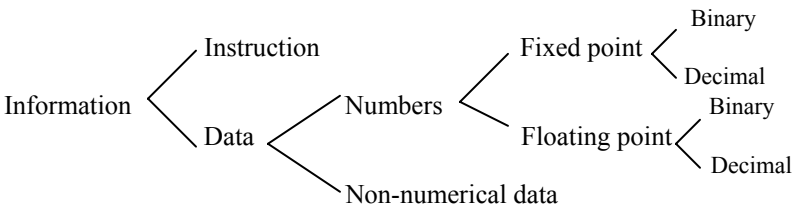
Lucrarea 1

Reprezentarea Numerelor în Virgulă Fixă

Prima lucrare prezintă modul în care este reprezentată informația în calculator şi detaliază reprezentările numerelor în virgulă fixă, analizând avantajele şi scăderile fiecărui tip de codificare: semn-mărime, complement de 1, complement de 2. În plus, sunt soluționate probleme practice precum shiftarea numerelor în diferite reprezentări de virgulă fixă dar şi conversia numerelor în reprezentare cu set redundant de cifre.

1 Reprezentarea Numerelor în Calculator
1.1 Clasificarea Informației

Informația în calculator este organizată în cuvinte (words). Cuvântul este unitatea de informație care are lungime fixă. El admite *n* cifre binare sau biți, unde *n* este în primul rând determinat din considerente legate de costul hard. În general este un multiplu de byte
Clasificarea informației:



ASCII (American Standard Code for Information Interchange)

1.2 Formate pentru Numere

Pentru selectarea unei reprezentări de număr care să fie utilizată în calculator trebuie să se țină cont de următoarele:
- tipul numerelor de reprezentat;
- domeniul valoric care trebuie acoperit de reprezentare;
- precizia numerelor, strict legată de acuratețea maximă a reprezentării;
- costul hardware-ului necesar memorării numerelor;

Există două formate de reprezentare a numerelor:
- formate de virgulă mobilă(flotantă);
- formate de vigulă fixă;

Formatele de virgulă fixă au un domeniu mai restrâns, dar şi costul circuisticiihardware este mai mic.

1.2.1 Reprezentarea Numerelor în Vigulă Fixă

Este dedusă în mod direct din forma zecimală ordinară a numerelor, în conformitate cu care numerele prezintă o virgulă zecimală, la partea stângă a acesteia corespunzând partea întreagă a numărului, iar la partea dreaptă corespunde partea zecimală. Fiecare poziție are asociată o pondere, fapt pentru care se spune că numerele sunt scrise într-o notație ponderată: „notație poziționlă“.

$$N = \sum_{i=-m}^{n} c_i \cdot r^i \qquad (1.1)$$

unde r este baza sistemului de numerație, $0 \le c_i < r$. Ponderile pozitive ale notației poziționale (având index $i \ge 0$) definesc partea întreagă a numărului N, iar ponderile negative ($i < 0$) definesc partea fracționară.

1.2.1.1 Numere binare

Pentru o reprezentare uniformă a numerelor binare cu semn, din cei n biți considerați disponibili, unul este sacrificat pentru semn, şi prin convenție este bitul situat în partea stângă (bitul cel mai semnificativ). Tot prin convenție se atribuie „0“ pentru semnul „+“ şi „1“ pentru semnul „-“.

Biții pot fi numerotați de la dreapta la stânga (cum se întâmplă la SPARC), caz în care se numeşte reprezentare *Little Endian*, sau de la

stânga la dreapta (cum se întâmplă la familia INTEL), caz în care se numeşte reprezentare *Big Endian*.

Cele două moduri de numerotare se vor reprezenta mai jos:

$$c_{n-1}c_{n-2}\ldots c_1c_0\cdot c_{-1}\ldots c_{-m} \qquad (1.2)$$

$$c_{n-1}c_{n-2}\ldots c_1c_0\cdot \qquad (1.3)$$

În Ecuația 1.2 semnul • indică punctul binar ce defineşte numerele fracționare, iar pentru Ecuația 1.3 numerele întregi. În mod clar, virgula nu necesită o poziție binară suplimentară, ci este implicită: la clasa numerelor întregi este admisă la dreapta poziției cea mai puțin semnificativă, iar la cele fracționare e implicit situată între primii doi biți, cei mai semnificativi biți din stânga

$$x_{n-1}x_{n-2}\ldots x_1x_0\cdot \qquad \text{integers} \qquad (1.4)$$

$$x_{n-1}\cdot x_{n-2}\ldots x_1x_0 \qquad \text{fractions} \qquad (1.5)$$

Există 3 reprezentări fundamentale ale numerelor binare în virgulă fixă:

a) Sign-magnitude (semn-mărime)

$$X=\underbrace{x_{n-1}}_{sign}\ \underbrace{x_{n-2}\ldots x_i\ldots x_1x_0}_{magnitude}\cdot \qquad \text{integers} \qquad (1.6)$$

$$X=\underbrace{x_{n-1}}_{sign}\cdot\underbrace{x_{n-2}\ldots x_i\ldots x_1x_0}_{magnitude} \qquad \text{fractions} \qquad (1.7)$$

În ceea ce priveşte domeniul valoric, pentru numerele întregi (Ecuația 1.6) avem gama valorică:

$$0\le|X|\le 2^{n-1}-1 \qquad (1.8)$$

Iar pentru numere fracționare gama valorică este:

$$0\le|X|\le 1-2^{-n+1} \qquad (1.9)$$

În reprezentarea semn-mărime operațiile de înmulțire şi împărțire pot

fi implementate în mod direct, dar dificultatea cea mai mare apare la operația de adunare – cea mai frecvent folosită în sistemele de calcul. Două exemple de adunare în semn-mărime pot fi edificatoare în acest sens.

$$\begin{array}{rcl} +3 & = & 0011 \\ +(-3) & = & \underline{1011} \\ & & 1110 = -6 \neq 0 \end{array} \qquad \begin{array}{rcl} +1 & = & 0001 \\ +(-6) & = & \underline{1110} \\ & & 1111 = -7 \neq -5 \end{array} \qquad (1.10)$$

Când efectuăm operația de adunare între numere care diferă ca semn trebuie să se țină cont de semn, iar această situație complică algoritmul de însumare şi circutele corespunzătoare, reprezentând o scădere a codificării semn-mărime. Pentru cifra '0', a cărei testare în calculator se efectuează frecvent, avem două reprezentări:

$$+0 = 00...0 \quad \text{şi} \quad -0 = 10...0 \qquad (1.11)$$

b) One's complement (complement de 1 – C1)

În conformitate cu această reprezentare, fiecare cifră binară este substituită prin valoarea ei complementară, adică '0' este substituit prin '1' şi invers. Față de reprezentarea semn-mărime, numerele pozitive au aceiași reprezentare, dar diferă reprezentarea numerelor negative. Complementul de 1 nu mai are o notație pozițională. Oricum, şi în cazul complementului de 1 există două scăderi, caracterizate prin corecția „end-around carry".

$$\begin{array}{rcl} -2 & = & 1101 \\ -4 & = & \underline{1011} \\ & & 11000 \\ & & \underline{0001} \\ & & 1001 = -6 \end{array} \qquad \begin{array}{rcl} 7 & = & 0111 \\ +(-3) & = & \underline{1100} \\ & & 10011 \\ & & \underline{0001} \\ & & 0100 = +4 \end{array} \qquad (1.12)$$

La ambele exemple apare un transport (carry) la bitul de semn, reprezentat prin '1' (în plus) din fața rezultatului, şi care va fi adunat la rezultat pentru efectuarea corecției de tip „end-around carry".
A doua scădere constă în faptul că '0' are şi în acest caz două reprezentări:

$$+0 = 00...0 \quad \text{şi} \quad -0 = 11...1 \qquad (1.13)$$

c) Two's complement (complement de 2 – C2)

La fel ca la complementul de 1, numerele pozitive rămân cu reprezentarea identică cu cea în semn-mărime, dar diferă atunci când vorbim de reprezentarea celor negative. Obținerea reprezentării în complement de 2 se face astfel: se complementează bit cu bit partea de mărime a reprezentării în semn-mărime, după care se adună un '1' la rezultatul obținut. Se poate folosi următoarea notație pentru numărul X din ecuațiile (1.6) şi (1.7): X este numărul în semn-mărime, $\overline{X}$ este în complement de 1, iar $-\overline{X}$ este în complement de 2. Astfel, reprezentarea în complement de 2 se poate scrie precum în ecuațiile (1.14) – numere întregi şi (1.15) – numere fracționare.

$$-\overline{X} = \overline{x}_{n-1}\overline{x}_{n-1}\ldots\overline{x}_1\overline{x}_0 \cdot +1 \ (\mathrm{mod}\, 2^n) \qquad (1.14)$$

$$-\overline{X} = \overline{x}_{n-1} \cdot \overline{x}_{n-1}\ldots\overline{x}_1\overline{x}_0 + 00\ldots01 \ (\mathrm{mod}\, 2) \qquad (1.15)$$

Nu s-a luat în considerație carry-ul care apare la bitul de semn. Pentru reprezentarea numerelor avem următoarele caracteristici:

- toate reprezentările au x_{n-1} ca bit de semn;
- ambele reprezentări complementare pentru numere negative nu au corespondent într-o notație pozițională;
- în codurile complementare operația de scădere se realizează prin adunarea complementului;
- în codurile complementare, exceptând cazul „end-around carry", bitul de semn nu revendică o tratare specială. El poate fi tratat ca un bit ordinar de mărime;
- apare anomalia complementului de 2, de exemplu pentru o reprezentare pe 4 biți -8 poate fi reprezentat (are un cod disponibil), chiar dacă acest număr nu poate fi reprezentat pe 4 biți în semn-mărime şi complement de 1.
- overflow–underflow (sitații de depăşire): la operația de adunare a două numere cu semn se spune că apare depăşire atunci când semnul rezultatului diferă de cel (identic) al operanzilor.

Considerăm adunarea a două numere în complement de doi pe *n* biți, *X*+*Y*=*Z*, unde reprezentarea binară este $X = x_{n-1}x_{n-2}\ldots x_1x_0$, $Y = y_{n-1}y_{n-2}\ldots y_1y_0$ şi $Z = z_{n-1}z_{n-2}\ldots z_1z_0$.

x_i	y_i	c_i	z_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

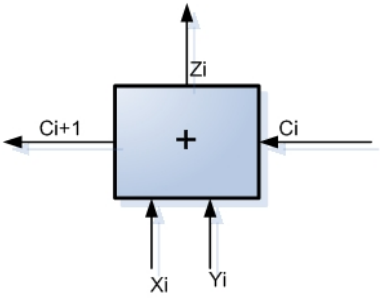


Figura 1.1 Tabela de adevăr şi simbolul logic pentru sumatorul complet pe 1 bit (1-bit full adder).

Ecuațiile fundamentale pentru sumatorul complet pe 1 bit, care adună x_i cu y_i, (i. pentru un carry-in c_i, rezultând bitul sumă z_i şi carry-out-ul c_{i+1}), conform cu tabela de adevăr din Figura 1.1, sunt:

$$\begin{cases} z_i = x_i \oplus y_i \oplus c_i \\ c_{i+1} = x_i y_i + x_i c_i + y_i c_i \end{cases} \qquad (1.16)$$

Detecția situației de overflow (v) se face pentru celula (sumator complet pe 1 bit) cea mai semnificativă (i=n-1): $v = x_{n-1} \cdot y_{n-1} \cdot \overline{z_{n-1}} + \overline{x_{n-1}} \cdot \overline{y_{n-1}} \cdot z_{n-1}$. Prin inspecția tabelei din Figura 1.1 sau prin deducție la nivel de ecuații booleene, se poate demonstra că:

$$v = c_n \oplus c_{n-1} \qquad (1.17)$$

2 Aplicații

Problema 1.1 (Shiftarea numerelor în complement de 1 şi de 2) Shiftarea la dreapta sau la stânga este folosită pentru a dubla/înjumătăți mărimea întrebilor binari fără semn. Cum putem realiza shiftarea pentru numerele în complement de 1 şi de 2?

Rezolvare:

La complementul de 1 avem shiftarea la stânga/dreapta din Figura 1.2, unde se explică şi modul de shiftare: bitul care vine pe poziția lăsată liberă este identic cu cel de dinainte de shiftare (shiftarea la dreapta – right), sau este '1' (shiftarea la stânga).

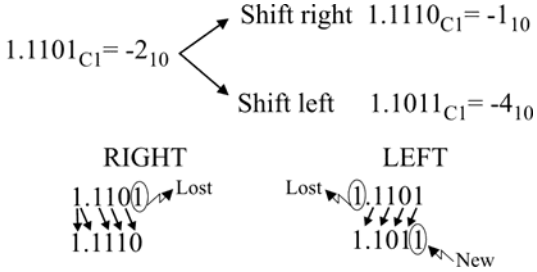


Figura 1.2 Shiftarea în complement de 1.

La complementul de 2, Figura 1.3 ne prezintă maniera în care se shiftează la stânga/dreapta: bitul de pe poziția rămasă liberă va fi identic cu cel de dinainte de shiftare (right shift), sau are valoarea '0' (left shift).

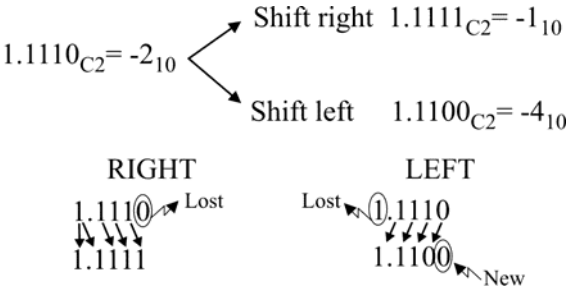


Figura 1.3 Shiftarea în complement de 2.

Problema 1.2 (Adunarea numerelor în complement de 1 şi de 2) Să se adune următoarele perechi de numere în semn-mărime, complement de 1 şi de 2: +47 şi +75, -47 şi +75, +47 şi -75, -47 şi -75 (numere exprimate în zecimal).

Rezolvare:
Figura 1.4 prezintă adunarea seturilor de numere cerute. În semn-mărime, doar adunarea numerelor pozitive este încununată de succes.

	X=+ 47 + Y=+ 75 Z=+122	X= - 47 + Y=+ 75 Z=+ 28	X=+ 47 + Y= - 75 Z=- 28	X= - 47 + Y= - 75 Z=- 122
SM	X=00101111+ Y=01001011 Z=01111010	—	—	—
C1		11010000+ 01001011 (1)00011011+ ▶1 00011100 28_{10}	00101111+ 10110100 11100011 -28_{10}	11010000+ 10110100 (1)10000100+ ▶1 10000101 -112_{10}
C2		11010001+ 01001011 00011100 28_{10}	00101111+ 10110101 00011100 -28_{10}	11010001+ 10110101 10000110 -112_{10}

Figura 1.4 Adunarea seturilor de numere în SM, C1 şi C2.

Problema 1.3 (Propusă) Să se indice o metodă de transformare a numerelor exprimate în sistemele $\{\bar{1},0,1\}$ şi $\{\bar{2},\bar{1},0,1,2\}$ redundante de reprezentare.

Indicaţie: În sistemul $\{\bar{1},0,1\}$, numărul -19_{10} se poate scrie ca $\bar{1}0\bar{1}01 = -1\times 2^4 - 1\times 2^2 + 1\times 2^0$, iar -457_{10} se poate scrie ca $\bar{1}210\bar{2}\bar{1} = -1\times 4^5 + 2\times 4^4 + 1\times 4^3 - 2\times 4^1 - 1\times 4^0$. Mecanismul de conversie în

binar este sugerat în Figura 1.5, atât pentru primul, cît si pentru cel de-al doilea sistem de reprezentare a numerelor întregi pe un set redundant de cifre.

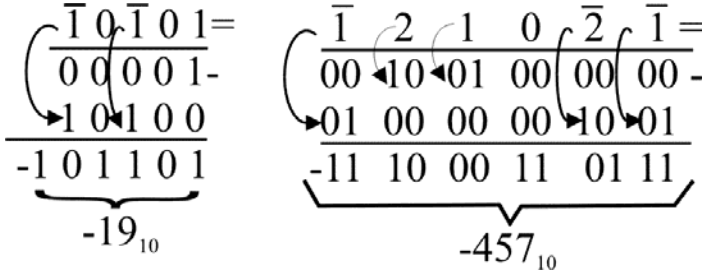


Figura 1.5 Conversia din sistemul de reprezentare cu set redundant de cifre în binar.