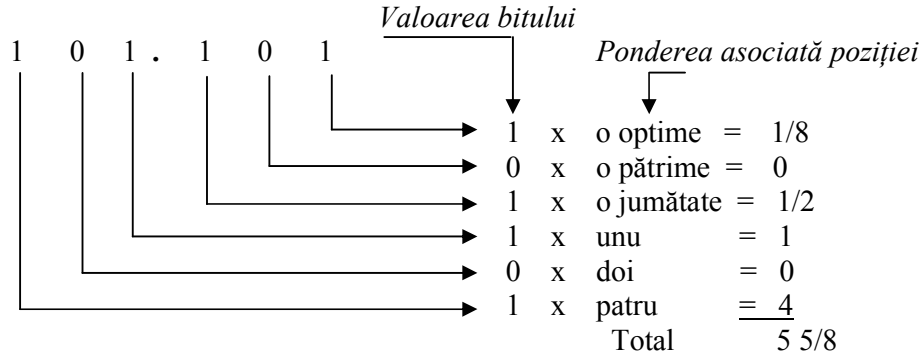
ARHITECTURA CALCULATOARELOR 2003-2004
CURSUL 3

1.4.2 Reprezentarea fracţiilor în sistemul binar

Pentru a extinde notaţia binară astfel încât să fie adecvată reprezentării valorilor fracţionare, vom utiliza notaţia în **virgulă fixă** (**radix point**): reprezentarea binară conţine un punct care are acelaşi rol ca şi virgula utilizată în notaţia zecimală. Aceasta înseamnă că cifrele de la stânga punctului reprezintă partea întreagă a valorii şi sunt interpretate ca şi în sistemul binar prezentat anterior, iar cifrele de la dreapta punctului reprezintă partea fracţionară a valorii şi sunt interpretate într-o manieră similară cu cea a celorlalţi biţi, cu excepţia faptului că poziţiile lor au asociate ponderi fracţionare. Astfel prima poziţie din dreapta punctului are atribuită ponderea 1/2, următoarea poziţie corespunde ponderii 1/4, următoarea corespunde ponderii 1/8 şi aşa mai departe. Observaţi ca aceasta nu înseamnă altceva decât aplicarea în continuare a regulii stabilite anterior: fiecare poziţie are alocată o pondere de două ori mai mare decât cea a poziţiei din dreapta sa. Având aceste ponderi asociate poziţiilor biţilor, decodificarea unei valori binare reprezentate în virgulă fixă se face la fel ca şi în cazul în care n-am fi avut punctul care separă partea fixă de cea fracţionară. În particular, vom înmulţi valoarea fiecărui bit cu ponderea asociată poziţiei acelui bit. De exemplu, decodificarea reprezentării binare 101.101 va avea ca rezultat 5 5/8, după cum se artă şi în figura 1.19.

Figura 1.19 Decodificarea reprezentării binare 101.101



În plus, tehnicile aplicate la sistemul de numeraţie în baza zece se pot folosi de asemenea şi în binar. Asta înseamnă că pentru a aduna două reprezentări binare în virgulă fixă, vom alinia unul sub altul punctele de separare între părţile întregi şi fracţionare şi vom aplica acelaşi proces de adunare ca şi cel prezentat anterior.

De exemplu, 11.011 adunat cu 100.11 produce rezultatul 111.001, după care se poate observa în continuare:

$$\begin{array}{r} 10.011 \\ +100.11 \\ \hline 111.001 \end{array}$$

1.5 Stocarea numerelor întregi

Atunci când avem nevoie de o tehnică eficientă pentru reprezentarea numerelor întregi ca şiruri de biți, prima idee este să recurgem la notația binară prezentată în sub capitolul 1.4. Acest lucru nu este însă posibil, pentru că adesea avem nevoie să memorăm atât valori pozitive, cât şi negative. Ne trebuie deci un sistem de notație care să permită reprezentarea atât a numerelor întregi pozitive, cât şi a celor negative. Matematicienii au dezvoltat domeniul sistemelor de notație pentru numere, iar unele dintre ideile lor s-au dovedit a fi foarte potrivite cu modul de realizare a circuitelor electronice şi de aceia sunt utilizate pe scară largă în cadrul echipamentelor de calcul. În acest capitol vom prezenta două astfel de sisteme de notație: notația în exces (excess notation) şi notația în complement față de doi.

1.5.1 Notația în exces

O metodă de reprezentare a valorilor întregi o reprezintă **notația în exces** (**excess notation**). Valorile din sistemul de reprezentare în exces sunt codificate utilizând cuvinte binare de aceeaşi lungime. Pentru a realiza un sistem de notație în exces, stabilim mai întâi lungimea cuvintelor binare utilizate, apoi scriem una sub alta toate combinațiile posibile în ordinea în care ar apărea dacă am număra în binar. În acest context, vom observa că primul cuvânt binar care are pe poziția bitului cel mai semnificativ valoarea 1 survine aproximativ la jumătatea listei. Vom considera că acest model reprezintă valorile 1,2 3…; iar cuvintele binare care-l preced sunt utilizate pentru codificarea valorilor negative –1, –2, –3, …. În figura 1.20 este înfățişat codul care rezultă în cazul în care folosim cuvinte cu lungimea de patru biți. Puteți observa că numărul 5 este reprezentat de combinația 1101, iar –5 este codificat cu 0011. Remarcați că în sistemul de notație în exces se poate face cu uşurință diferența între cuvintele care reprezintă valori negative şi cele care reprezintă valori pozitive. Primele au 0 pe poziția celui mai semnificativ bit , iar celelalte au 1. Cel mai semnificativ bit este adesea denumit **bit de semn** (**sign bit**). În cadrul notației în exces, un bit egal cu 0 indică o valoare negativă, iar un bit de semn egal cu 1 indică o valoare pozitivă sau valoarea zero.

Figura 1.20 Tabelul de conversie pentru sistemul de notație în exces cu opt

Cuvânt Binar	*Valoarea reprezentată*
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	-1
0110	-2
0101	-3
0100	-4
0011	-5
0010	-6
0001	-7
0000	-8

Figura 1.21 Sistem de notație în exces care utilizează cuvinte cu lungimea de trei biți

Cuvânt Binar	*Valoarea reprezentată*
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

Sistemul reprezentat în figura 1.20 este cunoscut sub numele de notația în exces cu opt. Pentru a înțelege de ce, să interpretăm mai întâi fiecare dintre cuvintele de cod utilizând sistemul binar clasic şi apoi să comparăm aceste rezultate cu valorile reprezentate în codul în exces. În fiecare caz, veți descoperi că valoarea rezultată în urma interpretării binare este mai mare cu 8 decât cea rezultată în urma interpretării în exces. De exemplu, cuvântul 1100 reprezintă în codificarea normală valoarea 12, dar în cazul codificării în exces reprezintă valoarea 4; 0000 reprezintă în mod normal valoarea zero, dar în cadrul codificării în exces reprezintă valoarea –8. Într-o manieră asemănătoare, un sistem de reprezentare în exces bazat pe cuvinte cu lungimea de 5 biți va purta denumirea de notație în exces cu 16, deoarece, de exemplu, cuvântul 10000 va fi utilizat pentru reprezentarea valorii 0, față de cazul codificării binare naturale, în care reprezintă valoarea 16. Similar, vă puteți convinge că un sistem de notație în exces pe trei biți poate fi numit şi notație în exces cu patru (figura 1.21).

Pe baza acestor observații obținem o metodă rapidă de codificare a valorilor utilizând notația în exces. Pentru a reprezenta o valoare în notația în exces cu opt, trebuie să adunăm 8 la acea valoare, să scriem rezultatul în baza 2 şi apoi să adăugăm în față biți de 0 (dacă este nevoie) pentru a obține un cuvânt cu lungimea de 4 biți. De exemplu, pentru a codifica valoarea 5 utilizând notația în exces cu opt, adunăm mai întâi 8, obținând valoarea 13, apoi scriem această valoare în binar obținând 1101, care este chiar cuvântul binar dorit. Pentru a afla cuvântul care corespunde valorii –5 adunăm 8 şi obținem valoarea 3, care în binar se scrie 11, astfel că modelul de patru biți pe care-l căutăm este 0011.

1.5.2 Notația în complement față de doi

Cel mai popular sistem de reprezentare a numerelor întregi în cadrul calculatoarelor actuale este **notația în complement față de doi** (**two's complement notation**). Ca şi sistemul de notație în exces, acest sistem utilizează un număr fix de biți pentru a reprezenta valorile din cadrul sistemului. Figura 1.22 prezintă sistemele în complement față de doi bazate pe cuvinte de cod de lungime 3 şi 4. Pentru construcția unui astfel de sistem, se începe cu un şir de biți de valoarea 0 de lungime adecvată şi apoi se numără în binar până când se ajunge la un şir de biți care începe cu un bit 0,1,2,3, … Cuvintele care reprezintă valorile negative se obțin

Figura 1.22 Sisteme de notație în complement față de doi

Utilizând cuvinte binare de lungime 3

Cuvânt binar	*Valoare reprezentată*
011	3
010	2
001	1
000	0
111	-1
110	-2
101	-3
100	-4

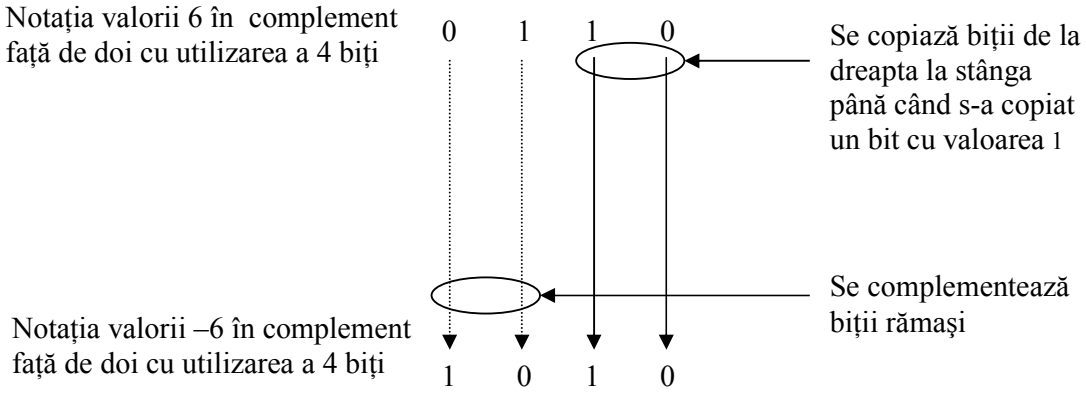
Utilizând cuvinte binare de lungime 4

Cuvânt binar	*Valoarea reprezentată*
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

începând cu un şir de biţi cu valoarea 1 şi numărând în sens descrescător până la atingerea cuvântului care începe cu un bit 1 şi are toţi ceilalţi biţi zero. Cuvintele astfel obţinute reprezintă valorile –1, –2, –3, …. (Dacă vă este greu să număraţi în sens invers binar, începeţi de la cuvântul alcătuit din 1 şi restul biţilor 0 şi număraţi ascendent până la atingerea cuvântului care conţine numai biţi 1, scriind valorile obţinute una deasupra celeilalte.)

Observaţi că în sistemul de notaţie în complement faţă de doi valorile negative sunt reprezentate de cuvinte al căror bit de semn este 1; în schimb, un bit de semn egal cu 0 indică faptul că valoarea reprezentată este zero sau pozitivă. Reţineţi de asemenea că între reprezentările valorilor pozitive şi negative cu acelaşi modul există o relaţie de corespondenţă foarte convenabilă. Aceste reprezentări sunt identice atunci când sunt citite de la dreapta la stânga, până la inclusiv primul bit de 1 din cuvânt. De aici fiecare cuvânt este complementul celuilalt. (**Complementul** unui cuvânt binar este cuvântul obţinut prin schimbarea tuturor biţilor de 1 la valoarea 0, respectiv a tuturor biţilor 0 în 1; cuvântul 0110 este complementul lui 1001 şi invers.) De exemplu, în sistemul de codificare pe patru biţi din figura 1.21 cuvintele care reprezintă valorile 2 şi –2 se termină amândouă cu 10, dar cuvântul corespunzător valorii 2 începe cu 00, în timp ce reprezentarea valorii –2 începe cu 11. Această observaţie conduce la realizarea unui algoritm care să permită obţinerea reprezentării binare pentru valori de semne contrare dar care au acelaşi modul. Vom copia cuvântul original începând de la dreapta până la apariţia unui bit cu valoarea 1, apoi vom face complementul biţilor rămaşi (vom schimba toţi biţi 1 rămaşi în 0 şi toţi biţii 0 în 1) pe măsură ce-i copiem (figura 1.23).

Figura 1.23 Codificarea valorii –6 în complement faţă de doi utilizând patru biţi



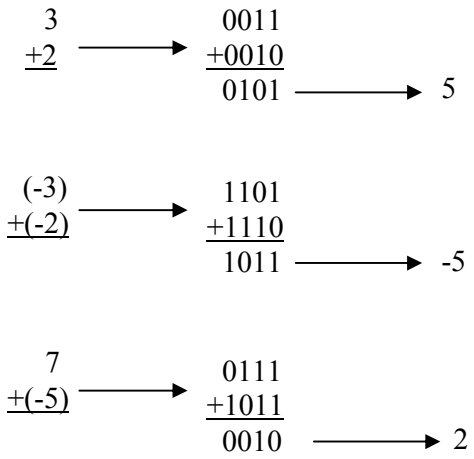
Înţelegerea acestor proprietăţii elementare ale sistemelor în complement faţă de doi conduce de asemenea la obţinerea unui algoritm pentru decodificarea reprezentărilor în complement faţă de doi. Dacă modelul de biţi ce trebuie decodificat are bitul de semn zero, se va citi direct valoarea ca şi cum cuvântul ar fi o reprezentare binară normală. De exemplu, 0110 reprezintă valoarea 6, deoarece 110 este reprezentarea binară a lui 6. Dacă şablonul care trebuie decodificat are bitul de semn 1, vom înţelege că valoarea reprezentată este negativă şi tot ce ne rămâne este să găsim modulul acelei valori. Vom realiza acest lucru copiind cuvântul original de la dreapta la stânga până la copierea unui bit de 1, apoi vom completa biţii rămaşi, îi vom copia şi în final vom decodifica cuvântul obţinut ca şi cum ar fi o reprezentare binară normală.

De exemplu, pentru a decodifica cuvântul 1010, vom observa mai întâi că bitul de semn este 1 şi deci valoarea pe care o reprezintă este negativă. În consecinţă, vom converti cuvântul în 0110, vom observa că aceasta corespunde valorii 6 şi vom trage concluzia că şablonul original reprezintă valoarea –6.

1.5.3 Adunarea numerelor reprezentate în complement faţă de doi

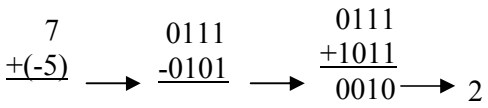
Pentru a aduna valori reprezentate în complement faţă de doi, aplicăm acelaşi algoritm ca şi pentru adunarea în binar, cu excepţia faptului că toate cuvintele binare, inclusiv rezultatul, au aceiaşi lungime. Aceasta înseamnă că atunci când se efectuează o adunare într-un complement faţă de doi, orice bit suplimentar generat la stânga răspunsului de către un transport final va fi eliminat. Astfel, "adunând" 0101 şi 0010 se obţine 0111, în timp ce "adunând" 0111 şi 1011 se obţine rezultatul 0010 (0111+1011 = 10010, care este trunchiat la 0010). Acceptând această convenţie, să considerăm cele trei probleme de adunare prezentate în figura 1.24. În fiecare caz în parte, vom codifica valorile utilizând notaţia în complement faţă de doi (pe patru biţii), vom efectua adunarea aşa cum s-a descris anterior şi vom decodifica apoi rezultatul înapoi în notaţia zecimală.

Figura 1.24 Probleme de adunare rezolvate utilizând notaţia în complement faţă de doi



Observaţi că dacă ar fi să folosim tehnicile uzuale învăţate în şcoala primară, a treia problemă ar fi un proces complet diferit de problemele precedente (este vorba de operaţia de scădere). Pe de altă parte, prin trecerea problemelor în notaţia în complement faţă de doi, putem calcula răspunsul corect aplicând în toate cazurile aceleaşi algoritm de adunare. Aceasta este prin urmare avantajul notaţiei în complement faţă de doi: adunarea oricărei combinaţii de numere cu semn se poate efectua utilizându-se acelaşi algoritm.

Spre deosebire de elevi din şcoala primară, care învaţă mai întâi adunarea şi apoi scăderea, un calculator care utilizează notaţia în complement faţă de doi trebuie să ştie numai să efectueze adunarea şi negarea biţilor. De exemplu, operaţia de scădere 7- 5 este identică cu problema de adunare 7 + (-5). În consecinţă, dacă i se cere unui calculator să scadă 5 (stocat sub forma 0101) din 7 (stocat ca 0111), acesta va schimba mai întâi pe 5 în –5 (reprezentat ca 1011) şi apoi va efectua adunarea 0111 + 1011, obţinând rezultatul 0010, care reprezintă valoarea 2, după cum este descris în continuare:



Observăm astfel că atunci că pentru reprezentarea valorilor numerice se utilizează notația în complement față de doi, este suficient să folosim un circuit pentru adunare combinat cu un circuit pentru negarea unei valori pentru a rezolva atât problema adunării, cât şi pe cea a scăderii numerelor. Dar avantajele nu se opresc aici. Înmulțirea este de fapt adunarea repetată, iar împărțirea reprezintă scăderea repetată (6/2 reprezintă de câte ori poate fi scăzut 2 din 6 fără să se obțină un rezultat negativ). Astfel că, în ultimă instanță, putem efectua toate cele patru operații aritmetice standard, adică adunarea, scăderea, înmulțirea şi împărțirea, utilizând doar cele două circuite.

1.5.4 Problema depăşirii superioare

O problemă pe care am evitat-o în problemele precedente este aceea că în oricare dintre sistemele de numerație pe care le-am prezentat există o limită privind mărimea pe care pot s-o reprezinte valorile. Atunci când utilizăm notația în complement față de doi cu cuvinte de patru biți, valoarea 9 nu are asociată nici un şablon, aşa că nu este nici o şansă să obținem un răspuns corect la efectuarea adunării 5 + 4. De fapt, rezultatul va fi –7. O problemă similară apare dacă utilizăm cuvinte cu lungimea de cinci biți şi încercăm să reprezentăm valoarea 17. O astfel de poată numele de **depăşire superioară** (**overflow**). Atunci când utilizăm notația în complement față de doi, eroarea de depăşire superioară poate apărea la adunarea a două valori pozitive sau la adunarea a două valori negative. În fiecare dintre cazuri, depăşirea poate fi detectată prin verificarea bitului de semn a răspunsului. Aceasta înseamnă că este semnalată depăşirea superioară dacă adunarea a două valori pozitive produce ca rezultat un şablon ce corespunde unei valori negative sau dacă rezultatul adunării a două valori negative apare ca fiind pozitiv.

Concluzia este aceea că şi calculatoarele pot face greşeli, aşa că persoana care utilizează calculatorul trebuie să fie atentă la pericole potențiale. Desigur, deoarece cele mai multe calculatoare manipulează cuvinte mult mai lungi decât cele pe care le-am utilizat în carte, valorile mari pot fi prelucrate fără să apară vreo eroare de depăşire. Multe calculatoare utilizează cuvinte de 32 de biți pentru stocarea valorilor utilizând notația în complement față de doi, permițând astfel lucrul cu valori de până la 2.147.483.647, fără apariția depăşiri superioare. Dacă sunt necesare şi valori şi mai mari, se foloseşte adesea tehnica denumită **dublă precizie** (**double precision**); ea permite ca lungimea cuvintelor utilizate să fie mărită față de cea utilizată de obicei de către calculator. O altă abordare a problemei o reprezintă schimbarea unităților de măsură. De exemplu, găsirea unei soluții exprimate în kilometri în loc de centimetri are ca efect utilizarea unor numere mai mici, păstrându-şi în unele cazuri precizia impusă.

1.6 Stocarea numerelor fracționare

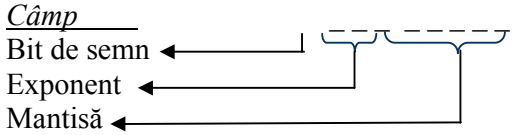
Spre deosebire de stocarea numerelor întregi, stocarea unei valori care conține o parte fracționară impune să memorăm nu numai modelele de 0 şi 1 care compun reprezentare binară, ci şi poziția virgulei zecimale. O metodă uzuală pentru a face acest lucru se bazează pe notația ştiințifică şi poartă numele de **notație în virgulă mobilă** (**floating-point notation**).

1.6.1 Notația în virgulă mobilă

Să explicăm notația în virgulă mobilă printr-un exemplu care utilizează numai un octet pentru efectuarea stocării. Cu toate că de obicei calculatoarele lucrează cu cuvinte mult mai lungi, acest exemplu este reprezentativ pentru sistemele actuale şi serveşte pentru a prezentă concepte importante, fără să ne complicăm folosind cuvinte lungi. Vom considera cel mai semnificativ bit din cadrul octetului ca fiind bitul de semn. Din nou, bitul de semn egal cu 0 înseamnă că valoarea stocată este pozitivă, iar 1 înseamnă că valoarea este strict negativă. Împărțim apoi cei şapte biți rămaşi în două grupuri dau câmpuri, **câmpul exponentului** (**exponent field**) şi **câmpul mantisei** (**mantissa field**). Să considerăm cei trei biți care urmează după bitul de semn ca fiind câmpul exponentului, iar cei patru biți rămaşi ca fiind câmpul mantisei. Octetul este deci împărțit ca în figura 1.25.

Putem explica semnificația acestor câmpuri considerând următorul exemplu. Să presupunem că un octet conține şirul de biți 01101011. Analizând acest cuvânt conform formatului precedent, putem vedea că bitul de semn este 0, exponentul este 110, iar mantisa este 1011. Pentru a decodifica octetul, extragem mai întâi mantisa şi plasăm o virgulă (punct) zecimală la stânga ei, obținând .1011

Figura 1.25 Elemente componente ale notației în virgulă mobilă



Apoi, extragem conținutul câmpului exponentului (110) şi-l interpretăm ca pe un întreg stocat utilizând metoda în exces pe trei biți (ca în figura 1.24). În acest caz, cuvântul binar din câmpul exponentului reprezintă valoarea pozitivă 2. Acest fapt ne precizează că trebuie să mutăm virgula la dreapta cu doi biți. (Un exponent negativ înseamnă că virgula trebuie deplasată la stânga.) În consecință, obținem 10.11 care înseamnă 2 3/4. Observăm apoi că bitul de semn din exemplul considerat este 0; valoarea reprezentată este pozitivă. Vom trage concluzia că 01101011 reprezintă valoarea 2 3/4. Să luăm alt exemplu, anume octetul 10111100. Extragem mantisa, obținem.1100 şi mutăm virgula cu un bit la stânga, deoarece câmpul exponentului (011) reprezintă valoarea –1. Acum avem .01100 care reprezintă 3/8. Deoarece bitul de semn din cuvântul inițial este 1, valoarea stocată este negativă. Tragem concluzia că 10111100 reprezintă valoarea – 3/8.

Pentru a stoca o valoare utilizând notația în virgulă mobilă, vom inversa procesul prezentat anterior. De exemplu, pentru a codifica valoarea 1 1/8 o vom exprima mai întâi în notație binară, obținând 1.001. Vom copia apoi cuvântul binar în câmpul rezervat mantisei de la stânga la dreapta, începând cu primul bit diferit de zero din reprezentarea binară. În acest moment octetul arată astfel: _ _ _ _ 1 0 0 1

Acum trebuie să completăm câmpul exponentului. În acest scop ne vom imagina conținutul câmpului mantisei având o virgulă zecimală la stânga şi vom determina numărul de biți şi direcția în care trebuie să fie deplasată virgula pentru a se obține numărul binar inițial. În exemplul nostru, observăm că virgula din .1001 trebuie deplasată cu un bit la dreapta pentru a se obține 1.001. Deoarece din această cauză exponentul trebuie să aibă valoarea pozitivă 1, vom plasa combinația 101 (care este reprezentarea valorii pozitive 1 st în notația în exces cu

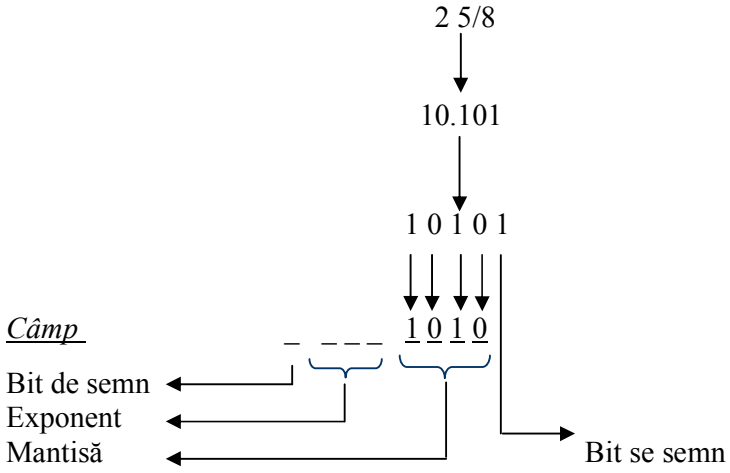
patru) în câmpul exponentului. În final vom scrie 0 în bitul de semn, deoarece valoarea stocată este zero. La sfârşit octetul arată astfel: 0 1 0 1 1 0 0 1

Înainte de a trece mai departe, să explicăm de ce este utilizată notația în exces pentru reprezentarea exponentului în cazul sistemelor în virgulă mobilă. Utilizarea notației în exces reduce operația de comparare a mărimii relative a două valori la simpla parcurgere a reprezentărilor de la stânga la dreapta, până la întâlnirea primului bit care diferă. De exemplu, dacă ambii biți de semn sunt zero, cea mai mare dintre cele două valori este aceea care are 1 pe poziția primului bit care diferă la parcurgerea de la stânga la dreapta celor două cuvinte. Astfel, dacă 00101010 şi 00011001 sunt reprezentări în virgulă mobilă, vom putea afirma că prima reprezentare corespunde unei valori mai mari, fără a trebui să determinăm mai întâi valorile care corespund efectiv celor două reprezentări.

1.6.2 Erori de rotunjire

Să studiem acum ce se întâmplă dacă încercăm să stocăm valoarea 2 5/8 utilizând sistemul în virgulă mobilă pe un octet (prezentat anterior). Scriem mai întâi valoarea 2 5/8 în binar, cea ce conduce la 11.101. Însă atunci când vom copia acest rezultat în câmpul mantisei, vom descoperi că nu avem suficient spațiu şi ca urmare ultimul 1 (cel care corespunde ultimului 1/8) se va pierde (figura 1.26). Dacă ignorăm această problemă şi continuăm cu completarea câmpului exponentului şi a bitului de semn, vom obține cuvântul 01101010, care reprezintă valoarea 2 1/2, în loc de 2 5/8. Ceea ce s-a întâmplat poartă numele de **eroare de rotunjire** (**round-off error**), cauzată în acest caz de faptul că lungimea câmpului mantisei este de patru biți, în timp ce pentru păstrarea preciziei de reprezentare este necesar un câmp de cinci biți. Soluția evidentă pentru evitarea acestei probleme constă în creşterea dimensiunii câmpului mantisei, adică exact cea ce se face în cazul calculatoarelor reale. La fel ca şi în situația stocării numerelor întregi, pentru memorarea valorilor în virgulă mobilă se utilizează de obicei cel puțin 32 de biți, în loc de 8 biți, cum am procedat noi.

Figura 1.26 Codificarea valorii 2 5/8



Această abordare permite de asemenea şi existența simultană a unui câmp al exponentului cu o lungime mai mare. Totuşi, chiar şi utilizând şi aceste formate de dimensiuni mari, mai apar uneori situați în care este necesară o acuratețe mai mare. În aceste cazuri, vom aplica din nou conceptul de dublă precizie, prezentat deja.

O altă sursă de erori de rotunjire este un fenomen cu care sunteţi obişnuiţi deja din notaţia zecimală; problema valorilor cu număr infinit de zecimale (periodice), cum ar fi de exemplu situaţia în care încercăm să exprimăm fracţia 1/3 în formă zecimală. Unele valori nu pot fi reprezentate precis, indiferent câte zecimale utilizăm.

Diferenţa dintre notaţia zecimală utilizată de noi în mod curent şi notaţia binară este aceea că în binar există mai multe valori care au reprezentări cu o infinitate de cifre. De exemplu, valoarea de o zecime este periodică atunci când este exprimată în binar. Imaginaţi-vă problemele pe care le cauzează acest fapt unei persoane neavizate care utilizează notaţia în virgulă mobilă pentru a stoca şi manevra valori ce reprezintă dolari şi cenţi. Concret, dacă se utilizează ca unitate de măsură dolarul, valoarea unei monezi de zece cenţi nu poate fi stocată în mod precis.

O soluţie la această problemă o constituie manipularea datelor de acest tip folosind ca unitate valoarea de un penny, astfel încât toate valorile să devină numere întregi, ce pot fi stocate cu precizie utilizând o metodă cum ar fi de exemplu notaţia în complement faţă de doi. Astfel de tehnici sunt reprezentative pentru cele folosite de pachetele software proiectate special pentru utilizatori care nu au o formaţie tehnică (cum ar fi sistemul de calcul tabelar). În aceste cazuri, pentru protejarea utilizatorilor de eventualele rezultate eronate, este evitată întrebuinţarea facilităţilor calculatorului pentru lucrul în virgulă mobilă.

Erorile de rotunjire, precum şi metodele legate de acestea, reprezintă preocuparea de zi cu zi a celor ce lucrează în domeniul analizei numerice. Această ramură a matematicii se ocupă cu studiul problemelor implicate de calculele reale, care sunt adesea foarte complexe şi necesită o precizie foarte mare.

Vom încheia acest capitol cu o exemplificare a unei reguli generale care va încălzi inima oricărui analist numeric. Să presupunem că ni se cere să adunăm următoarele trei valori utilizând notaţia în virgulă mobilă pe un bit (definită anterior): 2 1/2 + 1/8 + 1/8. Dacă adunăm valorile în ordinea afişată, vom aduna mai întâi 2 1/2 şi 1/8 şi vom obţine 2 5/8, care în binar este 10.101. Din nefericire, deoarece această valoare nu poate fi stocată cu acurateţe (după cum am văzut mai înainte), rezultatul primei operaţii va fi memorarea valorii 2 1/2 (care coincide cu primul termen al adunării). Următorul pas este să adunăm acest rezultat cu ultimul 1/8. Va apărea din nou o eroare de rotunjire, iar rezultatul final devine greşit 2 1/2. Să efectuăm acum operaţiile în ordine inversă. Adunăm mai întâi 1/8 cu 1/8 şi obţinem 1/4. În notaţia binară aceasta reprezintă .01; astfel încât rezultatul primei adunări va fi stocat pe un octet ca 00111000, memorarea făcându-se fără erori. Adunăm acum 1/4 cu ultima valoare rămasă, respectiv 2 1/2, şi obţinem 2 3/4, rezultat ce poate fi stocat exact pe un octet sub forma 01101011. De data aceasta am obţinut rezultatul corect.

Pe scurt, atunci când sunt adunate valori reprezentate în notaţia în virgulă mobilă, ordinea în care se fac operaţiile poate fi extrem de importantă. Regula generală este ca întotdeauna să fie adunate mai întâi valorile cele mai mici; totuşi, nici măcar acest mod de lucru nu garantează corectitudinea rezultatului.