## Redundanţa Informaţională a Datelor

Erorile în datele din sistemele de calcul pot apare atunci când datele sunt transferate de la o unitate la alta, ori chiar de la un sistem la altul, sau atunci când datele sunt stocate în unităţile de memorie.
Tolerarea acestor erori se poate realiza prin introducerea unor redundanţe în datele respective. Aceasta este redundanţa informaţională. Forma aceasta, cea mai răspândită, de redundanţă este codificarea.
Codificarea adaugă, datelor, biţi de verificare care permit validarea corectitudinii respectivelor date utilizate şi, în anumite situaţii, chiar corectarea biţilor de date eronaţi.

### 1. Codificarea

Codificarea este un domeniu de cercetare binecunoscut cu o practică reputată, mai ales, în domeniul comunicaţiilor.
Atunci când se aplică codificarea un cuvânt de informaţie cu *d* biţi este codificat într-un *cuvânt de cod* cu *c* biţi. Cuvântul de cod este, întotdeauna, mai lung decât cuvântul de informaţie, $c > d$. Codificarea introduce informaţie redundantă.
Aceasta înseamnă că se introduce informaţie care nu este necesară, în mod evident, pentru procesarea informaţiei. O consecinţă a informaţiei redundante este faptul că *nu toate* cele $2^c$ combinaţii de cod sunt cuvinte valide de cod.

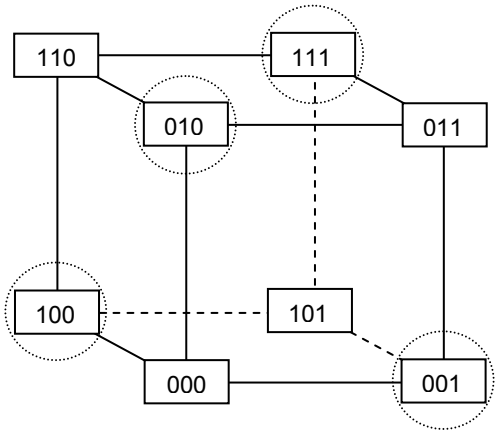



Figura 1. Distanţele Hamming în spaţiul cuvintelor cu 3 ranguri binare.

Din acest motiv, se poate întâmpla ca atunci când se *decodifică* un cuvânt de cod având *c* biţi, pentru regăsirea unui cuvânt original (iniţial) cu *d* biţi să se dovedească că se operează cu un cuvânt incorect. Acest fapt indică incidenţa unei erori asupra cuvântului de cod cu *c* biţi. Pentru unele dintre tipurile de scheme de codificare anumite erori fi chiar şi corectate, nu numai detectate.
Un cod este definit ca fiind mulţimea tuturor cuvintelor de cod. Parametrii cheie ai performanţei unui cod sunt *numărul de biţi eronaţi* care pot fi detectaţi dar şi *numărul de erori care pot fi corectate*. Efortul impus de un cod anume este evaluat atât prin numărul suplimentar de biţi cât şi prin durata codificării şi decodificării unui cuvânt de cod.
O metrică importantă în spaţiul cuvintelor de cod este *distanţa Hamming*. Distanţa Hamming dintre două cuvinte de cod este numărul de ranguri binare prin care cele

două cuvinte diferă. În figura 1 sunt prezentate cele opt cuvinte ale codului cu trei ranguri binare. Două cuvinte, în figura 1, sunt conectate printr-o muchie dacă acestea diferă printr-un singur rang, adică au distanţa Hamming egală cu 1. Cuvintele 101 şi 011 diferă prin două ranguri şi, din aceste motiv, au distanţa Hamming 2. Pentru a ajunge din nodul 101 în nodul 011 sunt parcurse minimum două muchii.
Se presupune că două cuvinte de cod valide diferă doar prin rangul binar cel mai puţin semnificativ, aşa cum sunt codurile 101 şi 100. În această situaţie o singură eroare care afectează bitul cel mai puţin semnificativ în oricare dintre cele două cuvinte ale codului va trece neobservată deoarece cuvântul eronat este de asemenea un cuvânt de cod valid, aparţinând codului. Dar dacă distanţa Hamming dintre două sau mai multe cuvinte de cod are valoarea doi sau mai mare decât doi, atunci acest fapt garantează că o eroare afectând un singur rang din două cuvinte de cod arbitrare nu va transforma un cuvânt de cod într-un alt cuvânt de cod.

Distanţa unui cod este distanţa Hamming minimă dintre oricare două cuvinte de cod valide. Codul care constă din patru cuvinte de cod {001, 010, 100, 111}, marcate prin încercuire în figura 1, are distanţa 2 şi este, astfel, capabil să detecteze orice eroare care afectează un singur bit. Iar codul care constă din cuvintele de cod {000, 111} are distanţa trei şi are, din acest motiv, capacitatea să detecteze orice eroare singulară ori dublă. Dar, dacă erorile duble sunt puţin probabile să apară, atunci acest cod poate fi utilizat să *corecteze* orice eroare simplă, eroare asupra unui singur rang.
Pentru detecţia a până la $k$ ranguri eronate, în general, distanţa de cod trebuie să fie cel puţin $k + 1$, iar pentru corecţia a până la $k$ erori distanţa de cod trebuie să fie minimum $2k + 1$.

O altă proprietate importantă a codurilor este *separabilitatea*. Un cod separabil are câmpuri separate de date şi de verificare. Pentru astfel de coduri procedeul de decodare este foarte simplu. Procedeul constă din separarea rangurilor de date, de rangurile dedicate verificării. Rangurile de verificare sunt procesate separat în vederea corectitudinii datelor.
Un cod neseparabil, pe de-altă parte , are rangurile de date şi de verificare integrate laolaltă iar extragerea datelor din cuvintele de cod necesită o oarecare procesare care poate aduce cu sine o întârziere.

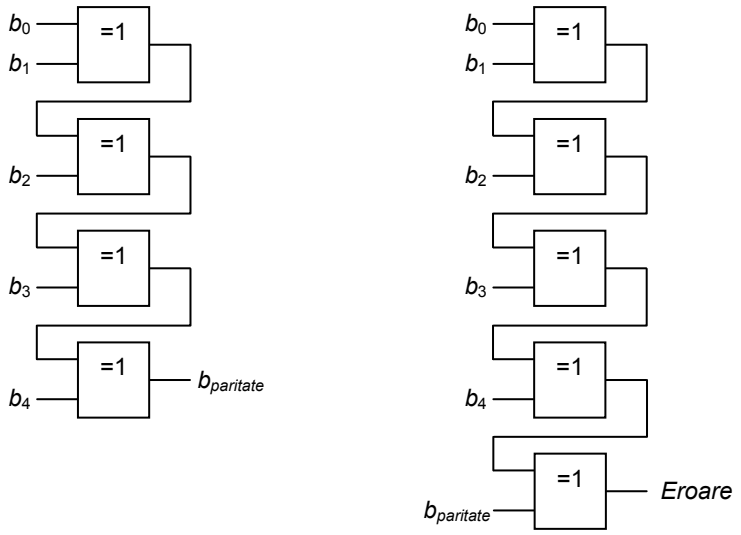
#### 2. Codurile de paritate

Cele mai simple coduri sunt, probabil, cele de paritate. În forma sa fundamentală un cod de paritate cuprinde $d$ ranguri de date şi un rang, suplimentar, care reprezintă paritatea. Într-un cod de paritate impară (pară), rangul suplimentar este stabilit astfel încât suma tuturor valorilor nenule din cele $d + 1$ ranguri să fie impară (pară). Raportul suplimentării codului de paritate este $1/d$.

Un cod de paritate are o distanţă Hamming egală cu 2 şi este garantat să detecteze toate erorile asupra unor singure ranguri. Dacă un rang trece din zero în unu, sau invers, paritatea globală a cuvântului de cod s-a schimbat iar eroarea este detectată. Codurile de paritate nu pot, totuşi, să corecteze orice eroare din cuvântul respectiv.

Deoarece codurile de paritate sunt coduri separabile, este imediată proiectarea codificării parităţii respectiv a decodificării acesteia. În figura 2 sunt arătate circuitele de codificare şi respectiv de decodificare pentru un cuvânt de date cu 5 ranguri.

Codificatorul este alcătuit dintr-un sumator modulo-2 cu cinci linii de intrare care generează o valoare zero dacă numărul de ranguri cu valoarea 1 este par. Decodificatorul generează paritatea din biţii de date recepţionaţi şi compară paritatea astfel generată cu cea sosită în bitul de paritate.
Dacă acestea coincid linia de ieşire a circuitului *SAU-EX* (XOR) este zero indicând că nu s-a detectat nici o eroare.
În cazul în care paritatea generată şi paritatea recepţionată nu coincid, linia de ieşire are valoarea 1 indicând prezenţa unei erori.
Este de reţinut că erori care afectează două ranguri ale cuvântului recepţionat nu pot fi detectate prin verificarea parităţii. Un număr impar de ranguri eronate va fi, totuşi, detectat.




(a) Codificatorul (b) Decodificatorul

Figura 2. Circuitele de calcul pentru bitul par de paritate.

Alegerea unei parităţi pare sau impare a bitului de paritate se determină în raport cu erorile uni-direcţionale ale tuturor rangurilor. Altfel spus, care dintre erorile:

- toate rangurile se modifică în 0, respectiv
- toate rangurile se modifică în 1,

este mai probabilă.

Dacă, spre exemplu, se alege codul de paritate pară, bitul de paritate generat pentru toate rangurile de date zero, va fi zero. Într-un astfel de caz, o eroare de felul *toate rangurile trecute în zero* va fi nedetectată deoarece cuvântul eronat este un cuvânt valid al codificării. Prin alegerea parităţii impare a rangului de paritate, acesta va detecta defectul care face ca toate rangurile cuvântului să fie 0.

Pe de-altă parte, dacă defectul *tot cuvântul are biţii 1* este mai probabil decât defectul atunci *toate rangurile trecute în zero*, trebuie să se facă astfel încât să nu fie cuvânt valid acela care are toate rangurile 1. În acest context trebuie ales modul impar de calcul a bitului de paritate.

Au fost propuse şi implementate mai multe varietăţi ale rangului de paritate. Una dintre acestea este cea care atribuie câte un bit de paritate la fiecare octet (*byte*). Astfel, în loc să fie determinat un bit de paritate pentru un cuvânt (de 16 biţi, 32 biţi

ori chiar mai mare) se atribuie câte un bit de paritate fiecărui octet. Acest mod face să crească raportul dintre biţii de paritate şi cei de informaţie de la 1/*d* la *n*/d, unde prin *n* s-a notat numărul de octeţi din cuvântul de date considerat.
Avantajul metodei este că pot fi detectate până la *m* defecte atâta vreme cât acestea apar în octeţi diferiţi. Dacă defectele *toţi biţii zero* şi respectiv *toţi biţii unu* sunt la fel de probabile atunci se poate alege ca biţii de paritate să fie calculaţi alternativ după paritate pară şi respectiv paritate impară.
O alternativă a acestei metode este un bit de paritate *întreţesut*. Astfel, dacă lungimea *d* a cuvântului de date este 64 atunci aceştia se vor nota respectiv prin $a_{63}$, $a_{62}$, … , $a_0$.
Se vor utiliza opt biţi de paritate astfel încât primul va fi bitul de paritate al biţilor aleşi astfel: $a_{63}$, $a_{47}$, $a_{39}$, $a_{31}$, $a_{23}$, $a_{15}$ şi $a_7$, cei mai semnificativi biţi din fiecare dintre cei opt octeţi ai cuvântului.
Similar, ceilalţi şapte biţi de pariate vor fi atribuiţi corespunzător grupurilor de biţi întreţesuţi.
O astfel de schemă de paritate este benefică atunci când au loc scurtcircuite între biţii adiacenţi ai cuvântului de date protejat (utilizarea magistralelor, este un bun exemplu în acest sens). Suplimentar biţii de paritate pot fi alternaţi (calculaţi fie pentru paritate pară, sau fie pentru paritate impară) între grupurile de biţi respectivi cu avantajul că erorile unidirecţionale (toţi biţii 1 sau 0) pot fi de asemenea detectate.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | **1** |
| 1 | 0 | 1 | 0 | 1 | 1 | **0** |
| 1 | 1 | 0 | 0 | 0 | 0 | **0** |
| 0 | 0 | 0 | 1 | 1 | 1 | **1** |
| 1 | 1 | 1 | 1 | 1 | 1 | **0** |
| **1** | **0** | **0** | **1** | **0** | **0** | **0** |

Figura 3. Exemplul unei parităţi de acoperire.

O altă extensie a conceptului parităţii poate opera, de asemenea, chiar şi corecţii ale erorilor. Cea mai simplă astfel de metodă implică organizarea datelor într-o matrice bi-dimensională aşa cum se poate vedea în figura 3. Biţii de paritate sunt cei reprezentaţi prin caractere îngroşate. Bitul de la sfârşitul unei linii reprezintă bitul de paritate al respectivei linii. Analog, bitul situat la baza unei coloane reprezintă bitul de paritate al acelei coloane. Schema de paritate pară utilizată este aceeaşi atât pentru linii cât şi pentru coloane. Un singur bit eronat, oriunde în matrice, va conduce la identificarea unei linii şi a unei coloane eronate. Deoarece o linie şi o coloană au intersecţie unică în matrice, bitul eronat va fi identificat şi corectat.

Tehnica prezentată anterior este un exemplu de *paritate suprapusă*, în care fiecare bit este *acoperit* prin mai mult decât un singur bit de paritate. Această manieră se poate generaliza printr-o metodă mai cuprinzătoare de paritate suprapusă. Scopul generalizării este să se ofere capacitatea identificării oricărui bit singular eronat.
Se presupune că sunt în total *d* biţi de date în total. Scopul generalizării constă în găsirea numărului de biţi de paritate, pe de-o parte şi care biţi ar trebui să fie acoperiţi prin fiecare bit de paritate, pe de-altă parte.

Dacă se notează prin *r* (redundanţi) numărul de biţi de paritate (de verificare) adăugaţi celor *d* biţi de date, atunci sunt în total, în fiecare cuvânt codat $d + r$ biţi. Rezultă că sunt în total $d + r$ stări de eroare, unde în starea *i* bitul cu rangul *i* al

cuvântului codat este eronat. Suplimentar mai există o stare în care nu există nici un bit eronat, aşa că sunt $d + r + 1$ stări care vor fi distinse, în total.
Trebuie reţinută ideea că se abordează cazul erorilor singulare, această schemă nefiind proiectată să detecteze toate erorile care vizează doi biţi.

Defectele vor fi detectate prin implementarea a $r$ verificări de paritate. Astfel, pentru fiecare bit de paritate se verifică dacă paritatea generală a respectivului bit în raport cu biţii acoperiţi este corectă. Aceste $r$ verificări de paritate pot să genereze până la $2^r$ rezultate distincte. Cu aceasta se poate concluziona că numărul minim de biţi de verificare $r$ satisface relaţia:

$$2^r \geq d + r + 1 \qquad (1)$$

Modul de asociere al biţilor de date la biţii de verificare care-i acoperă urmează să fie stabilit. Se asociază fiecăruia dintre cele $d + r + 1$ stări unul dintre cele $2^r$ coduri posibile ale biţilor de paritate. Exemplul următor ilustrează procedeul.

*Tabelul 1.*
Un exemplu de atribuire al valorilor de paritate la stări.

| Starea | Erorile de paritate | Sindromul |
|---|---|---|
| Fără erori | Niciuna | 000 |
| Bitul 0 ($p_0$) eroare | $p_0$ | 001 |
| Bitul 1 ($p_1$) eroare | $p_1$ | 010 |
| Bitul 2 ($p_2$) eroare | $p_2$ | 100 |
| Bitul 3 ($a_0$) eroare | $p_0$, $p_1$ | 011 |
| Bitul 4 ($a_1$) eroare | $p_0$, $p_2$ | 101 |
| Bitul 5 ($a_2$) eroare | $p_1$, $p_2$ | 111 |
| Bitul 6 ($a_3$) eroare | $p_0$, $p_1$, $p_2$ | 011 |

*Exemplul 1.* Se presupun patru biţi de date, notaţi respectiv prin $a_3a_2a_1a_0$. Din relaţia (1) se determină imediat că $r = 3$, numărul minim de biţi de paritate, notaţi în cele ce urmează prin $p_2p_1p_0$. Există în total 8 (4 + 3 + 1) stări pe care le poate lua cuvântul de cod. Cuvântul de cod complet arătă astfel:

$$a_3a_2a_1a_0p_2p_1p_0.$$

Aşa cum se poate remarca ultimii trei biţi din cuvântul de cod, cei aflaţi în rangurile puţin semnificative, sunt biţii de paritate, în timp ce ceilalţi biţi sunt biţii de date.

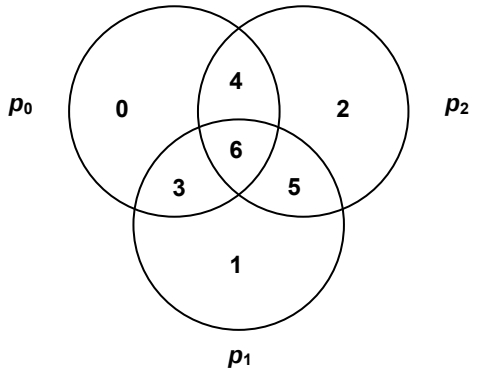



Figura 4. Atribuirea biţilor de paritate din Tabelul 1.

Tabelul 1 prezintă o posibilă atribuire a rezultatelor verificărilor de paritate în raport cu stările, care este ilustrată, de asemenea, în figura 4. Raţiunea existenţei unei stări „*fără erori*" este evidentă, ca şi atribuirea următoarelor trei stări în care un singur bit de paritate este eronat. Atribuirea ultimelor patru stări, corespunzătoare unor erori în biţii de date la cele patru coduri combinate ale biţilor de paritate poate fi operată în 4! moduri distincte. Unul dintre modurile posibile este prezentat în tabelul 1 şi în figura 4.

Dacă, spre exemplu, doi biţi de paritate $p_0$ şi $p_2$ sunt eronaţi (şi doar aceştia) aceasta arată că apare o problemă cu bitul din poziţia 4, mai exact cu bitul $a_1$.

Un bit de paritate va acoperi toate rangurile biţilor a căror eroare este indicată prin verificarea de paritate corespunzătoare.

Astfel, bitul de paritate $p_0$ acoperă rangurile 0, 3 şi 4 (aşa cum se poate urmări în figura 4). Această proprietate corespunde, matematic, expresiei:

$$p_0 = a_0 \oplus a_2 \oplus a_3.$$

În mod similar se pot scrie relaţiile omoloage:

$$p_1 = a_0 \oplus a_2 \oplus a_3,$$
$$p_2 = a_1 \oplus a_2 \oplus a_3.$$

Pentru biţii de date $a_3a_2a_1a_0 = 1100$, biţii de paritate sunt $p_2p_1p_0 = 001$.

Se presupune, acum, că pentru cuvântul de cod 1100001 are loc o eroare care afectează un singur bit şi acest cuvânt devine 1000001. Se recalculează biţii de paritate şi se obţine codul $p_2p_1p_0 = 111$. Prin calculul diferenţei (prin aplicarea bit cu bit a operatorului *SAU-EX*) dintre noile valori ale biţilor de paritate şi cele anterioare rezultă 110. Această diferenţă, numită sindrom, arată care bit dintre biţii cuvântului de cod este eronat. Sindromul 110 arată, aşa cum se poate remarca din tabelul 1, că bitul $a_2$ este eronat iar datele corecte arată astfel: $a_3a_2a_1a_0 = 1100$.

Acest cod se numeşte cod Hamming (7,4) corector al unei singure erori.

Sindromul (rezultat verificării parităţii) poate fi calculat direct, într-un singur pas, din biţii $a_3a_2a_1a_0p_2p_1p_0$. Modul de calcul este cel mai bine ilustrat prin operaţii matriceale în care toate sumele sunt calculate modulo-2. Matricea de mai jos se numeşte matricea verificării parităţii:

$$\begin{bmatrix} 1110100 \\ 1101010 \\ 1011001 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix} = \begin{bmatrix} s_2 s_1 s_0 \end{bmatrix}$$

□

Dacă $2^r > d + r + 1$, atunci este necesar ca $d + r + 1$ din cele $2^r$ să servească drept sindromuri. În astfel de situaţii este mai bine să se evite acele combinaţii care cuprind un număr mare de unităţi.

Codul din tabelul 1 are capacitatea să corecteze o eroare care afectează un singur bit dar nu poate detecta o eroare care afectează doi biţi (o eroare dublă). Astfel dacă apar două erori în cuvântul de cod 11001, cauzând cuvântul de cod eronat 1010001 (rangurile $a_2$ şi $a_1$ sunt afectate de aceste erori) sindromul rezultat este 011 indicând eronat că eroarea este în bitul $a_0$. O cale de creştere a capacităţii detecţiei erorilor este suplimentarea numărului de ranguri care servesc ca ranguri de paritate atât pentru rangurile de date cât şi pentru rangurile de verificare.
Codul rezultat este un cod Hamming (8,4) capabil să detecteze două erori şi să corecteze o singură eroare. Generarea sindroamelor pentru acest cod este prezentată în continuare.

$$\begin{bmatrix} 11111111 \\ 11100100 \\ 11010010 \\ 10110001 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \\ p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix} = \begin{bmatrix} s_3 s_2 s_1 s_0 \end{bmatrix}$$

Ca şi mai înainte ultimii trei biţi ai sindromului arată bitul eronat care poate fi corectat, atât timp cât primul bit $s_3$ are valoarea 1. Deoarece bitul $p_3$ este bitul de paritate al tuturor celorlalţi biţi ( atât biţi de paritate cât şi de date), o singură eroare modifică paritatea globală şi ca rezultat, bitul $s_3$ trebuie să fie egal cu 1. Dacă bitul $s_3$ este zero şi oricare alt bit de sindrom este 1, atunci s-a detectat o eroare dublă sau mai mare.
Dacă o eroare afectează cuvântul de cod 11001001 şi produce cuvântul eronat 10001001, sindromul calculat este 1110, arătând, ca mai înainte, că bitul $a_2$ este eronat.
Dar dacă, apar două erori rezultând cuvântul de cod eronat 10101001, sindromul determinat este 0011, indicând că a avut loc o eroare ne-corectabilă.
Un număr par de erori este detectabil, în timp ce un număr impar de erori, dar mai mare decât 1, nu este distinctibil de o eroare ce afectează un singur bit şi implică o corecţie eronată.

În mod curent circuitele de memorii, care au suport pentru corecţia unei singure erori şi detecţia a două erori, utilizează coduri Hamming de forma (39,7) sau (72,8).

Deoarece apariţia unor erori între două sau mai multe celule de memorie adiacente este mai probabilă, biţii unui singur cuvânt de memorie sunt adesea plasaţi în celule de memorie care nu sunt învecinate, reducându-se astfel probabilitatea apariţiei unei erori duble în cadrul unui cuvânt.

Un dezavantaj al codurilor Hamming considerate, capabile să detecteze două erori şi să corecteze o singură eroare, constă în faptul că acestea pot avea viteză mică de utilizare. Calculul biţilor adiţionali de verificare, care sunt paritatea altor biţi de verificare şi biţi de date, pot introduce penalizări considerabile prin procesele de codificare şi decodificarea induse.
O cale de evitare a acestor penalizări dar păstrând capacitatea detecţiei erorilor duble constă în atribuirea, atât pentru biţii de date cât pentru şi de verificare, a unor sindroame alcătuite dintr-un număr impar de unităţi. De remarcat faptul că pentru codul Hamming iniţial, corector de o singură eroare, sindroamele biţilor de verificare cuprind câte o singură unitate.

**2. Sumele de verificare**

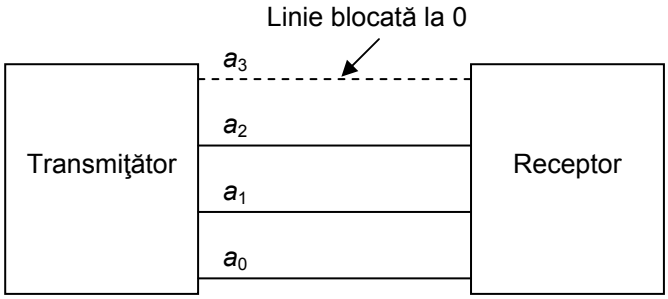
Sumele de verificare, sau sumele de control, sunt utilizate în primul rând pentru detectarea erorilor în transmisiile de date prin canalele de comunicaţie. Ideea fundamentală este sumarea informaţiei din blocul care urmează să fie transmis şi transmiterea acestei sume împreună cu blocul de date. Receptorul transmisiei calculează suma datelor receptate şi compară suma determinată, cu suma transmisă. Dacă în urma comparaţiei celor două sume acestea nu coincid, atunci se poate afirma că s-a determinat o eroare de transmisie.

| | | | |
|---|---|---|---|
| 0000 | 0000 | 0000 | |
| 0101 | 0101 | 0101 | |
| 1111 | 1111 | 1111 | 00000101 |
| 0010 | 0010 | 0010 | 11110010 |
| **0110** | **00010110** | **0111** | **11110111** |
| (a) | (b) | (c) | (d) |

Figura 5. Varietăţi de calcul ale sumelor de verificare
(cu litere îngroşate sunt prezentate sumele de verificare rezultate).
(a) Simplă precizie. (b) Dublă precizie. (c) Reziduu. (d) Honeywell.

Există câteva varietăţi de sume de verificare. Se notează prin *d* numărul de cuvinte de date transmise. În versiunea *simplă-precizie*, suma de verificare este calculată prin însumare *modulo-*$2^d$. În versiunea *dublă-precizie*, suma de verificare este calculată prin însumare *modulo-*$2^{2d}$.
Figura 5 prezintă un exemplu pentru fiecare varietate de calcul al sumei de verificare. Suma de verificare în simplă-precizie descoperă mai puţine erori decât suma de verificare calculată în versiunea dublă-precizie, deoarece determină doar ultimii *d* biţi din dreapta (rangurile cele mai puţin semnificative). Procedeul care determină suma de control *Reziduu* ia în consideraţie transportul din rangul cel mai semnificativ pe care-l întoarce şi-l adună la rangul cel mai puţin semnificativ.
În calculul sumei de verificare *Honeywell* se concatenează cuvintele în perechi succesive şi apoi se determină suma de control în dublă-precizie (*modulo-*$2^{2d}$). Acest mod de determinare al sumelor de control ţine seama de erorile care pot apărea în aceleaşi ranguri binare.
Se consideră, spre exemplu, cazul înfăţişat în figura 6. Se remarcă faptul că linia care transmite rangul $a_3$ al fiecărui cuvânt este blocată la valoarea logică 0. Din cauza acestui defect al liniei de transmisie, receptorul va stabili că suma de control trimisă şi

cea determinată în urma transmisiei se potrivesc în cazul sumei de verificare determinate în simplă-precizie.




(a)

| | | | |
|---|---|---|---|
| 1000 | 0000 | | |
| 1011 | 0011 | 10001011 | 00000011 |
| 0000 | 0000 | 00001100 | 00000100 |
| 1100 | 0100 | | |
| **1111** | **0111** | **10010111** | **00010111** |
| Transmis | Recepţionat | Transmis | Recepţionat |

(b) (c)

Figura 6. Comparaţie între sumele de control în simplă-precizie şi *Honeywell*.
(cu litere îngroşate sunt prezentate sumele de verificare corespunzătoare).
(a) Circuitul. (b) Simplă precizie. (c) Honeywell.

Pe de-altă parte suma de verificare *Honeywell* atunci când este calculată în raport cu datele recepţionate va fi diferită în comparaţie cu suma recepţionată. Aceasta va face să se producă semnalul de eroare. Toate sumele de verificare permit detecţia erorii dar nu pot determina localizarea erorii. Din acest motiv întregul bloc de date trebuie retransmis, dacă s-a detectat o eroare în blocul de date recpţionat.

#### 3. Codurile *M*-din-*N*

Un cod *M-din-N* ($M < N$) este un excelent exemplu pentru categoria de coduri detectoare de erori *unidirecţionale.* Erorile unidirecţionale sunt de aşa natură încât toţi biţii îşi schimbă starea în acelaşi sens, fie din 1 în 0, fie din 0 în 1, dar niciodată în ambele sensuri.

Într-un cod *M-din-N* fiecare cuvânt constând din *N* biţi are exact *M* biţi poziţionaţi în valoarea 1, rezultând existenţa a $C_N^M$ cuvinte de cod distincte.

Orice modificare a unui singur bit va conduce la modificarea numărului de unităţi din cuvânt ajungându-se fie la $M + 1$ ori la $M - 1$ biţi poziţionaţi în 1. Erorile unidirecţionale multiple pot fi şi acestea detectate, eventual.

Un exemplu foarte simplu şi clar de cod *M-din-N* este codul 2-din-5 care constă din 10 cuvinte de cod şi poate servi pentru codificarea cifrelor zecimale. Un exemplu de cod 2-din-5 este prezentat în tabelul 2.

Există 10! moduri diferite de atribuire a celor 10 cuvinte de cod cifrelor zecimale. Atribuirea arătată în tabelul 2 corespunde ordinii naturale binare. Avantajul fundamental al codurilor *M*-din-*N* rezidă în simplitatea conceptuală a acestora. Dar, pe

de-altă parte, codificarea şi decodificarea acestora este devine relativ mai complexă, în general, deoarece aceste coduri nu sunt separabile, aşa cum se-ntâmplă în cazurile codurilor de paritate şi celor cu sume de verificare.

*Tabelul 2*

Codul 2-din-5 pentru cifrele zecimale

| Cifra | Cuvântul de cod |
|---|---|
| 0 | 00011 |
| 1 | 00101 |
| 2 | 00110 |
| 3 | 01001 |
| 4 | 01010 |
| 5 | 01100 |
| 6 | 10001 |
| 7 | 10010 |
| 8 | 10100 |
| 9 | 11000 |

Se pot, la rigoare, genera coduri *M*-din-*N* separabile. Dacă sunt considerate, spre exemplu, codurile *M*-din-2*M* atunci se adaugă *M* biţi de verificare la deja existenţii *M* biţi utilizaţi pentru codificare astfel încât cuvântul de cod cu lungimea 2*M* biţi să conţină exact *M* unităţi. Astfel de coduri sunt uşor de codificat şi decodificat dat au o supra-lungime mare (100% sau mai mult) comparativ cu omoloagele lor neseparabile. Astfel, pentru codificarea celor zece cifre zecimale ajungem să se genereze un cod 4-din-8 având un nivel de redundanţă superior celui exemplificat în tabelul 2, codul 2-din-5.

#### 4. Codurile ciclice

În codurile ciclice codificarea datelor constă în multiplicarea, modulo-2, a cuvântului de date printr-un factor constant iar produsul calculat reprezintă cuvântul codificat. Decodificarea se face prin divizarea cu aceeaşi constantă. Dacă restul este nenul, aceasta arată apariţia unei erori. Codurile se numesc ciclice deoarece pentru orice cuvânt de cod $a_{n-1}, a_{n-2}, \ldots, a_0$, deplasarea circulară, ciclică, a acestuia $a_0, a_{n-1}, a_{n-2}, \ldots, a_1$, este de asemenea un cuvânt de cod. Codul cu 5 biţi constând din secvenţele binare: {00000, 00011, 00110, 011000, 11000, 10001, 00101, 01010, 10100, 01001, 10010, 01111, 11110, 11101, 11011, 10111}, este un cod ciclic. Codurile ciclice au fost ţinta unui efort susţinut de cercetare şi sunt larg utilizate în memorarea datelor dar şi în comunicaţii. Teoria codurilor ciclice se bazează pe elemente de algebra discreta. Astfel, se presupune că sunt utilizaţi *k* biţi pentru datele care se doresc codificate. Cuvântul codificat având lungimea de *n* biţi se obţine multiplicând cei *k* biţi de date printr-un factor care are *n*-*k*+1 biţi lungime.
În teoria codificării ciclice multiplicatorul este reprezentat printr-un polinom, numit *polinom generator*. Unităţile şi zerourile din acest polinom sunt considerate ca fiind coeficienţii unui polinom de gradul *n*-*k*. Astfel, dacă multiplicatorul cu 5 ranguri binare este de forma 11001, atunci generatorul polinomial este:

$$G(X) = 1 \cdot X^4 + 1 \cdot X^3 + 0 \cdot X^2 + 0 \cdot X^1 + 1 \cdot X^0 = X^4 + X^3 + 1.$$

Un cod ciclic utilizând un generator polinomial de gradul *n*-*k* şi în total *n* biţi codificaţi se numeşte un cod ciclic (*n, k*).

Aceste coduri sunt mult utilizate în aplicaţii cum sunt comunicaţiile fără fir, în care canalele sunt adesea afectate de zgomot şi supuse unor rafale de interferenţe care se soldează cu producerea de erori binare adiacente.
Pentru ca un polinom de gradul *n-k* să fie utilizat ca polinom generator al unui cod ciclic (*n, k*) acesta trebuie să fie un factor al polinomului $X^n - 1$.
Polinomul $X^4 + X^3 + 1$, spre exemplu, este un factor al polinomului $X^{15} - 1$, putând servi ca polinom generator pentru un cod ciclic (15,11).
Un alt factor al polinomului $X^{15} - 1$ este polinomul $X^4 + X + 1$, care poate genera un alt cod ciclic (15,11). Polinomul $X^{15} - 1$ are, în fapt, cinci factori primi:

$$X^{15} - 1 = (X + 1)(X^2 + X + 1)(X^4 + X + 1)(X^4 + X^3 + 1)(X^4 + X^3 + X^2 + X + 1).$$

Oricare dintre factorii acestuia ori produse de doi ori mai mulţi factori ai acestuia pot servi ca polinom generator ai unui cod ciclic.
Produsul primilor doi factori $(X + 1)(X^2 + X + 1) = X^3 + 1$, spre exemplu, poate genera un cod ciclic (15,12). Adunarea şi scăderea în aritmetica modulo-2 coincid, astfel are loc egalitatea:

$$X^{15} - 1 = X^{15} + 1.$$

Codul ciclic (5,4), cu 5 biţi {00000, 00011, 00110, 011000, 11000, 10001, 00101, 01010, 10100, 01001, 10010, 01111, 11110, 11101, 11011, 10111}, menţionat anterior, are ca generator polinomul $X + 1$, satisfăcând relaţia:

$$X^5 - 1 = (X + 1)(X^4 + X^3 + X^2 + X + 1).$$

Se poate verifica simplu faptul că $X + 1$ este polinomul generator al codului ciclic (15,12) prin multiplicarea tuturor cuvintelor de cod cu patru biţi, începând cu 0000 şi încheind cu 1111 prin acest polinom $X + 1$, ori 11 reprezentat în binar. Astfel, cuvântul de date 0110 se reprezintă algebric prin polinomul $X^2 + X$, iar acesta multiplicat prin polinomul generator $X + 1$ rezultă: $X^3 + X^2 + X^2 + X = X^3 + X$ reprezentând cuvântul de cod cu 5 biţi 01010.
Multiplicarea prin polinomul generator poate fi realizată şi binar aritmetic, nu numai prin calcul algebric modulo-2 aşa cum se poate remarca din figura 7:

```
  1110 ×
    11
 ------
  1110
 1110
------
 10010
```

Figura 7. Codificarea cuvântului de date 1110
prin multiplicarea cu constanta de generare 11.

De reţinut faptul că un cod ciclic nu este un cod separabil, astfel biţii de date şi biţii de verificare pentru exemplul din figura 7, nu sunt separabili în cuvântul de cod 10010.
Una dintre raţiunile cele mai importante ale utilizării largi ale codurilor ciclice constă în faptul că multiplicarea şi divizarea prin polinomul generator se pot implementa în hardware prin registre de deplasare şi porţi *SAU-EX*. O astfel de implementare permite atât o codificare rapidă cât şi o decodificare foarte eficientă.

Pentru exemplificare se va considera polinomul generator $X^4 + X^3 + 1$, corespunzător multiplicatorului 11001. În acest sens se va considera circuitul prezentat în figura 8 unde sunt utilizate patru bistabile *D* astfel conectate, încât să formeze un registru deplasare cu reacţie liniară (porţile *SAU-EX*).

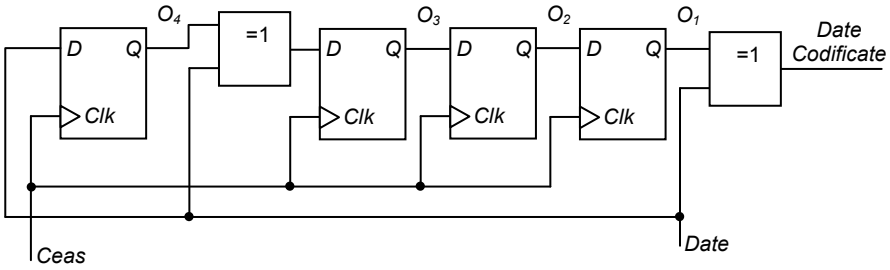



Figura 8. Circuitul de codificare al codului ciclic (15,11) cu polinomul generator $X^4 + X^3 + 1$.

Este destul simplu de constatat faptul că în figura 8 circuitul multiplică serial datele prin polinomul generator $X^4 + X^3 + 1$.

```
  10001100101 ×
        11001
 ---------------
   10001100101
  00000000000
 00000000000
10001100101
10001100101
 ---------------
11000010001101
```

Figura 9. Exemplul unei multiplicări modulo-2 în cadrul codificării unui cuvânt de date cu 11 biţi prin polinomul generator $X^4 + X^3 + 1$.

Coloana marcată în figura 9 arată modul în care se produce cel de-al cincilea bit al produsului prin însumarea modulo-2 a fiecărui bit corespunzător din cuvântului de date 10001100101. Cuvântul de date care urmează să fie codificat ciclic este introdus serial în dispozitivul de codificare, începând cu rangul cel mai puţin semnificativ.

```
110000100011101 : 11001 = 10001100101
 11001
    10100
    11001
     11010
     11001
       11111
       11001
         11001
         11001
         00000
```

Figura 10. Decodificarea cuvântului de date recepţionat prin divizarea în raport cu polinomul generator $X^4 + X^3 + 1$.

Procesul de decodificare se realizează prin divizarea cuvântului de date codificat în raport cu polinomul generator. O ilustrare imediată a procesului de divizare prin polinomul $X^4 + X^3 + 1$ (respectiv 11001) este prezentată în figura 10. În cazul în care mesajul a fost recepţionat fără erori restul împărţirii este nul, aşa cum era de aşteptat.
În cazul în care mesajul a fost modificat prezentând o eroare şi se recepţionează mesajul 11000010**1**11101 (eroarea este marcată prin îngroşare) divizarea prin polinomul generator va produce un rest nenul, aşa cum se poate vedea din figura 11.

```
11000010011101 : 11001 = 10001100101
11001
 10100
 11001
  11011
  11001
   10111
   11001
    11100
    11001
    001011
```

Figura 11. Decodificarea cuvântului de date recepţionat conţinând o singură eroare şi restul nenul obţinut prin divizarea în raport cu polinomul generator $X^4 + X^3 + 1$.

Se poate arăta că orice eroare care afectează un singur bit din cuvântul de date codificat ciclic este întotdeauna descoperită. O eroare care afectează un singur bit poate fi reprezentată prin polinomul $X^i$ iar cuvântul de date codificat ciclic poate fi reprezentat, în acest caz, astfel: $D(X)G(X) + X^i$, unde s-a notat prin $D(X)$ cuvântul iniţial de date şi prin $G(X)$ polinomul generator al codului ciclic. dacă polinomul generator al codului ciclic $G(X)$ are cel puţin doi termeni atunci acesta nu va diviza exact expresia $D(X)G(X) + X^i$ şi în consecinţă se va genera un rest nenul. Aşa cum a fost ales codul ciclic în aceste exemple se poate arăta că acesta are o distanţă Hamming egală cu trei. Aceasta revine la a spune că acest cod ciclic poate detecta orice pereche de biţi eronaţi independent de localizarea acestora în mesaj. Situaţia se schimbă în cazul apariţiei a trei erori simultan prezente în mesajul recepţionat codificat ciclic.

```
1100001101010101 : 11001 = 10001101101
11001
 10111
 11001
  11100
  11001
   10110
   11001
    11111
    11001
     11001
     11001
     00000
```

Figura 12. Decodificarea cuvântului de date recepţionat conţinând trei erori neadiacente, reprezentate prin caractere îngroşate.

În figura 12 este înfăţişată situaţia în care cele trei erori nu sunt adiacente, dovedindu-se că nu poate fi detectată prezenţa acestora (restul divizării mesajul eronat recepţionat, prin polinomul generator este nul).

```
1100000011011101 : 11001 = 10001110011
11001
 10011
 11001
  10100
  11001
   11011
   11001
    10110
    11001
     11111
     11001
      00110
```

Figura 13. Decodificarea cuvântului de date recepţionat conţinând trei erori adiacente, reprezentate prin caractere îngroşate.

Dar, dacă cele trei valori binare sunt adiacente, atunci codul ciclic considerat le detectează, aşa cum se poate vedea din exemplul prezentat în figura 13, restul ne-nul arătând acest fapt.

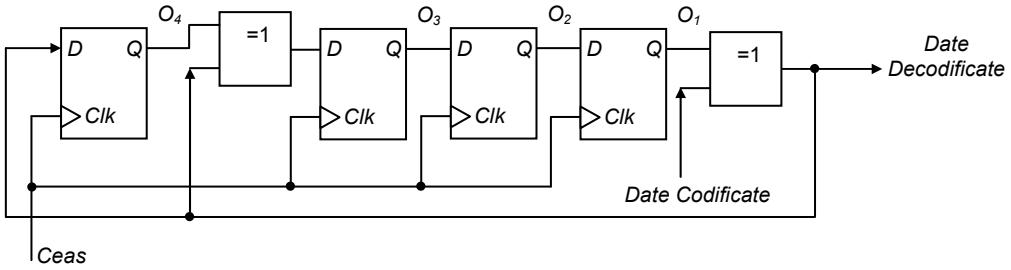



Figura 14. Circuitul decodificator al codului ciclic (15,11) cu polinomul generator $X^4 + X^3 + 1$.

Pentru determinarea unui circuit logic care sa efectueze decodificarea unui mesaj transmis fără existe erori se va considera un exemplu practic. Se notează prin $E(X)$ un mesaj $D(X)$ codificat ciclic prin schema logică din figura 8, în care se utilizează polinomul generator $G(X) = X^4 + X^3 + 1$.

Dacă mesajul $E(X)$ este recepţionat fără erori, adică restul împărţirii este nul, atunci putem scrie că:

$$E(X) = D(X) \cdot G(X).$$

Deoarece, în acest caz, polinomul generator $G(X) = X^4 + X^3 + 1$, se poate rescrie relaţia anterioară astfel:

$$E(X) = D(X) \cdot ( X^4 + X^3 + 1),$$

sau încă:

$$D(X) = E(X) - D(X)( X^4 + X^3),$$

dar deoarece în cazul sumelor modulo-2 diferenţa este tot suma rezultă:

$$D(X) = E(X) + D(X)( X^4 + X^3).$$

Circuitul corespunzător decodificatorului este înfăţişat în figura 14.

În multe aplicaţii de transmisia datelor este necesar să se asigure că orice rafală de erori, cu lungimea 16 sau mai mică, este detectată. În acest scop sunt utilizate coduri ciclice de tipul, (16 + *k*, *k*). Generarea polinoamelor de gradul 16 se va alege în aşa fel încât numărul maxim de biţi de date este suficient de mare astfel încât să permită utilizarea aceluiaşi cod (şi respectiv aceleaşi circuite de codificare şi decodificare) independent de lungimea blocurilor de date codificate, în genere. Acestea sunt polinoamele CRC-16 (CRC fiind abrevierea denumirii *Cyclic Redundancy Check*),

$$G(X) = (X + 1)(X^{15} + X + 1) = X^{16} + X^{15} + X^2 + 1,$$

Iar polinomul CRC-CCITT este:

$$G(X) = (X + 1)(X^{15} + X^{14} + X^{13} + X^{12} + X^4 + X^3 + X^2 + X + 1) = X^{16} + X^{12} + X^5 + 1,$$

În ambele cazuri polinoamele de grad 16 divid polinomul $X^n - 1$ pentru $n = 2^{15}$ -1, dar nu pentru orice valoare mai mică a parametrului *n*. Astfel, acest cod poate fi utilizat pentru blocuri de date a căror mărime ajunge până la $2^{15} - 1 = 32\ 767$ biţi. În acest sens, este util de remarcat faptul că blocurile mai scurte de date, decât 32 767 biţi, pot fi tamponate cu un număr suficient de 0-uri la început, acestea fiind ignorate în operaţiile de codificare şi decodificare.
O altă remarcă utilă priveşte numărul mic de coeficienţi nenuli anume doar patru. Aceasta simplifică mult proiectarea circuitelor pentru codificare şi pentru decodificare.

În transferurile de date prin Internet se utilizează mult codul CRC-32:

$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

Acest cod permite detectarea unor rafale de erori care pot conduce până la 32 de biţi pentru blocuri de date având mărimea mai mică sau egală cu $n = 2^{32} - 1$, biţi.

În cazul transmisiilor de date prin blocuri lungi, este mult mai eficient să se utilizeze o codificare separabilă. Aceasta pentru că în această manieră se permite utilizarea imediată a datelor primite fără să fie necesar să se aştepte primirea şi decodificarea tuturor biţilor din cuvântul de cod. Un cod ciclic separabil va permite detecţia erorilor independent de procesarea datelor propriuzise.
Există, din fericire, o cale simplă de generare a unui cod ciclic (*n*, *k*) separabil. Astfel, în loc să se codifice un cuvânt de date de forma:

$$D(X) = d_{k-1}X^{k-1} + d_{k-2}X^{k-2} + \ldots + d_0,$$

prin multiplicarea acestuia cu polinomul sau generator $G(X)$ de grad $n$-$k$, se adaugă întâi ($n$-$k$) zerouri la polinomul $D(X)$, obţinându-se polinomul:

$$D^*(X) = d_{k-1}X^{n-1} + d_{k-2}X^{n-2} + \ldots + d_0 X^{n-k}\ .$$

Ulterior se divide $D^*(X)$ prin $G(X)$ rezultând:

$$D^*(X) = Q(X)G(X) + R(X),$$

Unde $R(X)$ este un polinom de grad mai mic decât $n$-$k$.
În sfârşit se formează cuvântul cod:

$$C(X) = D^*(X) - R(X),$$

care va fi transmis. Acest cuvânt de cod cu $n$ biţi va avea drept factor polinomul $G(X)$ şi pe cale de consecinţă dacă după divizarea acestuia se va obţine un rest nul, rezultă că nu s-au înregistrat erori. În această codificare, $D^*(X)$ şi $R(X)$ nu au nici un termen în comun şi, astfel, primii $k$ biţi din $C(X) = D^*(X) - R(X) = D^*(X) + R(X)$ sunt biţii iniţiali, originali, iar restul de $n$-$k$ sunt biţi de verificare, făcând astfel codificarea separabilă.

#### 4. Codurile aritmetice

Codurile de erori aritmetice sunt acele coduri care sunt păstrate în cadrul unei anumite mulţimi de operaţii aritmetice. Această proprietate permite detectarea erorilor care pot avea loc pe durata execuţiei unei operaţii aritmetice din mulţimea de operaţii aritmetice respective.

Se spune că un cod este păstrat în raport cu o operaţie aritmetică * dacă pentru orice doi operanzi $X$ şi $Y$, şi pentru entităţile codificate corespunzător $X'$ şi $Y'$, există o altă operaţie aritmetică # pentru operanzii codificaţi astfel încât să fie satisfăcută relaţia:

$$X' \# Y' = (X * Y)'. \qquad (4.1)$$

Această relaţie poate fi interpretată astfel:
Rezultatul operaţiei aritmetice # va produce acelaşi rezultat ca şi prin codificarea rezultatului operaţiei * aplicată operanzilor $X$ şi $Y$.

Codurile aritmetice se aşteaptă, de regulă, să detecteze toate defectele care afectează un singur bit. Este de remarcat faptul că, totuşi, o eroare a unui singur bit dintr-un operand sau dintr-un rezultat intermediar poate, adesea, să cauzeze o eroare a mai multor biţi. La sumarea a doi operanzi binari un defect în rangului $i$ al sumatorului poate cauza erori, eventual, în toate rangurile $n - i$ de ordin superior ale sumatorului.

Există două clase de coduri aritmetice:

- Codurile aritmetice separabile, şi
- Codurile aritmetice neseparabile.

Cele mai simple coduri neseparabile sunt codurile $AN$, obţinute prin multiplicarea operanzilor printr-o constantă $A$.
Cu alte cuvinte, $X'$ din relaţia (4.1) se obţine prin $A \cdot X$, iar operaţiile * şi # sunt identice pentru adunarea şi înmulţire. Astfel, dacă $A = 3$, atunci se multiplică cu 3 fiecare operand cu 3 şi se verifică rezultatul unei adunări ori scăderi pentru a vedea dacă acesta este un multiplu de 3 ori nu. Toate erorile de magnitudine care sunt multipli a constantei $A$ sunt nedetectabile. Prin urmare nu vor nu se vor alege valori ale constantei $A$ care sunt puteri ale lui 2 (baza numerelor din sistemele de calcul). O valoare impară a constantei $A$ va detecta toate defectele care afectează un singur digit,

deoarece o astfel de eroare are magnitudinea $2^i$. Prin alegerea valorii $A = 3$, se ajunge la cel mai convenabil cod *AN* care poate încă detecta toate erorile singulare.
Numărul $0110_2 = 6_{10}$, este reprezentat în cod *AN*, cu $A = 3$, prin $010010_2 = 18_{10}$. Un defect al bitului cu rangul trei, spre exemplu, poate conduce la valoarea eronată $011\mathbf{1}010_2 = 26_{10}$. Această eroare va fi simplu de detectat deoarece valoarea 26 nu este un multiplu al lui 3.

Cele mai simple coduri aritmetice separabile sunt codul rest şi codul invers al restului, care se mai numesc şi coduri reziduale. Pentru fiecare dintre aceste coduri se ataşează un simbol de verificare, notat prin $C(X)$, fiecărui operand $X$. pentru fiecare cod rest, $C(X) = X \bmod A = |X|_A$, unde $A$ se numeşte modulul de verificare. În cazul codului invers al restului, simbolul de verificare se calculează astfel:

$$C(X) = A - (X \bmod A).$$

Relaţia (4.1) se rescrie, în acest caz, astfel:

$$C(X) \# C(Y) = C(X * Y).$$

Egalitatea anterioară se păstrează pentru sume şi produse deoarece se aplică următoarele relaţii:

$$|X + Y|_A = ||X|_A + |Y|_A|_A,$$
$$|X \cdot Y|_A = ||X|_A \cdot |Y|_A|_A.$$

*Exemplul 4.1*

Considerând un exemplu simplu în care $A = 3$, $X = 7$, iar $Y = 5$, rezultă:
Resturile sunt $|X|_A = 1$ şi $|Y|_A = 2$.
În urma sumării celor doi operanzi se obţine:

$$|7 + 5|_3 = ||7|_3 + |5|_3|_3 = |1+2|_3 = 0.$$

Multiplicând cei doi operanzi, se obţine:

$$|7 \cdot 5|_3 = 2 = ||7|_3 \cdot |5|_3| = |1 \cdot 2|_3 = 2.$$

◊

Pentru împărţire se dovedeşte că ecuaţia $X - S = Q \cdot D$, este satisfăcută, unde s-au utilizat notaţiile: $X$ este deîmpărţitul, $D$ este împărţitorul, $Q$ este câtul, iar $S$ este restul. Din acest motiv verificarea corespunzătoare restului arată astfel:

$$||X|_A - |S|_A|_A = ||Q|_A \cdot |D|_A|_A$$

*Exemplul 4.2*

Considerând acelaşi exemplu simplu în care $A = 3$, $X = 7$, iar $Y = 5$, rezultă $Q = 1$ iar $S = 2$. Verificarea corespunzătoare a resturilor este:
$||7|_3 - |2|_3|_3 = ||5|_3 \cdot |1|_3|_3 = 2$.
Scăderea din membrul stâng al expresiei precedente este efectuată prin adunarea complementului modulo-3, adică: $|1 + |3-2|_3|_3 = |1 + 1|_3 = 2$.

◊

Un cod rest având modulul de verificare $A$ are aceeaşi magnitudine a erorii nedetectabile ca şi codul corespunzător *AN*. Dacă $A = 3$, spre exemplu, vor fi nedetectate doar erorile care vor modifica rezultatul printr-un multiplu faţă de trei. Rezultă că toate erorile care constau din modificarea unui singur bit sunt întotdeauna

detectabile. Mai mult, algoritmii de verificare corespunzători codului *AN* şi codului rest sunt aceeaşi: în amândouă cazurile trebuie determinat restul modulo-*A* al rezultatului. Chiar şi creşterea lungimii cuvântului, $|\log_2 A|$ este aceeaşi în ambele cazuri.
Cea mai importantă deosebire apare din proprietatea de separabilitate. Unitatea aritmetică a simbolului de verificare $C(X)$ este complet separată de unitatea principală de operare pentru entitatea $X$, în timp ce, în cazul codului *AN*, există o singură unitate de calcul (de complexitate mai mare).
Un sumator cu cod rest este prezentat în figura 15. Blocul de detecţie al erorii calculează restul modulo-*A* al sumei $|X + Y|_A$ iar acesta este comparat cu rezultatul determinat de sumatorul modulo-*A*, $|X|_A + |Y|_A$ . Dacă acestea sunt diferite, atunci este determinată o eroare.

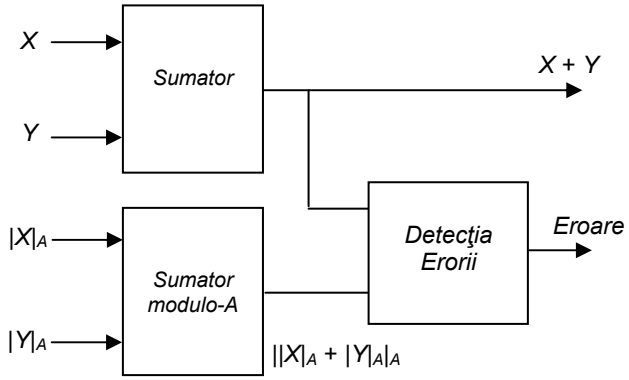



Figura 15. Sumator cu verificarea separată a restului.

Codurile *AN* şi cu rest având $A = 3$ sunt cele mai simple exemple de clase de coduri aritmetice (atât separabile cât şi neseparabile). Care utilizează valoarea parametrului $A$ de forma $A = 2^n - 1$, $n$ fiind un număr natural.
Această alegere simplifică mult calculul restului atunci când sunt operate împărţiri prin $A$ (necesare prin algoritmul de verificare) şi pentru acest motiv astfel de coduri sunt numite *coduri aritmetice cu costuri reduse*. Calculul restului atunci când are loc o împărţire prin $A = 2^n - 1$ este simplu, deoarece ecuaţia:

$$| z_i r^i |_{r-1} = | z_i |_{r-1}, r = 2^n$$

permite utilizarea sumării modulo - $(2^n - 1)$ a grupurilor de $n$ biţi, care alcătuiesc numărul (fiecare grup are valoarea $0 \le z_i \le 2^n - 1$).

*Exemplul 4.3*

Calculul restului atunci când numărul binar $X = 11110101011$ este împărţit prin $A = 7 = 2^3 - 1$, se face astfel:
- se partiţionează $X$ în grupuri de câte 3 biţi, începând cu ultimul bit semnificativ. Aceasta conduce în cazul de faţă la următoarea grupare:
$X = (z_3, z_2, z_1, z_0) = (11, 110, 101, 011)$.
- sumează aceste grupuri modulo-7, adică se sumează iar dacă apare transportul din rangul superior acesta este re-întors la rangul inferior,

deoarece acest transport are ponderea 8, iar $|8|_7 = 1$. Calculul concret este prezentat mai jos:

```
     11 z3
+  110 z2
-------------------------------------------------
1  001
+     1 transportul se sumează la rangul cel mai puţin semnificativ.
-------------------------------------------------
   010
+  101  z1
-------------------------------------------------
   111
+  011 z0
-------------------------------------------------
1  010
      1 transportul se sumează la rangul cel mai puţin semnificativ.
-------------------------------------------------
+  011
```

În adevăr, restul modulo-7 al numărului binar $X$ = 11110101011, este 3 acesta fiind restul modulo-7 al numărului $1963_{10}$.

◊

Atât codurile separabile cât şi codurile ne-separabile se păstrează atunci când sunt realizate operaţii aritmetice asupra unor operanzi fără semn. Dacă se doreşte şi includerea operanzilor cu semn, atunci este necesar să fie codul complementabil în raport cu $R$, unde $R$ este fie $2^n$, sau $2^n - 1$ ($n$ fiind numărul de biţi al operandului codificat). În funcţie de alegerea valorii parametrului $R$ se va determina dacă se va opera într-o aritmetică cu complement faţă de doi ($R = 2^n$) sau într-o aritmetică cu complement faţă de unu ($R = 2^n - 1$).
Pentru codurile $AN$ , expresia $R - AX$ trebuie să fie divizibilă prin $A$, iar aceasta impune ca $A$ să fie un factor întreg al $R$. Dar dacă este necesar ca parametrul $A$ să fie impar atunci aceasta exclude alegerea $R = 2^n$, rămânând doar utilizarea complementului faţă de 1.

*Exemplul 4.4*

Se consideră $n = 4$, atunci $R$ are valoarea $2^n - 1 = 15$, pentru complementul faţă de unu şi este divizibil prin $A$ pentru codul $AN$ cu codul $A = 3$. Numărul $X = 0110$ este reprezentat prin $3X = 010010$, iar complementul acestuia (faţă de unu) este 101101 (echivalent zecimal 45), fiind divizibil prin 3.
Dacă $n = 5$, atunci complementul faţă de unu $R$ este 31 şi acesta nu este divizibil prin $A$. Numărul $X = 00110$ este reprezentat prin $3X = 0010010$, iar complementul faţă de unu al acestuia este 1101101 ($109_{10}$), nefiind divizibil prin 3.

◊

Pentru codul cu rest (rezidual), cu valoarea modulului de verificare $A$, trebuie satisfăcută ecuaţia $A - |X|_A = |R - A|_A$. Acesta fapt conduce la concluzia că $R$ trebuie să fie întreg multiplu al lui $A$, implicând încă odată utilizarea complementului faţă de unu. Se poate totuşi modifica procedeul astfel încât să se utilizeze complementul faţă de doi (cu $R = 2^n$):

$$|2^n - X|_A = |2^n - 1 - X + 1|_A = |2^n - 1 - X|_A + |1|_A.$$

Din acest motiv este necesară adăugarea unui termen corector $|1|_A$ la codul rezidual atunci când se calculează complementul faţă de doi. De reţinut că parametrul $A$ trebuie să fie în continuare un factor al expresiei $2^n - 1$.

*Exemplul 4.5*

Pentru codul rezidual cu $A = 7$, iar $n = 6$, $R = 2^6 = 64$ complementul faţă de $R - 1 = 63$, este divizibil prin 7. Numărul $001010_2 = 10_{10}$ are reziduul 3 modulo – 7. Complementul faţă de doi al numărului $001010_2$ este 110110. Complementul lui $|3|_7$ este $|4|_7$, iar prin adunarea termenului de corecţie $|1|_7$ se obţine 5, ceea ce este reziduul corect modulo-7 al numărului binar 110110 (echivalentul zecimal fiind 54).

◊

O corecţie similară este necesară atunci când sunt sumaţi operanzi reprezentând complemente faţă de doi şi se generează un transport din rangul cel mai semnificativ (cu ponderea $2^n$) în sumator. Un astfel de transport este neglijat în regulile de calcul aritmetice ale complementului faţă de doi.

*Exemplul 4.6*

Se însumează numărul $X = 110110$ (în complement faţă de doi) cu numărul $Y = 001101$. Se generează un transport din rangul cel mai semnificativ, care este ignorat.

Din acest motiv trebuie scăzut termenul de corecţie $|2^6|_7 = |1|_7$ din reziduul de verificare cu modulul $A = 7$, obţinând 3 ceea ce este evident reziduul modulo-7 corect al rezultatului 000011.

```
  110110  = X          101 = |X|7
+ 001101  = Y        + 110 = |Y|7
1 000011             1 011
                       1 transportul din rangul maxim
                     100
                   -   1 termenul de corecţie
                     011
```

◊

Modificările introduse anterior conduc la o interdependenţă a aritmeticii unităţii de calcul cu unitatea de verificare care operează cu reziduurile. Astfel de interdependenţe pot cauza situaţii în care o eroare din unitatea de calcul aritmetic se propagă în unitatea de verificare iar prezenţa erorii este mascată. S-a demonstrat, totuşi că apariţia unei singure erori a fost întotdeauna detectabilă.

Corectarea unei erori poate fi realizată prin utilizarea a două sau mai multe reziduuri de verificare. Cel mai simplu caz este codul bi-rezidual. Acesta constă din două reziduuri de verificare $A_1$ şi $A_2$.

Dacă $n$ este numărul de biţi din operanzi, se aleg două constante naturale $a$ şi $b$ astfel încât $n$ este cel mai multiplu comun al celor două constante naturale $a$ şi $b$.

Dacă $A_1 = 2^a - 1$, iar $A_2 = 2^b - 1$, sunt două reziduuri de verificare, atunci poate fi corectată orice eroare a unui singur bit.