BAZELE ARITMETICE ALE CALCULATOARELOR (II)

1. Scopul lucrării

Lucrarea prezintă reprezentarea numerelor în virgulă mobilă, standardul IEEE 754 pentru reprezentarea în virgulă mobilă, coduri binar-zecimale, coduri detectoare de erori, coduri corectoare de erori şi coduri alfanumerice.

2. Considerații teoretice

2.1. Reprezentarea numerelor în virgulă mobilă

2.1.1. Principii de reprezentare a numerelor în virgulă mobilă

În cazul reprezentării în forma cu virgulă fixă, poziția virgulei, stabilită prin proiectare, nu mai poate fi schimbată, cu toate că virgula nu mai este reprezentată fizic în calculator. Dacă virgula este amplasată după cifra de semn, se lucrează cu numere fracționare subunitare. Deoarece nu toate numerele sunt subunitare, pentru a le aduce la această formă trebuie executate o serie de operații de scalare (multiplicare cu un anumit factor de scală) sau deplasare, ataşând numerelor factori de scală. Evidența acestora trebuie realizată prin program, motiv pentru care acestea se complică.

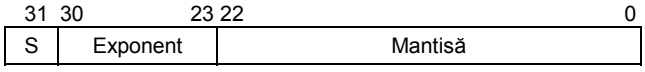
Această dificultate se poate rezolva utilizând reprezentarea în *virgulă mobilă* (*virgulă flotantă*). În acest caz, factorul de scală devine o parte a cuvântului din calculator, poziția virgulei variind pentru fiecare număr în mod automat, ceea ce conduce la simplificarea programelor.

Un număr reprezentat în virgulă mobilă (VM) are două componente. Prima componentă este *mantisa*, care indică valoarea exactă a numărului într-un anumit domeniu, fiind reprezentată de obicei ca un număr fracționar cu semn. A doua componentă este *exponentul*, care indică ordinul de mărime al numărului. Considerând un număr N, reprezentarea acestuia în VM poate avea forma următoare:

$$N = \pm M \cdot B^{\pm E} \tag{2.1}$$

unde M este mantisa, B este baza sistemului de numerație, iar E este exponentul.

Această reprezentare poate fi memorată într-un cuvânt binar cu trei câmpuri: semnul, mantisa şi exponentul. De exemplu, presupunând un cuvânt de 32 de biți, o asignare posibilă a biților la fiecare câmp poate fi următoarea:



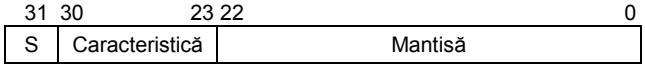
Aceasta este o reprezentare în mărime şi semn, deoarece semnul are un câmp separat față de restul numărului. Câmpul de semn constă dintr-un bit care indică semnul numărului: 0 pentru un număr pozitiv şi 1 pentru un număr negativ. Nu există un câmp rezervat pentru baza B, deoarece această bază este implicită şi ea nu trebuie memorată, fiind aceeaşi pentru toate numerele.

De obicei, în locul exponentului se reprezintă o valoare numită *caracteristică*, care se obține prin adunarea unui deplasament la exponent, astfel încât să rezulte întotdeauna o valoare pozitivă. Deci, nu se rezervă un câmp separat pentru semnul exponentului. Caracteristica C este deci exponentul deplasat:

$$C = E + deplasament \tag{2.2}$$

Valoarea reală a exponentului se poate afla prin scăderea deplasamentului din caracteristica numărului. De exemplu, dacă pentru caracteristică se rezervă un câmp de 8 biți, valorile caracteristicii pot fi cuprinse între 0 şi 255. Cu un deplasament de 128 (80h), exponentul poate lua valori între –128 şi +127, fiind negativ dacă $C < 128$, pozitiv dacă $C > 128$, şi 0 dacă $C = 128$. Exponentul este deci reprezentat *în exces* 128.

Reprezentarea cu ajutorul caracteristicii va fi următoarea:



În acest caz, semnul mantisei este acelaşi cu semnul numărului.

Unul din avantajele utilizării exponentului deplasat constă în simplificarea operațiilor executate cu exponentul, datorită lipsei exponenților negativi. Al doilea avantaj se referă la modul de reprezentare al numărului zero.

Mantisa numărului zero are cifre de 0 în toate pozițiile. Exponentul numărului zero poate avea, teoretic, orice valoare, rezultatul fiind tot zero. În unele calculatoare, dacă un rezultat are mantisa zero, exponentul este lăsat la valoarea pe care o are în momentul respectiv, rezultând un "*zero impur*".

La majoritatea calculatoarelor, se recomandă ca numărul zero să aibă cel mai mic exponent posibil, rezultând astfel un "*zero pur*". În cazul exponenților nedeplasați, exponentul cu cea mai mică valoare este cel mai mic număr negativ pe care îl poate avea exponentul, iar în cazul exponenților deplasați, această valoare este 0.

Deci, prin utilizarea caracteristicii, reprezentarea în VM a numărului zero este aceeaşi cu reprezentarea în VF, adică toate pozițiile sunt 0. Aceasta înseamnă că se pot utiliza aceleaşi circuite pentru testarea valorii zero.

În reprezentarea anterioară, mantisa constă din 23 de biți. Deşi virgula binară nu este reprezentată, se presupune că ea este aşezată înaintea bitului c.m.s. al mantisei. De exemplu, dacă $B = 2$, numărul 1,75 poate fi reprezentat sub mai multe forme:

$+0{,}111 * 2^1$	0	1000 0001	1110 0000 0000 0000 0000 000
$+0{,}00111 * 2^3$	0	1000 0011	0011 1000 0000 0000 0000 000

Pentru simplificarea operațiilor cu numere în VM şi pentru creşterea preciziei acestora, se utilizează reprezentarea sub forma normalizată. Un număr în VM este normalizat dacă bitul c.m.s. al mantisei este 1. Din cele două reprezentări ale numărului 1,75 ilustrate anterior, prima este cea normalizată.

Deoarece bitul c.m.s. al unui număr normalizat în VM este întotdeauna 1, acest bit nu este de obicei memorat, fiind un *bit ascuns* la dreapta virgulei binare. Aceasta permite ca mantisa să aibă un bit semnificativ în plus. Astfel, câmpul de 23 de biți este utilizat pentru memorarea unei mantise de 24 de biți cu valori între 0,5 şi 1,0.

Cu această reprezentare, Figura 2.1 indică gama numerelor care pot fi reprezentate într-un cuvânt de 32 de biți.

Dacă se utilizează reprezentarea în C2, se pot reprezenta toate numerele întregi între -2^{31} şi $2^{31}-1$, cu un total de 2^{32} numere diferite. Pentru formatul prezentat, se pot reprezenta numere în următoarele domenii (Figura 2.1):

- Numere negative între $-(1-2^{-24}) \cdot 2^{127}$ şi $-0{,}5 \cdot 2^{-128}$
- Numere pozitive între $0{,}5 \cdot 2^{-128}$ şi $(1-2^{-24}) \cdot 2^{127}$

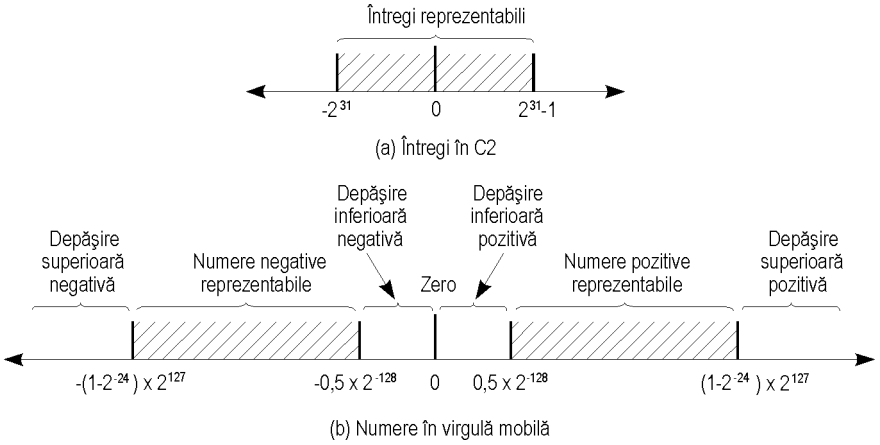


Figura 2.1. Numere reprezentabile în formate tipice de 32 de biți.

Există cinci regiuni care nu sunt cuprinse în aceste domenii:

- Numere negative mai mici decât $-(1-2^{-24}) \cdot 2^{127}$, apariția acestora determinând o *depăşire superioară negativă*
- Numere negative mai mari decât $-0{,}5 \cdot 2^{-128}$, care determină o *depăşire inferioară negativă*
- Zero
- Numere pozitive mai mici decât $0{,}5 \cdot 2^{-128}$, care determină o *depăşire inferioară pozitivă*
- Numere pozitive mai mari decât $(1-2^{-24}) \cdot 2^{127}$, care determină o *depăşire superioară pozitivă*

În unele cazuri, bitul ascuns se presupune poziționat la stânga virgulei binare. Astfel, mantisa memorată *M* va reprezenta de fapt valoarea 1,*M*. În acest caz, numărul normalizat 1,75 va avea următoarea formă:

$+1.11 * 2^0$

0	1000 0000	1100 0000 0000 0000 0000 000

Presupunând că bitul ascuns este poziționat la stânga virgulei binare în formatul prezentat, un număr normalizat diferit de zero reprezintă următoarea valoare:

$$(-1)^S * (1,M) * 2^{E-128} \tag{2.3}$$

unde *S* indică bitul de semn.

În acest format se pot reprezenta numere în următoarele domenii:

- Numere negative între $-[1 + (1 - 2^{-23})] * 2^{127}$ şi $-1{,}0 * 2^{-128}$
- Numere pozitive între $1{,}0 * 2^{-128}$ şi $[1 + (1 - 2^{-23})] * 2^{127}$

Problema care apare în cazul formatului prezentat este că nu există o reprezentare pentru valoarea zero. Aceasta deoarece valoarea zero nu poate fi normalizată. Totuşi, reprezentările în VM cuprind de obicei o combinație specială de biți pentru reprezentarea valorii zero.

Depăşirea superioară apare atunci când exponentul depăşeşte valoarea maximă, de exemplu peste 127 în cazul formatului prezentat. *Depăşirea inferioară* apare atunci când exponentul are o valoare negativă prea mică, de exemplu sub –128. În cazul depăşirii inferioare, rezultatul se poate aproxima cu 0. Coprocesoarele matematice au anumite mecanisme pentru detectarea, semnalarea şi tratarea depăşirii superioare şi a celei inferioare.

Pentru alegerea unui format în VM trebuie realizat un compromis între dimensiunea mantisei şi cea a exponentului. Creşterea dimensiunii mantisei va conduce la creşterea preciziei numerelor, iar

creşterea dimensiunii exponentului va conduce la creşterea domeniului numerelor care pot fi reprezentate. Singura cale de a creşte atât precizia, cât şi domeniul numerelor, este de a utiliza un număr mai mare de biţi pentru reprezentare. Cele mai multe calculatoare utilizează cel puţin două formate, în *simplă precizie* (de exemplu pe 32 de biţi), şi *dublă precizie* (de exemplu pe 64 de biţi).

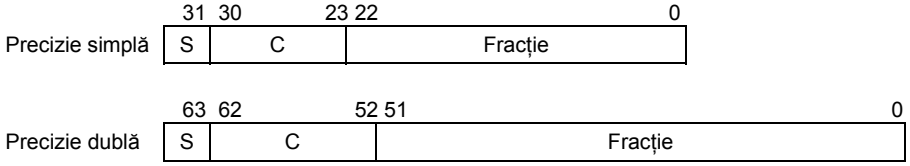
2.1.2. Reprezentarea numerelor în formatul IEEE 754

IEEE (*Institute of Electrical and Electronics Engineers*) a dezvoltat un standard pentru reprezentarea numerelor în VM şi operaţiile aritmetice în această reprezentare. Scopul era facilitarea portabilităţii programelor între diferite calculatoare. Standardul IEEE 754 a fost publicat în 1985. Cele mai multe coprocesoare aritmetice, printre care şi cele *Intel* pentru familia de microprocesoare 80x86, se conformează acestui standard.

Standardul defineşte trei formate:

- Formatul *scurt* (precizie simplă): 4 octeţi
- Formatul *lung* (precizie dublă): 8 octeţi
- Formatul *temporar* (precizie extinsă): 10 octeţi

Baza implicită este 2. Formatul *scurt* şi cel *lung* sunt prezentate în continuare.



S indică semnul numărului. *C* reprezintă exponentul deplasat, deci caracteristica, pentru care se rezervă 8 biţi în formatul scurt şi 11 biţi în formatul lung. Pentru formatul scurt, deplasamentul este 127 (7Fh), iar pentru formatul lung deplasamentul este 1023 (3FFh). Valorile minime (0) şi cele maxime (255, respectiv 2047) ale caracteristicii nu sunt utilizate pentru numerele normalizate, ele având utilizări speciale.

Bitul ascuns este utilizat şi la standardul IEEE 754, dar mantisa este reprezentată într-un mod diferit. Ea constă dintr-un bit implicit cu valoarea 1 (partea întreagă), virgula binară implicită, şi apoi cei 23, respectiv 52 de biţi ai fracţiei. Dacă toţi biţii fracţiei sunt 0, mantisa este 1,0; dacă toţi biţii fracţiei sunt 1, mantisa este cu puţin mai mică decât 2,0. Deci:

$$1{,}0 \le M < 2{,}0$$

Mantisa are valoarea:

$$M = 1{,}Fracţie$$

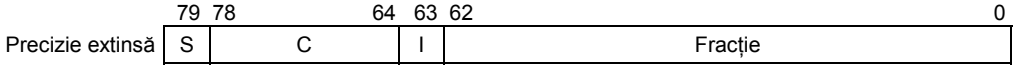
iar valoarea numărului în precizie simplă (N_S), respectiv în precizie dublă (N_D) este:

$$N_S = (-1)^s \cdot M \cdot 2^{C-127} \tag{2.4}$$

$$N_D = (-1)^s \cdot M \cdot 2^{C-1023} \tag{2.5}$$

Gama numerelor care pot fi reprezentate în precizie simplă este cuprinsă între aproximativ $2{,}2\times10^{-38}$ şi $3{,}4\times10^{38}$, iar cea a numerelor reprezentate în precizie dublă este cuprinsă între $2{,}2\times10^{-308}$ şi $1{,}7\times10^{308}$.

Formatul *temporar* este utilizat pentru reprezentarea numerelor în cadrul coprocesoarelor aritmetice, în scopul reducerii erorilor datorate rotunjirilor. Acest format este următorul:



Bitul 63 reprezintă partea întreagă a mantisei, care nu este implicită. Numerele în formatul temporar nu sunt întotdeauna normalizate, de aceea nu încep în mod obligatoriu cu un bit de 1. De aceea acest bit este reprezentat explicit, fiind notat cu *I* în cadrul formatului. Deplasamentul exponentului este 16.383, iar valoarea numărului (N_E) este:

$$N_E = (-1)^s \cdot M \cdot 2^{C\text{-}16383} \tag{2.6}$$

Una din problemele care apare la calculele cu numere în VM se referă la modul de tratare al depăşirilor inferioare şi superioare, şi la reprezentarea valorilor nedefinite. În acest scop, pe lângă numerele normalizate, standardul mai permite şi reprezentări ale unor valori speciale, pentru care sunt rezervate valorile 0 şi 255 ale exponentului. Unele din aceste valori speciale sunt prezentate în continuare.

S	0	≠0	Număr denormalizat
S	0	0	Zero
S	11…1	0	Infinit
S	11…1	≠0	NaN (Nu este număr)

În cazul obţinerii unui rezultat cu o valoare mai mică decât numărul normalizat cel mai mic posibil, în mod obişnuit rezultatul este setat la zero şi operaţiile se continuă, sau se semnalează o eroare de depăşire inferioară. Nici una din aceste soluţii nu este satisfăcătoare, astfel încât standardul permite utilizarea numerelor nenormalizate (*denormalizate*), care au caracteristica 0, iar fracţia diferită de 0.

Pentru valoarea *zero*, atât caracteristica, cât şi fracţia, sunt egale cu 0. Există două reprezentări pentru valoarea 0, în funcţie de bitul de semn: +0, respectiv -0. Bitul de la stânga punctului binar este implicit 0 în loc de 1.

Pentru cazul în care apare o depăşire superioară, există o reprezentare specială pentru *infinit*, constând din caracteristica formată din biţi de 1 (255 pentru formatul scurt), şi o fracţie egală cu 0. Valoarea infinit se poate utiliza ca operand, de exemplu:

$$\infty + n = \infty$$
$$n / \infty = 0$$
$$n / 0 = \infty$$

Astfel, utilizatorul poate decide dacă va trata depăşirea superioară ca o condiţie de eroare, sau va continua calculele cu valoarea infinit.

Pentru indicarea diferitelor condiţii de excepţie, ca în cazul nedefinirilor de forma ∞/∞, ∞ ∗ 0, 0/∞, s-a prevăzut un format special *NaN* (*Not a Number*). Caracteristica este formată din biţi de 1, iar fracţia este diferită de 0.

Exemple

1) Care este reprezentarea binară a numărului –0,75 în simplă precizie?

Numărul –0,75 poate fi scris ca – 3/4 sau – 0,11 în binar. Notaţia ştiinţifică a numărului este $-0{,}11\times2^0$, iar forma normalizată a acestei notaţii este $-1{,}1\times2^{-1}$. Caracteristica va fi –1 + 127 = 126 (7Eh). Reprezentarea numărului în precizie simplă este deci:

31	30 … 23	22 … 0
1	0111 1110	1000 0000 0000 0000 0000 000

2) Care este numărul zecimal reprezentat de următorul cuvânt?

31	30 … 23	22 … 0
1	1000 0001	0100 0000 0000 0000 0000 000

Bitul de semn este 1, câmpul rezervat caracteristicii conține 81h = 129, iar câmpul fracției conține $1 \times 2^{-2} = 0{,}25$. Valoarea numărului este:

$$(-1)^1 \times 1{,}25 \times 2^{(129-127)} = -1{,}25 \times 2^2 = -1{,}25 \times 4 = -5{,}0$$

2.2. Coduri binar-zecimale

Aceste coduri se utilizează pentru codificarea cifrelor zecimale. Pentru codificarea fiecăreia din cele 10 cifre, sunt necesari 4 biți; din cele 16 valori posibile, 6 vor fi neutilizate. Prin stabilirea unor corespondențe între mulțimea cifrelor zecimale şi mulțimea celor 16 cuvinte de 4 biți, se obțin numeroase posibilități de codificare (A_{16}^{10}). Din numeroasele coduri posibile se utilizează practic doar o mică parte.

Codurile utilizate se împart în coduri ponderate şi neponderate.

În cazul *codurilor ponderate*, o cifră zecimală este exprimată printr-o combinație de 4 cifre binare, în care fiecărei cifre i se asociază o anumită pondere. Ponderile pot fi pozitive sau negative. Valoarea cifrei zecimale se obține prin suma biților din cod, fiecare bit fiind multiplicat cu valoarea ponderii asociate.

Considerând un cod format din biții b_0, b_1, b_2, b_3, ponderile asociate acestora fiind p_0, p_1, p_2, respectiv p_3, valoarea cifrei zecimale codificate este:

$$N = p_0b_0 + p_1b_1 + p_2b_2 + p_3b_3 \tag{2.7}$$

Ponderile fiecărui bit reprezintă valoarea corespunzătoare din denumirea codului. Pentru ponderile de sus, codul are denumirea $p_3p_2p_1p_0$. În Tabelul 2.1 se prezintă exemple de coduri ponderate de 4 biți mai des utilizate.

Tabelul 2.1. Coduri binar-zecimale ponderate de 4 biți.

Nr. zecimal	8421	2421	$6\bar{4}\bar{2}3$	$8\bar{4}\bar{2}\bar{1}$
0	0000	0000	0000	0000
1	0001	0001	0101	0111
2	0010	0010	0010	0110
3	0011	0011	1001	0101
4	0100	0100	0100	0100
5	0101	1011	1011	1011
6	0110	1100	0110	1010
7	0111	1101	1101	1001
8	1000	1110	1010	1000
9	1001	1111	1111	1111

Exemple

$0101_{8421} = 8 \cdot 0 + 4 \cdot 1 + 2 \cdot 0 + 1 \cdot 1 = 5$

$1010_{8\bar{4}\bar{2}\bar{1}} = 8 \cdot 1 + 4 \cdot 0 + (-2) \cdot 1 + (-1) \cdot 0 = 8 - 2 = 6$

În cazul codului 8421, deoarece fiecare bit are ponderea numărării în binar (2^0, 2^1, 2^2, 2^3), iar cuvintele de cod reprezintă numerele succesive în sistemul binar natural, codul se mai numeşte cod binar-zecimal natural (NBCD – *Natural Binary Coded Decimal*). În mod obişnuit, acest cod se numeşte, impropriu, cod BCD.

În cazul codului 2421, numit şi *cod Aiken* (după numele prof. Howard Aiken, care a realizat calculatorul MARK I), primele 5 cifre zecimale (0 – 4) au aceeaşi exprimare ca şi în codul 8421. Cifra zecimală 5 poate fi exprimată fie prin 0101, fie prin 1011. Deci, reprezentarea unor cifre zecimale nu

este unică, această proprietate fiind valabilă şi pentru alte coduri. Pentru codificare s-a ales reprezentarea 1011, deoarece codul pentru cifra 5 se poate obține atunci prin complementarea codului pentru cifra 4. Aceeaşi regulă se poate aplica pentru obținerea codului cifrei 6 din codul cifrei 3, a codului cifrei 7 din codul cifrei 2 etc.

Codurile care au această proprietate se numesc coduri *autocomplementare*. Un cod este autocomplementar dacă cuvântul de cod al complementului față de 9 al cifrei *N* (deci 9 – *N*) se poate obține din codul cifrei *N*, prin complementarea fiecăruia din cei 4 biți. De exemplu, codul 8421 nu este autocomplementar, iar codurile 2421, $64\bar{2}\bar{3}$, $84\bar{2}\bar{1}$ sunt autocomplementare.

Condiția necesară pentru ca un cod ponderat să fie autocomplementar este ca suma ponderilor să fie egală cu 9. Autocomplementaritatea constituie un avantaj în construcția unui dispozitiv aritmetic care lucrează cu numere zecimale reprezentate în codul respectiv.

Observație. În forma internă, numerele sunt reprezentate şi prelucrate fie sub formă binară, fie codificate printr-un cod binar-zecimal. Trebuie sesizată diferența dintre conversia unui număr zecimal în echivalentul său binar şi codificarea binar-zecimală a numărului zecimal. De exemplu:

$12_{10} = 1100_2$ (4 biți)
$12_{10} = (0001\ 0010)_{BCD}$ (8 biți)

Codurile neponderate pot avea un număr mai mare de 4 biți. Codurile cele mai uzuale sunt prezentate în Tabelul 2.2.

Tabelul 2.2. Coduri binar-zecimale neponderate.

Nr. zecimal	Exces 3	2 din 5	Gray
0	0011	00011	0000
1	0100	00101	0001
2	0101	00110	0011
3	0110	01001	0010
4	0111	01010	0110
5	1000	01100	0111
6	1001	10001	0101
7	1010	10010	0100
8	1011	10100	1100
9	1100	11000	1101

Codul *Exces 3* este autocomplementar, şi derivă din codul 8421 (BCD) prin adăugarea la fiecare cifră a valorii 3. Utilizând acest cod, se poate face distincție între lipsa unei informații înscrise într-un registru sau locație de memorie şi înscrierea valorii zero (0000 reprezintă lipsa unei informații, iar zero este codificat prin 0011).

Codul *2 din 5* se utilizează pentru reprezentarea numerelor zecimale printr-un grup de 5 biți. Denumirea derivă din faptul că fiecare cifră zecimală codificată în binar conține doi biți de 1 din cei 5 biți.

Codul *Gray* are proprietatea de adiacență, adică trecerea de la o cifră zecimală la următoarea sau precedenta necesită modificarea unui singur bit din cuvântul de cod. Este util pentru mărimile care cresc sau descresc succesiv.

2.3. Coduri detectoare de erori

Transmiterea informațiilor prin medii influențate de zgomote poate fi însoțită de introducerea unor erori. Verificarea transmiterii corecte a informațiilor se poate realiza cu ajutorul unor coduri speciale numite *coduri detectoare de erori*.

Una din metodele de detectare a unei erori o constituie *detectarea combinațiilor interzise*. În cazul codurilor binar-zecimale, deoarece se utilizează 10 din cele 16 combinații posibile de 4 biți, celelalte combinații nu trebuie să apară. Aceste combinații interzise se pot utiliza pentru detectarea erorii. Dacă, de exemplu, în codul BCD 1000 apare o singură eroare, codul poate fi transformat în 0000, 1100, 1010 sau 1001. Dintre aceste combinații, a doua şi a treia constituie combinații interzise, astfel încât în aceste cazuri eroarea poate fi detectată. Celelalte combinații nu sunt interzise, deci în cazurile respective eroarea nu poate fi detectată.

O modalitate de creştere a probabilității de detectare a erorilor constă în folosirea mai multor combinații interzise, care pot fi disponibile dacă codurile au mai mult de 4 biți. De exemplu, în codul *2 din 5* apare o eroare ori de câte ori o cifră codificată are mai mult sau mai puțin de doi biți de 1. Astfel, se pot detecta erori multiple.

O altă metodă pentru detectarea erorilor constă în folosirea unor biți suplimentari de verificare. De exemplu, un cod de n biți poate fi format din m biți de date şi r biți redundanți de verificare ($n = m + r$). Fiind date două cuvinte de cod, de exemplu 1000 1001 şi 1011 0001, se poate determina numărul biților care diferă. În acest caz, există 3 biți care diferă. Pentru determinarea numărului de biți care diferă, se efectuează suma modulo 2 între cele două cuvinte de cod, şi se calculează numărul biților de 1 ai rezultatului. Numărul pozițiilor în care două cuvinte de cod diferă reprezintă *distanța Hamming*. Dacă între două cuvinte de cod se află o distanță Hamming d, sunt necesare d erori de câte un singur bit pentru trecerea de la un cod la al doilea cod.

Proprietățile de detectare a erorilor ale unui cod depind de distanța sa Hamming. Pentru detectarea a d erori de câte un singur bit, este necesar un cod cu distanța d+1, deoarece cu un asemenea cod nu există posibilitatea ca d erori de un singur bit să modifice un cuvânt de cod valid într-un alt cuvânt de cod valid.

Un exemplu simplu de cod detector de erori este un cod care conține un bit suplimentar numit *bit de paritate*. Acest bit se poate alege astfel încât numărul total al biților având valoarea 1 în exprimarea numărului să fie par, respectiv impar. Dacă se utilizează *paritatea pară*, notând cu $x_3x_2x_1x_0$ biții cifrei zecimale şi cu p bitul de paritate, valoarea bitului de paritate determină ca suma modulo 2 a valorii tuturor biților să fie 0:

$$x_3 \oplus x_2 \oplus x_1 \oplus x_0 \oplus p = 0 \tag{2.8}$$

de unde rezultă:

$$p = x_3 \oplus x_2 \oplus x_1 \oplus x_0 \tag{2.9}$$

Dacă se alege *paritatea impară*, trebuie ca:

$$x_3 \oplus x_2 \oplus x_1 \oplus x_0 \oplus p = 1 \tag{2.10}$$

sau

$$p = x_3 \oplus x_2 \oplus x_1 \oplus x_0 \oplus 1 \tag{2.11}$$

Tabelul 2.3. Codul 8421 cu paritate pară.

Nr. zecimal	Cod
0	0 0000
1	1 0001
2	1 0010
3	0 0011
4	1 0100
5	0 0101
6	0 0110
7	1 0111
8	1 1000
9	1 1001

Un asemenea cod are distanța 2, deoarece o eroare de un singur bit produce un cuvânt de cod cu paritatea eronată. La transmisia datelor se adaugă bitul de paritate pară sau impară, iar la recepție se determină dacă paritatea este aceeaşi cu cea de la transmisie.

Codurile 8421 cu paritate pară ale cifrelor zecimale sunt indicate în Tabelul 2.3.

2.4. Coduri corectoare de erori

Verificarea parității nu poate detecta apariția erorilor duble, deoarece aceste erori nu modifică suma modulo 2 a biților. Există coduri mai complexe, numite coduri corectoare de erori, care permit şi corectarea unui bit eronat sau a mai multor biți eronați. Aceste coduri sunt utile atunci când retransmisia informației nu este posibilă, sau necesită un timp care nu ar fi acceptabil.

Presupunem un cod cu m biți de date şi r biți de verificare, cod care permite corectarea tuturor erorilor de un singur bit. Fiecăruia din cele 2^m cuvinte de cod valide îi corespund n cuvinte de cod ilegale cu distanța 1. Acestea se formează prin inversarea sistematică a fiecăruia din cei n biți. Fiecare din cele 2^m cuvinte valide necesită n+1 combinații de biți dedicate pentru cuvântul respectiv. Deoarece numărul total al combinațiilor de biți este 2^n, trebuie ca $(n+1)2^m \le 2^n$. Utilizând relația $n = m + r$, această cerință devine $(m + r + 1) \le 2^r$. Fiind dat m, aceasta impune o limită inferioară asupra numărului biților de verificare necesari pentru corectarea erorilor de un singur bit. De exemplu, pentru un cuvânt de $m = 4$ biți, numărul minim al biților de verificare este $r = 3$, iar pentru un cuvânt de $m = 8$ biți, numărul minim al biților de verificare este $r = 4$.

Aceste limite teoretice pot fi atinse utilizând o metodă datorată lui Richard Hamming. Metoda poate fi utilizată pentru construirea codurilor corectoare de erori pentru cuvinte de cod de orice dimensiune. Într-un cod Hamming, se adaugă r biți de paritate la un cuvânt de m biți, rezultând un nou cuvânt cu lungimea de $m + r$ biți. Biții sunt numerotați începând cu 1 (şi nu cu 0), bitul 1 fiind bitul c.m.s. Toți biții ai căror număr este o putere a lui 2 sunt biți de paritate, restul biților fiind utilizați pentru date. De exemplu, în cazul unui cuvânt de 4 biți, biții 1, 2 şi 4 sunt biți de paritate. În total, cuvântul de cod va conține 7 biți (4 de date şi 3 de paritate). În exemplul prezentat se va utiliza în mod arbitrar paritatea pară.

Fiecare bit de paritate verifică anumite poziții de biți. Aceste poziții sunt ilustrate în Figura 2.2, unde prin pătrate s-a indicat poziția biților de paritate, iar prin cercuri s-a indicat poziția biților de informație.

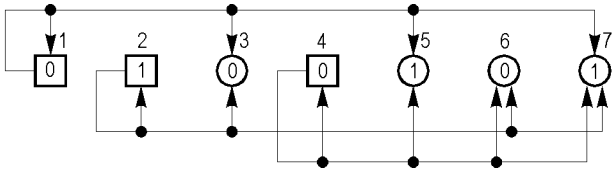


Figura 2.2. Construcția codului Hamming pentru valoarea binară 0101 prin adăugarea a trei biți de paritate.

Pozițiile de biți verificate de biții de paritate sunt următoarele:

Bitul 1 de paritate verifică biții 1, 3, 5, 7.
Bitul 2 de paritate verifică biții 2, 3, 6, 7.
Bitul 4 de paritate verifică biții 4, 5, 6, 7.

În general, bitul b este verificat de acei biți $b_1, b_2, \ldots, b_j$ astfel încât $b_1 + b_2 + \ldots + b_j = b$. De exemplu, bitul 5 este verificat de biții 1 şi 4 deoarece 1 + 4 = 5. Bitul 6 este verificat de biții 2 şi 4 deoarece 2 + 4 = 6.

Se consideră exprimarea în cod 8421 a cifrei zecimale 5 (0101). Bitul 1 de paritate va trebui să determine un număr par de cifre de 1 pentru pozițiile 1, 3, 5 şi 7, deci va avea valoarea 0. Similar se

determină valoarea bitului 2 de paritate, care va avea valoarea 1, şi a bitului 4 de paritate, care va avea valoarea 0.

Dacă transmiterea cifrei zecimale se realizează corect, toţi biţii de paritate verifică în mod corect paritatea. Dacă apare o eroare de transmisie, fie la un bit de paritate, fie la un bit de informaţie, acest cod poate corecta o singură eroare. De exemplu, presupunem că a apărut o eroare la transmiterea bitului de informaţie din poziţia 6. Codul recepţionat va fi 0100111 în loc de 0100101. Se verifică biţii de paritate, cu următoarele rezultate:

Bitul 1 de paritate este corect (biţii 1, 3, 5, 7 conţin doi biţi de 1)
Bitul 2 de paritate este incorect (biţii 2, 3, 6, 7 conţin trei biţi de 1)
Bitul 4 de paritate este incorect (biţii 4, 5, 6, 7 conţin trei biţi de 1)

Bitul incorect trebuie să fie unul din biţii testaţi de bitul 2 de paritate (2, 3, 6 sau 7). Deoarece bitul 4 de paritate este incorect, unul din biţii 4, 5, 6 sau 7 este incorect. Bitul eronat este unul din cei care se află în ambele liste, deci poate fi bitul 6 sau 7. Bitul 1 de paritate fiind corect, rezultă că şi bitul 7 este corect. Bitul eronat este deci bitul 6, şi valoarea acestui bit trebuie inversată. În acest fel, eroarea poate fi corectată.

O metodă simplă pentru determinarea bitului eronat este următoarea. Se calculează biţii de paritate, şi dacă toţi biţii sunt corecţi, înseamnă că nu există eroare (sau există mai mult de o eroare). Se atribuie apoi biţilor de paritate ponderile 1, 2, respectiv 4, şi se adună ponderile biţilor de paritate eronaţi. Suma rezultată reprezintă poziţia bitului eronat. De exemplu, dacă biţii de paritate 2 şi 4 sunt eronaţi, bitul eronat este bitul 6.

2.5. Coduri alfanumerice

Datele alfanumerice conţin cifre, litere şi semne speciale, numite, în general, *caractere*. Codurile care pot reprezenta asemenea caractere se numesc *coduri alfanumerice*.

2.5.1. ASCII

Un cod alfanumeric foarte des utilizat este codul ASCII (*American Standard Code for Information Interchange*), care codifică literele mari şi mici ale alfabetului englez, cifrele zecimale, semnele de punctuaţie şi alte caractere speciale. Codul ASCII utilizează 7 biţi pentru a codifica 128 de caractere. Din cele 128 de caractere, 94 sunt caractere care pot fi tipărite, iar 34 sunt caractere utilizate pentru diferite funcţii de control.

În Tabelul 2.4 se prezintă codurile ASCII. Cei 7 biţi ai codului sunt notaţi cu b_0 până la b_7, b_7 fiind bitul c.m.s. De notat că cei trei biţi mai semnificativi ai codului determină coloana din tabelă, iar cei patru biţi mai puţin semnificativi determină linia din tabelă. De exemplu, litera A este reprezentată în ASCII prin codul binar 100 0001 (coloana 100, linia 0001).

Tabelul 2.4. Codurile ASCII.

$b_3b_2b_1b_0$	$b_6b_5b_4$							
	000	001	010	011	100	101	110	111
0000	NULL	DLE		0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x

$b_3b_2b_1b_0$	$b_6b_5b_4$							
	000	001	010	011	100	101	110	111
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Cele 34 de caractere de control sunt desemnate în tabelul caracterelor ASCII prin nume abreviate. Caracterele de control se utilizează pentru transmisia datelor şi pentru aranjarea textului într-un anumit format. Există trei tipuri de caractere de control: de formatare, de separare a informaţiei şi de control al comunicaţiei. Dintre caracterele de formatare a textului menţionăm cele pentru deplasare înapoi BS (*Backspace*), retur de car CR (*Carriage Return*) şi tabulare orizontală HT (*Horizontal Tabulation*). Separatorii de informaţii se utilizează pentru separarea datelor în secţiuni, de exemplu în paragrafe şi pagini. Acestea cuprind caractere cum sunt separatorul de înregistrare RS (*Record Separator*) şi separatorul de fişier FS (*File Separator*). Caracterele de control al comunicaţiei se utilizează la transmisia textului. Exemple de asemenea caractere sunt STX (*Start of Text*) şi ETX (*End of Text*), care se pot utiliza pentru încadrarea unui mesaj transmis pe liniile de comunicaţie.

2.5.2. Unicode şi ISO/IEC 10646

Unicode este un set de caractere specificat de un consorţiu de producători importanţi de calculatoare din SUA, definit cu scopul de a elimina dificultăţile datorate utilizării seturilor de caractere codificate diferit în programele elaborate pentru mai multe limbi. Începând cu versiunea 1.1, *Unicode* este compatibil cu standardul 10646, a cărui primă versiune a fost elaborată în 1993 de organizaţiile ISO (*International Standards Organization*) şi IEC *International Electrotechnical Commission*). Numele standardului este *"Universal Multiple-Octet Coded Character Set"*, întâlnit şi sub forma acronimului UCS (*Universal Character Set*). Versiunea curentă a standardului 10646 este 3.0. UCS a fost primul set de caractere standardizat elaborat cu scopul de a include în final toate caracterele utilizate în toate limbile scrise din lume, ca şi alte simboluri, cum sunt cele matematice. UCS este utilizat atât pentru reprezentarea internă a datelor în sistemele de calcul, cât şi pentru comunicaţiile de date.

În versiunile uzuale ale UCS, codul unui caracter este reprezentat prin patru cifre hexazecimale; aceasta corespunde formei pe doi octeţi, denumită UCS-2. A fost definită de asemenea şi o formă pe 4 octeţi, UCS-4, pentru a garanta faptul că spaţiul de codificare va fi suficient şi în viitor. În forma pe 2 octeţi a UCS, cele 65536 coduri sunt împărţite în 256 de *linii* de câte 256 de *celule* fiecare. Primul octet al codului unui caracter indică numărul liniei, iar al doilea numărul celulei. Primele 128 de caractere din linia 0 reprezintă caracterele ASCII. Întreaga linie 0 conţine aceleaşi caractere ca şi cele definite de standardul ISO/IEC 8859-1. Octetul reprezentând codul unui caracter ASCII sau ISO/IEC 8859-1 poate fi transformat simplu în reprezentarea sa UCS prin adăugarea cifrelor 00 în faţa acestuia. UCS cuprinde aceleaşi caractere de control ca şi cele din setul ASCII.

În forma pe 4 octeţi, pot fi reprezentate peste 2 miliarde de caractere diferite. Primul bit al primului octet trebuie să fie 0, astfel încât se utilizează numai 31 de biţi din cei 32. Acest spaţiu de codificare este împărţit în 128 de *grupe*, fiecare conţinând 256 de *planuri*. Primul octet de cod indică numărul grupei, iar al doilea numărul planului. Al treilea şi al patrulea octet indică numărul liniei, respectiv al celulei. Caracterele care pot fi reprezentate în forma UCS-2 aparţin planului 0 din grupa 0, care este numit plan multilingv de bază sau BMP (*Basic Multilingual Plane*). Încă nu au fost alocate caractere poziţiilor în afara planului BMP, iar în practică se utilizează numai forma pe doi octeţi.

În forma UCS-2, caracterele cu coduri cuprinse între 00A0h şi 00FFh formează setul Latin-1. Aceste coduri conţin litere suplimentare utilizate în principalele limbi din Europa de Vest, ca şi semne matematice şi de punctuaţie. Codurile din setul Latin-1 se bazează pe standardul ISO/IEC 8859-1. Ta-

belul 2.5 prezintă setul Latin-1. În acest tabel, primele trei cifre hexazecimale ale codului sunt cele care desemnează coloana, iar ultima cifră a codului este cea care desemnează linia.

Tabelul 2.5. Setul de caractere Latin-1.

	00A	00B	00C	00D	00E	00F
0		°	À	Ð	à	ð
1	¡	±	Á	Ñ	á	ñ
2	¢	²	Â	Ò	â	ò
3	£	³	Ã	Ó	ã	ó
4	¤	´	Ä	Ô	ä	ô
5	¥	µ	Å	Õ	å	õ
6	¦	¶	Æ	Ö	æ	ö
7	§	·	Ç	×	ç	÷
8	¨	¸	È	Ø	è	ø
9	©	¹	É	Ù	é	ù
A	ª	º	Ê	Ú	ê	ú
B	«	»	Ë	Û	ë	û
C	¬	¼	Ì	Ü	ì	ü
D	-	½	Í	Ý	í	ý
E	®	¾	Î	Þ	î	þ
F	¯	¿	Ï	ß	ï	ÿ

Există patru zone principale pentru asignarea codurilor UCS-2. Zona A conține caractere alfabetice şi simboluri, având primele două cifre hexazecimale ale codului cuprinse între 00h şi 4Dh. De exemplu, această zonă cuprinde seturile Basic Latin, Latin-1, Latin Extended-A, Latin Extended-B; alfabetele grec, chirilic, armean, ebraic, arab, bengali, tamil, georgian; diferite simboluri, săgeți, operatori matematici, forme geometrice etc. Literele ă, Ă, ş, Ş, ţ, Ţ din limba română fac parte din setul Latin Extended-A, în timp ce literele â, Â, î, Î fac parte din setul Latin-1. De menționat că toate aceste caractere fac parte din setul de caractere ISO/IEC 8859-2, care cuprinde caracterele speciale necesare pentru limbile țărilor din Europa Centrală şi de Est. În Tabelul 2.6 se prezintă codurile ISO/IEC 10646 ale caracterelor speciale utilizate în limba română.

Tabelul 2.6. Codurile ISO/IEC 10646 ale caracterelor speciale din limba română.

Caracter	Cod hexa	Caracter	Cod hexa
Ă	01 02	ă	01 03
Â	00 C2	â	00 E2
Î	00 CE	î	00 EE
Ş	01 5E	ş	01 5F
Ţ	01 62	ţ	01 63

Zona I conține caractere ideografice, având primele două cifre hexazecimale ale codului cuprinse între 4Eh şi 9Fh. Un caracter ideografic reprezintă un cuvânt sau o unitate gramaticală inseparabilă. Există un mare număr de caractere ideografice în diferite limbi. Ca un principiu general de codificare, pentru ca un nou caracter să fie inclus în setul UCS, acest caracter trebuie să difere de toate caracterele deja incluse, atât în semnificație, cât şi ca formă. Formele grafice alternative ale caracterelor existente (variante de fonturi, hieroglife) nu au deci coduri distincte. În limbile chineză, japoneză şi coreeană există un mare număr de caractere ideografice care au aceeaşi origine istorică şi doar diferențe minore ca formă în cele trei limbi. Acestor variante naționale ale aceluiaşi caracter ideografic li s-a atribuit un cod UCS unificat, o soluție cunoscută sub numele de *unificare* CJK. Zona I conține asemenea caractere unificate.

Zona O, având primele două cifre hexazecimale ale codului cuprinse între A0h şi DFh, este rezervată pentru versiunile ulterioare ale standardului. Zona R este o zonă pentru utilizare restrânsă, codurile având primele două cifre hexazecimale cuprinse între E0h şi FFh . Această zonă este împărţită la rândul ei în zona de utilizare privată, zona de compatibilitate a caracterelor ideografice CJK şi zona caracterelor speciale. Zona de utilizare privată este disponibilă pentru utilizatorii care necesită caractere speciale pentru programele lor de aplicaţii. De exemple, icoanele utilizate în meniuri pot fi specificate prin coduri de caractere în această zonă. Există posibilitatea utilizării unui număr de 6400 de caractere private, cu coduri având primele două cifre hexazecimale cuprinse între E0h şi F8h. Această zonă nu va fi ocupată de versiunile viitoare ale standardului. Zona de compatibilitate conţine caracterele care sunt mapate la alte zone în spaţiul de codificare. Caracterele disponibile în această zonă specială au o utilizare largă, dar nu sunt direct compatibile cu modul de reprezentare a caracterelor UCS, astfel încât nu pot fi incluse direct în alte zone. Codurile FFFEh şi FFFFh nu reprezintă coduri de caractere, fiind excluse din UCS.

Există o problemă de ordonare a octeţilor din cadrul cuvintelor de 16 biţi reprezentând coduri de caractere în *Unicode*. În general, dacă octetul 0 se află în dreapta cuvântului (în partea mai puţin semnificativă), iar octetul 1 se află în stânga, ordonarea octeţilor este numită *"little-endian"*. Dacă octetul 0 se află în stânga cuvântului (în partea mai semnificativă), iar octetul 1 se află în dreapta, ordonarea octeţilor este numită *"big-endian"*. Presupunem că un şir de 2*n* octeţi reprezentând *n* caractere *Unicode* sunt transferate de la un calculator cu ordonarea *"little-endian"* la un calculator cu ordonarea *"big-endian"*. Ordinea octeţilor din cuvintele de 16 biţi va fi atunci schimbată. Dacă la începutul şirului original se include codul FEFFh, care este un cod pentru marcajul ordinii octeţilor (*Byte Order Mark*), atunci prin inversarea octeţilor acesta va apare sub forma FFFEh, care este un cod invalid. Acest cod invalid indică faptul că ordinea octeţilor din cadrul tuturor cuvintelor este incorectă, şi deci ordinea trebuie inversată înaintea interpretării cuvintelor ca şi caractere. Astfel, prin plasarea codului FEFFh la începutul unui şir de caractere, o aplicaţie poate determina dacă octeţii trebuie inversaţi înainte de interpretarea codurilor. O situaţie similară apare la transferul de la un calculator cu ordonarea *"big-endian"* la un calculator cu ordonarea *"little-endian"*.

3. Desfăşurarea lucrării

3.1. Reprezentaţi în formatul IEEE 754 cu precizie simplă următoarele numere zecimale:

a) 1; b) -1; c) 5; d) -5; e) 35,4; f) -35,4; g) 2,6; h) -192

Scrieţi rezultatele în hexazecimal.

3.2. Reprezentaţi în formatul IEEE 754 cu precizie dublă următoarele numere zecimale:

a) 1; b) 1,5; c) 2,5; d) 5; e) 35,4; f) -35,4

3.3. Scrieţi numerele zecimale corespunzătoare următoarelor reprezentări în formatul IEEE 754 cu precizie simplă:

a) 41 8A 1E 94; b) 41 36 66 6A; c) BE CC CC CB; d) BD AB 40 C0

3.4. Deduceţi algoritmii pentru adunarea şi scăderea a două cifre zecimale exprimate în codul 8421. Pentru deducerea algoritmului de adunare se va întocmi un tabel cu toate sumele posibile care se pot obţine prin adunarea a două cifre zecimale şi a unui eventual transport, indicând pentru fiecare din acestea suma corectă. Similar se va proceda pentru scădere, ţinând cont şi de un eventual împrumut.

3.5. Deduceţi algoritmii pentru adunarea şi scăderea a două cifre zecimale exprimate în codul Exces 3.

3.6. Deduceţi algoritmii pentru conversia din binar în codul Gray şi din codul Gray în binar.

3.7. Construiţi codul Hamming pentru un cuvânt de 16 biţi, considerând paritatea impară. Explicaţi modul în care funcţionează corecţia unei erori.

3.8. Deduceți regula de conversie a literelor mari codificate în ASCII în litere mici, şi a literelor mici în litere mari.

3.9. Determinați cuvântul reprezentat de următorul şir *Unicode* specificat în hexazecimal:

`FFFE`, `4800`, `6900`, `7200`, `6100`, `6700`, `6100`, `6E00`, `6100`

3.10. Decodificați următorul şir *Unicode* specificat în hexazecimal:

`FEFF`, `0031`, `00F7`, `0034`, `003D`, `00BC`, `0020`, `0065`, `0073`, `0074`, `0065`,
`0020`, `006F`, `0020`, `0066`, `0072`, `0061`, `0063`, `0163`, `0069`, `0065`, `002E`