paşii adunării și respectiv înmulțirii în virgulă mobilă pentru reprezentările numerelor scrise în baza zece ca -0.75 şi 0.375
- se va considera reprezentarea în simplă precizie