1 BAZELE MATEMATICE ALE CALCULATOARELOR

1.1 Reprezentarea informaţiei

Informaţiile prelucrate prin sistemele de calcul sunt de diverse tipuri dar ele sunt reprezentate la nivel elementar sub formă binară. O informaţie elementară corespunde deci unei cifre binare (0 sau 1) numită **bit.** O informaţie mai complexă (un caracter, un număr etc.) se exprimă printr-o mulţime de biţi.

Codificarea unei informaţii constă în a stabili o corespondenţă între reprezentarea externă a informaţiei (caracterul A sau numărul 33, de exemplu) şi reprezentarea sa internă, care este o secvenţă de biţi.

Avantajele reprezentării binare se referă în special la facilitatea de realizare tehnică cu ajutorul elementelor bistabile (sisteme cu 2 stări de echilibru) precum şi la simplitatea efectuării operaţiilor fundamentale sub forma unor circuite logice, utilizând logica simbolică cu două stări (0, 1).

Informaţiile prelucrate în sistemele de calcul sunt de două tipuri: **instrucţiuni** şi **date.**

Instrucţiunile, scrise în limbaj maşină, reprezintă operaţiile efectuate în sistemul de calcul şi ele sunt compuse din mai multe câmpuri:

- **codul** operaţiei de efectuat;
- **operanzii** implicaţi în operaţie.

Codul operaţiei trebuie să suporte o operaţie de **decodificare** (transformare inversă codificării) pentru a se putea efectiv executa.

Datele sunt operanzii asupra cărora acţionează operaţiile (prelucrările), sau sunt produse de către acestea. O adunare, de exemplu, se aplică la doi operanzi, furnizând un rezultat care este suma acestora.

Se pot distinge datele numerice, rezultat al unei operaţii aritmetice, sau date nenumerice, de exemplu simbolurile care constituie un text.

Date nenumerice

Datele nenumerice corespund caracterelor alfanumerice: A, B, ..., Z, a, b, ..., z, 0, 1, 2, ..., 9 şi caracterelor speciale: ?, !, **“**, $, ;, ...

Codificarea se realizează pe baza unei tabele de corespondenţă specifică fiecărui cod utilizat. Printre cele mai cunoscute coduri putem enumera:

- **BCD** [Binary Coded Decimal] prin care un caracter este codificat pe 6 biţi;
- **ASCII** [American Standard Code for Information Interchange] (7 biţi);
- **EBCDIC** [Extended Binary Coded Decimal Internal Code] (8 biţi).

Figura următoare prezintă corespondenţa dintre diferite coduri.

Figura 1.1 Tabela de corespondenţă între coduri

caracter	BCD	ASCII	EBCDIC
0	000000	0110000	11110000
1	000001	0110001	11110001
2	000010	0110010	11110010
...	...	...	
...			
9	001001	0111001	11111001
A	010001	1000001	11000001
B	010010	1000010	11000010
C	010011	1000011	11000011
	(6 biţi)	(7 biţi)	(8 biţi)

Datele numerice

Datele numerice sunt de următoarele tipuri:

a) **numere întregi pozitive sau nule**: 0; 1; 315...
b) **numere intregi negative**: -1; -155...
c) **numere fracţionare**: 3.1415; -0.5...
d) **numere în notaţie ştiinţifică**: 4.9×10^7 ; 10^{23} ...

Codificarea se realizează cu ajutorul unui **algoritm de conversie** asociat tipului de dată corespunzător. Operaţiile aritmetice (adunare, scădere, înmulţire, împărţire) care se pot aplica asupra acestor date se efectuează de regulă în aritmetica binară. Figura de mai jos arată regulile operaţiilor binare.

0 + 0 = 0	0 × 0 = 0
0 + 1 = 1	0 × 1 = 0
1 + 0 = 1	1 × 0 = 0
1 + 1 = 10	1 × 1 = 1

Numerele întregi pozitive sau nule cuprind: 0, 1, 2, ...,N, N + 1...

Sisteme de numeraţie

Un sistem de numeraţie face să-i corespundă unui număr N, un anumit simbolism scris şi oral. Într-un sistem de numeraţie cu baza p > 1, numerele 0, 1, 2, ..., p –1 sunt numite **cifre**.

Orice număr întreg pozitiv poate fi reprezentat astfel:

$$N = a_n p^n + a_{n-1} p_{n-1} + ... + a_1 p + a_0$$

cu $a_i \in \{0, 1, 2, p-1\}$ şi $a_n \neq 0$.

Se utilizează de asemenea notaţia echivalentă $N = a_n a_{n-1} ... a_1 a_0$.

Numerele scrise în sistenul de numeraţie cu baza 2 (binar) sunt adesea compuse dintr-un mare număr de biţi, şi de aceea se preferă exprimarea acestora în sistemele **octal** (p = 8) şi **hexazecimal** (p = 16), deoarece conversia cu sistemul binar este foarte simplă.

Schimbări de bază

a) **binar ⇒ zecimal**

Conversia se realizează prin însumarea puterilor lui 2 corespunzătoare biţilor egali cu 1;

Exemplu: $10101_2 = 2^4 + 2^2 + 2^0 = 16 + 4 + 1 = 21_{10}$

b) **zecimal ⇒ binar**

Conversia se efectuează prin împărţiri întregi succesive cu 2. Testul de oprire corespunde situaţiei câtului nul. Numărul binar este obţinut considerând resturile în ordinea inversă.

Exemplu: Conversia lui 26:

26 : 2 = 13	rest 0
13 : 2 = 6	rest 1
6 : 2 = 3	rest 0
3 : 2 = 1	rest 1
1 : 2 = 0	rest 1

Se obţine (de jos în sus): $26_{10} = 11010_2$.

c) **octal (hexazecimal) ⇒ zecimal**

Conversia se reduce la însumarea puterilor lui 8 (16).

d) **zecimal ⇒ octal (hexazecimal)**

Conversia se efectuează prin împărţiri întregi succesive prin 8 (16). Testul de oprire corespunde situaţiei câtului nul. Numărul octal (hexazecimal) este obţinut considerând resturile obţinute de la ultimul către primul.

e) **octal (hexazecimal) ⇒ binar**

Conversia corespunde dezvoltării fiecărei cifre octale (hexazecimale) în echivalentul ei binar pe 3 (4) biţi.

Exemplu:

$27_8 = 010'111_2$ deoarece $2_8 = 010_2$ şi $7_8 = 111_2$.

$3A_{16} = 0011'1010_2$ deoarece $3_{16} = 0011_2$ şi $A_{16} = 1010_2$.

f) **binar ⇒ octal (hexazecimal)**

Conversia se realizează înlocuind de la dreapta la stânga, grupele de 3 (4) biţi prin cifra octală (hexazecimală) corespunzătoare. Dacă numărul de

biţi nu este multiplu de 3 (4) se completează configuraţia binară la stânga cu zerouri.

Exemplu: $101011_2 = 53_8 = 2B_{16}$.

Numere întregi negative

Numerele întregi negative pot fi codificate prin trei metode:

- **semn şi valoare absolută (SVA)**;
- **complement logic sau restrâns sau faţă de 1 (C1)**;
- **complement aritmetic sau adevărat sau faţă de 2 (C2)**;

Prin metoda **" semn şi valoare absolută"**, numerele se codifică sub forma*: ± valoare absolută*.

Prin această reprezentare se sacrifică un bit pentru semn. În mod normal, 0 este codul semnului +, iar 1 este codul semnului -. În aceste condiţii, pe un cuvânt de k biţi se pot reprezenta numere întregi pozitive şi negative N, astfel încât: $-(2^{k-1}-1) \le N \le (2^{k-1}-1)$.

Această metodă de reprezentare prezintă unele inconveniente:

- numărul zero are două reprezentări distincte: 000...0 şi 100...0, adică +0 şi - 0;
- tabelele de adunare şi înmulţire sunt complicate din cauza bitului de semn care trebuie tratat separat.

Complement logic şi aritmetic

Complementul logic (complement faţă de 1) se calculează înlocuind, pentru valorile negative, fiecare bit 0 cu 1 şi 1 cu 0.

Complementul aritmetic (complement faţă de 2) este obţinut adunând o unitate la valoarea complementului logic.

Exemplu: Reprezentarea numărului (-6) pe 4 biţi: +6 = 0110

Semn şi valoare absolută:	- 6 = 1110
Complement faţă de 1:	- 6 = 1001
Complement faţă de 2:	- 6 = 1010

Se poate uşor constata că intervalul numerelor întregi N care se pot reprezenta în complement faţă de 1 este acelaşi ca şi pentru reprezentarea **"**semn şi valoare absolută**"**.

Pentru reprezentarea în complement faţă de 2 există o valoare în plus, deci pentru k biţi vom avea: $-2^{k-1} \le N \le (2^{k-1}-1)$.

Se poate remarca faptul că bitul cel mai din stânga (bitul de semn) este întotdeauna 0 pentru numere pozitive şi 1 pentru cele negative şi aceasta pentru fiecare din cele trei reprezentări, conform tabelului următor.

(16 biţi $\Rightarrow 2^{16} = 65536 = 2 \times 32768$ valori posibile)

zecimal	semn şi valoare absolută	complement faţă de 2	complement faţă de 1
+32767	0111...1...1111	0111...1...1111	0111...1...1111
+32766	0111...1...1110	0111...1...1110	0111...1...1110
...	...	...	...
+1	0000...0...0001	0000...0...0001	0000...0...0001
+0	0000...0...0000	0000...0...0000	0000...0...0000
-0	1000...0...0000	----------	1111...1...1111
-1	1000...0...0001	1111...1...1111	1111...1...1111
...	...	...	...
-32766	1111...1...1110	1000...0...0010	1000...0...0001
-32767	1111...1...1111	1000...0...0001	1000...0...0000
-32768	----------	1000...0...0000	----------

Reprezentarea în complement faţă de 1 recunoaşte două zerouri (+0 şi –0), dar este simetrică, deoarece aceleaşi numere pozitive şi negative sunt reprezentabile, iar această situaţie se poate uşor realiza electronic.

În complement faţă de 1 sau faţă de 2, operaţiile aritmetice sunt avantajoase, deoarece operaţia de scădere se realizează prin adunarea complementului.

Într-o adunare în complement faţă de 1, o cifră de transport către ordinul superior generată de bitul de semn trebuie adăugată la rezultatul obţinut, spre deosebire de complementul faţă de 2, când această cifră de transport se ignoră.

În complement faţă de 1 sau 2 nu se produce depăşire de capacitate decât în cazul în care cifrele de transport generate de bitul de semn şi de bitul anterior acestuia sunt diferite.

Numere fracţionare

Numerele fracţionare sunt numerele subunitare.

Schimbări de bază

a) **binar $\Rightarrow$ zecimal**

Conversia se face adunând puterile corespunzătoare ale lui 2.

Exemplu: $0.01_2 = 0 \times 2^{-1} + 1 \times 2^{-2} = 0.25_{10}$.

b) **zecimal $\Rightarrow$ binar**

Conversia se efectuează prin înmulţiri succesive cu 2 a numerelor pur fracţionare. Acest algoritm trebuie să se termine când se obţine o parte

fracţionară nulă sau când numărul de biţi obţinuţi corespunde mărimii registrului sau a cuvântului de memorie în care se va stoca valoarea. Numărul binar se obţine citind părţile întregi în ordinea calculării lor.

Exemplu: $0.125 \times 2 = 0.250 = 0 + 0.250$

$0.25 \times 2 = 0.50 = 0 + 0.50$

$0.50 \times 2 = 1.0 = 1 + 0.0$

Vom considera părţile întregi de sus în jos, deci: $0.25_{10} = 0.001_2$.

Pentru numerele fracţionare se pot remarca reprezentările în virgulă fixă şi virgulă mobilă.

Virgula fixă (VF)

Sistemele de calcul nu posedă virgula la nivelul maşinii, deci reprezentarea numerelor fracţionare se face ca şi când acestea ar fi întregi, cu o virgulă virtuală a cărei poziţie este controlată de către programator.

Datorită dificultăţii de gestionare a virgulei de către programator (pot apare frecvent situaţii de depăşire a capacităţii de memorare), se preferă soluţia aritmeticii în virgulă mobilă.

Virgula mobilă (VM)

Primele sisteme de calcul utilizau doar virgula fixă pentru efectuarea operaţiilor aritmetice, iar către sfârşitul anilor ‘50, în urma apariţiei logicii cablate s-a introdus pe scară largă reprezentarea în virgulă mobilă a numerelor fracţionare.

În majoritatea sistemelor de calcul actuale destinate în special aplicaţiilor de natură tehnico-ştiinţifică, cele două metode de reprezentare (virgula fixă şi virgula mobilă) coexistă şi sunt foarte utile.

Reprezentarea în virgulă mobilă constă în a reprezenta numerele sub forma următoare:

$N = M \times B^E$ cu: B = baza (2, 8, 10, 16...)
M = mantisa
E = exponentul

Exponentul este un număr întreg, mantisa **normalizată** este un număr pur fracţionar (fără cifre semnificative la partea întreagă). Cu excepţia numărului zero (în general reprezentat prin cuvântul 000...0), vom avea întotdeauna: $0.1_2 \le |M| < 1_2$, sau $0.5_{10} \le |M| < 1_{10}$.

Exponentul şi mantisa trebuie să poată reprezenta atât numere pozitive cât şi negative. Cel mai adesea, mantisa admite o reprezentare sub forma **"**semn şi valoare absolută**"**, iar exponentul este fără semn, dar **decalat.**

Exemplu

unde SM este semnul mantisei, ED este exponentul decalat şi M mantisa.

Pentru nu număr de k biţi rezervaţi pentru ED se pot reprezenta fără semn 2^k valori, de la 0 la $2^k - 1$. Decalajul considerat este 2^{k-1}, ceea ce permite ca valorile de la 0 la 2^{k-1}-1 pentru ED să corespundă unui exponent real (ER) negativ, iar valorile de la 2^{k-1} la $2^k - 1$ ale lui ED să corespundă unui exponent real (ER) pozitiv. Deci domeniul de valori reprezentabile pentru exponentul real este de la -2^{k-1} la 2^{k-1}-1.

De exemplu, pentru k = 4, pe cei 4 biţi putem reprezenta fără semn numere de la 0 la 15 pentru ED. Decalajul considerat este $2^{k-1} = 2^3 = 8$, deci pentru exponentul real (ER) putem considera valori de la $-2^{k-1} = -2^3 = -8$ şi până la $2^{k-1} - 1 = 2^3 - 1 = 7$.

Relaţia existentă se poate scrie astfel: ER = ED – D.

Exponentul determină intervalul de numere reprezentabile în sistemul de calcul, iar numerele prea mari pentru a putea fi reprezentate corespund unei **"**depăşiri superioare**"** de capacitate de memorare [overflow], iar numerele prea mici corespund unei **"**depăşiri inferioare**"** de capacitate de memorare [underflow].

Mărimea mantisei exprimă precizia de reprezentare a numerelor.

Avantajul utilizării virgulei mobile faţă de virgula fixă constă în intervalul mult mai extins al valorilor posibile de reprezentat.

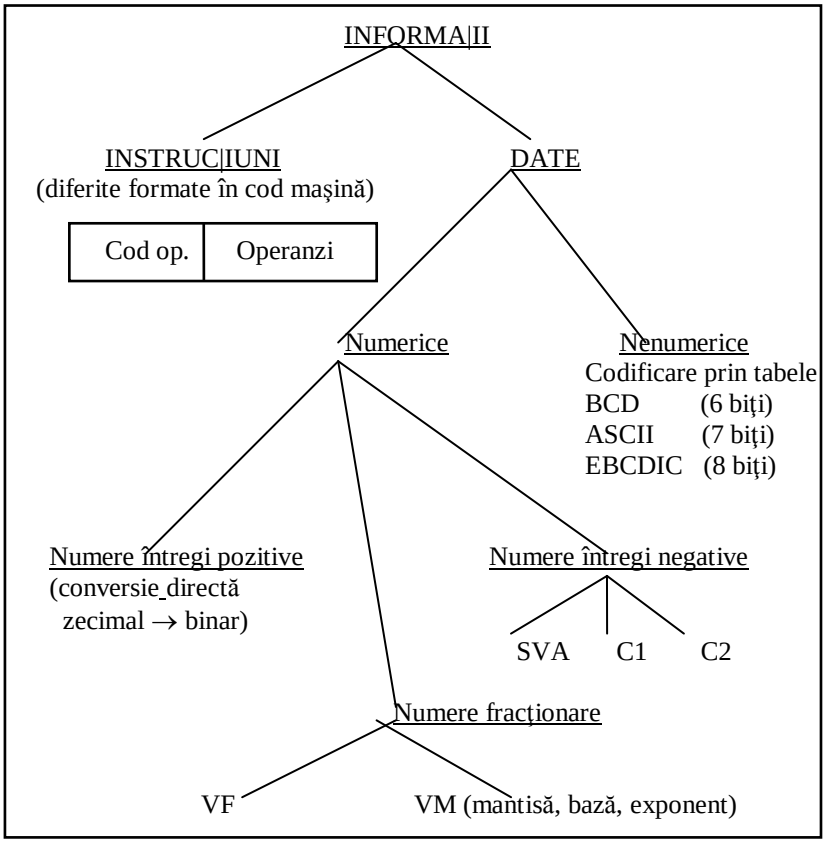
Figura următoare prezintă schematic tipurile diverse de informaţii prelucrate în sistemele de calcul.

Standardul IEEE 754

Standardul IEEE [Institute of Electrical and Electronics Engineers] defineşte trei formate de reprezentare a numerelor în virgulă mobilă:

a) **simplă precizie** pe 32 de biţi (1 bit pentru SM, 8 biţi pentru ED şi 23 pentru M);
b) **dublă precizie** pe 64 biţi (1 bit pentru SM, 11 biţi pentru ED şi 52 biţi pentru M);
c) **dublă precizie extinsă** pe 96 biţi (1 bit pentru SM, 15 biţi pentru ED şi 80 biţi pentru M) .
d) **precizie cvadruplă** pe 128 biţi (1 bit pentru SM, 15 biţi pentru ED şi 112 biţi pentru M)

Exponentul este decalat cu 128 pentru reprezentarea în simplă precizie şi cu 1024 pentru reprezentarea în dublă precizie. Mantisa fiind normalizată, există siguranţa că primul bit al mantisei are valoarea 1, ceea ce permite omiterea sa (**bit ascuns**) pentru creşterea preciziei de reprezentare, dar complică prelucrarea informaţiei.



Coduri zecimale codificate în binar

Dacă aritmetica sistemelor de calcul este de regulă binară, ea poate fi de asemenea şi zecimală. În cazul calculatoarelor de buzunar şi de birou, în sistemele de calcul specifice aplicaţiilor comerciale, operaţiile se efectuează direct asupra reprezentării zecimale a numerelor.

Un număr zecimal, care cuprinde una sau mai multe cifre (de la 0 la 9), este codificat cu ajutorul biţilor utilizând anumite coduri. Tabela de mai jos prezintă patru exemple de astfel de coduri.

zecimal	BCD	excedent-3	2 din 5	bicvintal
0	0000	0011	00011	01 00001
1	0001	0100	00101	01 00010
2	0010	0101	00110	01 00100
3	0011	0110	01001	01 01000
4	0100	0111	01010	01 10000
5	0101	1000	01100	10 00001
6	0110	1001	10001	10 00010
7	0111	1010	10010	10 00100
8	1000	1011	10100	10 01000
9	1001	1100	11000	10 10000

Exemplu: zecimal :129
binar :10000001 = $2^7 + 2^0 = 128 + 1$
BCD :0001’0010’1001

Codul BCD

Codul BCD [Binary Coded Decimal] este unul dintre cele mai răspândite coduri cu semnificaţia **“**zecimal codificat în binar**”**, în care fiecare cifră zecimală este codificată în mod individual în echivalentul său binar pe patru biţi.

Orice cifră zecimală se poate reprezenta pe patru biţi, dar valorile reprezentabile pe patru biţi sunt în număr de $2^4 = 16$, deci vor rămâne 6 configuraţii neutilizate, de care trebuie să se ţină seama la efectuarea operaţiilor aritmetice.

În situaţia operaţiei de adunare trebuie să se adauge 6 ori de câte ori rezultatul este superior lui 9, iar pentru operaţia de scădere se va extrage 6 dacă rezultatul este negativ.

Exemplu:

zecimal	binar	BCD	
15+	01111+	0001’0101+	
18	10010	0001’1000	
---	-------	----------	
33	100001	0010’1101	> 9
	(= 33)	0110	+6

		0011’0011	(=33)

Operaţiile aritmetice sunt deci destul de complicate, dar operaţiile de intrare / ieşire sunt uşor de realizat deoarece fiecare entitate **BCD** este direct asociată unui caracter. Din aceste motive, codul **BCD** se găseşte în sistemele de calcul de gestiune, unde operaţiile aritmetice sunt mult mai puţin numeroase decât operaţiile de intrare / ieşire.

Codul **BCD** este un **cod ponderat** 8 – 4 – 2 – 1, cei patru biţi necesari pentru a codifica o cifră au o pondere corespunzătoare cu poziţia lor, respectiv $8 = 2^3$ pentru bitul cu numărul 3, $4 = 2^2$ pentru bitul cu numărul 2, $2 = 2^1$ pentru bitul cu numărul 1 şi $1 = 2^0$ pentru bitul cu numărul 0.

Codul "excedent – 3"

Codul **"**excedent - 3**"** nu este un cod ponderat, fiecare cifră zecimală este codificată separat în echivalentul său binar + 3.

Exemplu: 129_{10} = 0100**'**0101**'**1100 excedent - 3

Avantajul acestui cod faţă de codul BCD este acela că operaţiile aritmetice sunt mai simple.

De exemplu, complementarea faţă de 9 (similară în sistemul zecimal cu complementarea faţă de 1 în sistemul binar) este imediată: este suficientă complementarea fiecărui bit.

Codul "2 din 5"

Codul **"**2 din 5**"** este un cod neponderat în care fiecare cifră zecimală este codificată pe 5 biţi, dintre care numai 2 au valoarea 1.

Exemplu: 129_{10} = 00101**'**00110**'**11000 cod **"**2 din 5**"**

Avantajul acestui cod este acela de detectare (nu şi corectare) a unei erori sau a unui număr impar de erori.

Codul bicvintal

Codul bicvintal este un cod ponderat 50**'**43210 care permite detectarea erorilor. O cifră zecimală este codificată printr-un număr binar pe 7 biţi, având un singur bit egal cu 1 pe primele 2 poziţii din stânga şi un singur bit egal cu 1 pe 5 poziţii cele mai din dreapta.

Exemplu: 129_{10} = 0100010**'**0100100**'**1010000 bicvintal.

1.2 Coduri de erori

Informaţiile pot fi modificate involuntar în timpul transmisiei sau stocării lor în memorie. Trebuie deci utilizate anumite **coduri** care permit detectarea sau chiar corectarea erorilor datorate acestor modificări.

Aceste coduri se constituie pe un număr de biţi superior celui strict necesar pentru a codifica informaţia. Astfel, celor *m* biţi de date li se adaugă *k* biţi de control. Deci, în memorie vor fi stocaţi $n = m + k$ biţi. Asemenea configuraţii definesc **codurile redundante**.

Unele coduri nu permit decât detectarea erorilor (coduri **autoverificatoare**), altele permit detectarea şi corectarea uneia sau mai multor erori (coduri **autocorectoare**).

Controlul de paritate este codul autoverificator cel mai simplu. El se compune din $m + 1$ biţi: cei *m* biţi de informaţie la care se adaugă al $(m + 1)$ – lea bit numit **bit de paritate**. Valoarea sa este aleasă astfel ca numărul total de biţi egali cu 1, calculat pe $n + 1$ biţi să fie par (în cazul unei **parităţi pare**), sau impar (**paritate impară**).

Exemplu: Transmiterea de caractere codificate în **ASCII** pe 7 biţi plus un bit de paritate între un calculator şi un terminal.

A → 1 1000001
B → 1 1000010
E → 0 1000101
↑ bit de paritate impară

Dacă un bit este schimbat din eroare în timpul transferului, paritatea nu mai este verificată şi informaţia trebuie retransmisă, deoarece eroarea nu poate fi localizată pentru a putea fi corectată.

În general, controlul de paritate nu permite decât detectarea unui număr impar de erori, în cazul unui număr par efectele se pot anula.

Controlul de paritate nu poate fi utilizat decât pentru transmisii în care posibilitatea apariţiei erorilor este scăzută (de exemplu, în interiorul unui calculator sau între calculator şi perifericele sale).

Codul dublei parităţi

Codificare: considerăm exemplul unui cod ASCII (7 biţi);

- fiecare caracter este codificat pe o linie a unui tablou;
- un cod de paritate este efectuat pe fiecare linie (transversal);
- un cod de paritate este efectuat pe fiecare coloană (longitudinal);

Decodificare

- controlul transversal permite detectarea erorilor pe linie;
- controlul longitudinal permite detectarea erorilor pe coloană;

Dubla paritate permite corectarea unei erori, sau în anumite cazuri a unui număr impar de erori (ca de exemplu un număr impar de biţi eronaţi pe aceeaşi linie, coloană sau repartizaţi pe linii şi coloane diferite).

Exemplu: Se doreşte transmiterea mesajului 1968 cu paritate impară; se detectează o eroare la intersecţia primei linii cu coloana a 4-a.

	1	2	3	4	5	6	7	bit paritate	control longitudinal
1→	0	1	1	1	0	0	1	0	F
9→	0	1	1	1	0	0	1	1	A
6→	0	1	1	0	1	1	0	1	A
8→	0	1	1	1	0	0	0	0	A
bit paritate	1	1	1	1	0	0	1		
contr. transv.	A	A	A	F	A	A	A		

Principiul dublei parităţi este adesea utilizat în stocarea pe bandă magnetică a informaţiilor. Astfel, pentru o bandă cu *n* piste:

- fiecare caracter este stocat transversal pe *n –1* piste;
- un bit de paritate transversal este stocat pe a *n*-a pistă;
- un bloc de caractere suportă un control longitudinal de paritate.

Codul lui Hamming

Codul lui Hamming este un cod autocorector bazat pe teste de paritate. Versiunea cea mai simplă permite corectarea unui bit eronat. Celor *m* biţi de informaţie li se adaugă *k* biţi de control al parităţii. Vom avea $n = m+k$ biţi necesari pentru transmiterea informaţiei.

Deoarece trebuie indicate $n + 1$ posibilităţi de eroare (inclusiv absenţa erorii) prin cei *k* biţi de control, trebuie ca $2^k \geq n + 1$. Cele 2^k posibilităţi de de codificare pe *k* biţi servesc la determinarea poziţiei erorii, apoi se poate corecta bitul eronat.

Tabelul următor permite determinarea lui *k* când se cunoaşte *n*:

m	0	0	1	1	2	3	4	4	5	6	7	8	9	10	...	120
k	1	2	2	3	3	3	3	4	4	4	4	4	4	4	...	8
n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...	128

De obicei se ia $n = 2^k - 1$ în loc de $n < 2^k - 1$.

Dacă se numerotează biţii de la dreapta spre stânga pornind de la 1, biţii de control (sau de paritate) sunt plasaţi pe poziţia puterilor lui 2 (biţii cu numărul 1, 2, 4, 8, 16, ...). Fiecare bit de control efectuează control de paritate (pară sau impară) asupra unui anumit număr de biţi de date. Se determină astfel cei n biţi de transmis sau de stocat.

Exemplu: Dacă m = 4 se poate construi un cod Hamming (CH) pe 7 biţi (n = 7), adăugând 3 biţi de control (k = 3).

7	6	5	4	3	2	1
m_4	m_3	m_2	k_3	m_1	k_2	k_1

Cei trei biţi de control sunt plasaţi pe poziţia puterilor lui 2:

$k_1 \rightarrow 1$; $k_2 \rightarrow 2$; $k_3 \rightarrow 4$.

Vom vedea acum, pentru fiecare bit al mesajului care sunt biţii de control care permit verificarea parităţii sale.

$7 = (0111)_2 = 4 + 2 + 1 \rightarrow$ 7 este controlat de k_3, k_2, k_1;
$6 = (0110)_2 = 4 + 2 \rightarrow$ 6 este controlat de k_3, k_2;
$5 = (0101)_2 = 4 + 1 \rightarrow$ 5 este controlat de k_3, k_1;
$4 = (0100)_2 = 4 \rightarrow$ 4 este controlat de k_3;
$3 = (0011)_2 = 2 + 1 \rightarrow$ 3 este controlat de k_2, k_1;
$2 = (0010)_2 = 2 \rightarrow$ 2 este controlat de k_2;
$1 = (0001)_2 = 1 \rightarrow$ 1 este controlat de k_1;

Problema se pune şi invers: care sunt poziţiile binare controlate de către fiecare cod? Răspunsul este următorul:

k_1 controlează biţii cu numerele 1, 3, 5, 7;
k_2 controlează biţii cu numerele 2, 3, 6, 7;
k_3 controlează biţii cu numerele 4, 5, 6, 7.

Când se recepţionează informaţia, se efectuează controlul de paritate. Pentru fiecare bit de control se compară valoarea transmisă cu cea recalculată. Dacă cele două valori sunt identice, se atribuie valoarea 0 unei variabile binare A_i asociată bitului de control k_i, altfel, A_i primeşte valoarea 1.

Valoarea zecimală a configuraţiei binare formată din variabilele A_k, A_{k-1}, ..., A_1 furnizează poziţia bitului eronat, care se poate corecta.

Presupunem că: pentru k_1, $A_1 = 1$, pentru k_2, $A_2 = 1$, iar pentru k_3, $A_3 = 0$. Eroarea se găseşte în poziţia $(A_3A_2A_1)_2 = (011)_2 = 3$.

Într-adevăr, k_1 poate detecta o eroare în poziţiile 1, 3, 5, 7, k_2 poate detecta o eroare pe poziţiile 2, 3, 6, 7, iar k_3 pe poziţiile 4, 5, 6, 7. O eroare detectată de k_1 şi k_2 nu şi de k_3 nu poate proveni decât din bitul 3.

Exemple:

$(A_3A_2A_1)_2 = (000)_2 \rightarrow$ indică absenţa unei erori;
$(A_3A_2A_1)_2 = (001)_2 \rightarrow$ indică eroare pe bitul 1;
$(A_3A_2A_1)_2 = (110)_2 \rightarrow$ indică eroare pe bitul 6.

Exemplu de recepţionare a unui mesaj: $(1011100)_2$. Dacă s-a utilizat un CH cu paritate pară, să se reconstituie mesajul iniţial (n = 7, k = 3, m = 4).

număr	7	6	5	4	3	2	1
tip	m_4	m_3	m_2	k_3	m_1	k_2	k_1
valoare	1	0	1	1	1	0	0

$k_1 = 0$ controlează poziţiile 1, 3, 5, 7, nu se verifică, deci $A_1 = 1$;

$k_2 = 0$ controlează poziţiile 2, 3, 6, 7, se verifică, deci $A_2 = 0$;

$k_3 = 0$ controlează poziţiile 4, 5, 6, 7, nu se verifică, deci $A_3 = 1$;

Adresa erorii $(A_3A_2A_1)_2 = (101)_2 = 5$. Bitul numărul 5, care este egal cu 1 este eronat. Mesajul iniţial corectat şi fără biţii de control este: $(1001)_2$.

Calculul simplificat al codului lui Hamming (CHS)

Metoda Hamming poate simplifica calculul biţilor de control, astfel:

- se transformă în binar poziţiile din mesaj care conţin valoarea 1;
- se însumează modulo 2, astfel:
  - pentru paritate pară: un număr par de $1 \rightarrow 0$, un număr impar de $1 \rightarrow 1$ (sumă modulo 2 directă);
  - pentru paritate impară: un număr par de $1 \rightarrow 1$, un număr impar de $1 \rightarrow 0$ (sumă modulo 2 inversată).

Exemplu: Să codificăm mesajul $(10101011001)_2$ cu paritate pară: m = 11, deci k = 4 şi n = 15.

număr	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
tip	m_{11}	m_{10}	m_9	m_8	m_7	m_6	m_5	k_4	m_4	m_3	m_2	k_3	m_1	k_2	k_1
valoare	1	0	1	0	1	0	1	?	1	0	0	?	1	?	?

Biţii cu valoarea 1 se găsesc pe poziţiile 15, 13, 11, 9, 7, 3, deci:

```
15 = 1  1  1  1
13 = 1  1  0  1
11 = 1  0  1  1
 9 = 1  0  0  1
 7 = 0  1  1  1
 3 = 0  0  1  1
     -----------
     0  1  0  0      → biţi de paritate
     k4 k3 k2 k1
```

Mesajul codificat este deci $(101010101001100)_2$.

Exemplu de recepţie a unui mesaj:

S-a primit mesajul următor: $(101000101001100)_2$, codificat cu paritate impară. Biţii cu valoarea 1 se găsesc pe poziţiile 15, 13, 9, 7, 4, 3.

```
15 = 1  1  1  1
13 = 1  1  0  1
 9 = 1  0  0  1
 7 = 0  1  1  1
 4 = 0  1  0  0
 3 = 0  0  1  1
     ----------   → adunare modulo 2 inversată.
     0  1  0  0
     A4A3A2A1   → eroare pe poziţia 4.
```

După corectarea erorii şi eliminarea codurilor de control, mesajul iniţial este: $(10100011001)_2$.

Codul lui Hamming şi erorile grupate

Metoda lui Hamming poate corecta în general doar un bit eronat, dar se poate utiliza pentru detectarea şi corectarea erorilor multiple pe o secvenţă de biţi aranjând mesajul sub formă matricială, codificând pe linie după metoda HC şi transmiţând mesajul pe coloane.

De exemplu, pentru transmiterea mesajului **"**Hamming**"** se codifică pe o linie fiecare caracter în cod ASCII, completând cu biţii de control după metoda CH.

ASCII	Cod Hamming (pentru fiecare literă)										
	11	10	9	8	7	6	5	4	3	2	1 (număr)
H → 1001000	1	0	0	1	1	0	0	1	0	0	0
a → 1100001	1	1	0	0	0	0	0	0	1	1	0
m → 1101101	1	1	0	0	1	1	0	0	1	1	1
m → 1101101	1	1	0	0	1	1	0	0	1	1	1
i → 1101001	1	1	0	0	1	0	0	1	1	0	1
n → 1101110	1	1	0	0	1	1	1	1	0	0	1
g → 1100111	1	1	0	0	0	1	1	0	1	0	1

Dacă se produc erori grupate, pentru o secvenţă de biţi suficient de scurtă, (≤ 7 pentru acest caz) atunci, efectuând transmisia pe coloană vom avea un singur bit eronat pe linie, pe care-l putem corecta datorită biţilor de control adăugaţi potrivit metodei lui Hamming.

În ultimii ani, codurile autocorectoare sunt din ce în ce mai utilizate pentru a asigura integritatea informaţiilor stocate în memorie.

Un cod autocorector permite creşterea considerabilă a timpului mediu între două defecţiuni care pot să apară: erorile nu apar decât atunci când numărul lor depăşeşte capacitatea de corectare a codului respectiv. Erorile care nu sunt corectate, cel puţin se pot detecta.

Detectarea erorilor grupate

În comunicaţiile la distanţă, erorile sunt mult mai frecvente decât în interiorul calculatorului. Erorile consecutive pot fi extinse adesea la un bloc întreg de biţi de informaţie.

Se vor utiliza în acest sens coduri care permit detectarea erorilor grupate, corectarea acestora fiind adesea prea costisitoare.

Metoda codurilor polinomiale (CRC)

CRC [Cyclic Redondant Coding] este metoda cea mai folosită pentru detectarea erorilor grupate. Înaintea transmiterii, informaţiei i se adaugă biţi

de control, iar pe baza acestora, dacă la recepţionarea mesajului se detectează erori, atunci acesta trebuie retransmis.

O informaţie pe n biţi poate fi considerată ca lista coeficienţilor binari ai unui polinom cu n termeni, deci de grad n-1.

Exemplu: 110001→ $x^5 + x^4 + 1$;

Pentru a calcula biţii de control se va efectua un anumit număr de operaţii cu aceste polinoame cu coeficienţi binari. Operaţiile se vor efectua modulo 2, adunarea şi scăderea nu va ţine seama de cifra de transport, deci toate operaţiile de adunare şi scădere sunt identice cu operaţia logică XOR.

Pentru generarea şi verificarea biţilor de control atât sursa cât şi ddestinaţia mesajului utilizează un polinom generator G (x).

Dacă M (x) este polinomul corespunzător mesajului iniţial (de transmis), iar r este gradul polinomului generator G (x), atunci algoritmul de construire şi verificare a codurilor care se incorporează în mesajul de transmis este următorul:

1) se înmulţeşte M (x) cu x^r (se adaugă *r* zerouri la sfârşitul mesajului iniţial);
2) se efectuează împărţirea modulo 2: $M(x)*x^r/G(x)=Q(x)+R(x)/G(x)$;
   Câtul Q (x) se ignoră, iar restul R (x) conţine *r* biţi. Se efectuează scăderea modulo 2: M (x) * x^r – R (x) = T (x), iar T (x) este polinomul care reprezintă mesajul de transmis. Polinomul ciclic T (x) = Q (x) * G (x) este un multiplu al polinomului generator.
4) La recepţionarea mesajului se efectuează împărţirea T (x) /G (x):
   a) dacă restul = 0 nu sunt erori de transmisie;
   b) altfel, s-au produs erori, deci mesajul trebuie retransmis.

Exemplu de transmitere a unui mesaj. Se doreşte transmiterea mesajului 101101 (6 biţi) → M (x) =$x^5 + x^3 + x^2 + 1$.

Polinomul generator este: 1011→ G (x) = $x^3 + x + 1$ de grad r = 3.

1) Efectuăm înmulţirea: M (x) x^r = 101101000 (se adaugă r = 3 zerouri la M (x).
2) Realizăm împărţirea modulo 2: $M(x)*x^r/G(x)$:

```
1 0 1 1 0 1 0 0 0 | 1 0 1 1
 1 0 1 1            ---------
 --------             1 0 0 0 0 1 → câtul Q(x)
0 0 0 0 0 1 0 0 0
      1 0 1 1
      --------
      0 0 1 1 → R (x) = 0 1 1
```

3) Câtul Q (x) este ignorat. Pentru a realiza diferenţa modulo 2 M (x) *x^r – R (x) este suficientă adăugarea celor r biţi din R (x) la

sfârşitul mesajului M (x) → mesajul de transmis este T (x) = 101101011.

Exemplu de recepţionare a unui mesaj. S-a primit mesajul următor: 11010101. G (x) = 1011 (4 biţi) → G (x) = $x^3 + x + 1$ de grad r = 3.

4) Se efectuează împărţirea T (x) / G (x).

```
1 1 0 1 0 1 0 1 | 1 0 1 1
1 0 1 1           ---------
--------            1 1 1 1 0
0 1 1 0 0
  1 0 1 0
  --------
        0 1 1 1 1
        1 0 1 1
        --------
        0 1 0 0 0
          1 0 1 1
          --------
          0 0 1 1 1 →R (x) = 1 1 1
```

R (x) ≠ 0, s-au detectat erori de transmisie, mesajul se retransmite.

Cele mai utilizate polinoame generatoare G (x) sunt:

- CRC – 12 = $x^{12} + x^{11} + x^3 + x^2 + x + 1$;
- CRC – 16 = $x^{16} + x^{15} + x^2 + 1$;
- CRC – CCITT = $x^{16} + x^{12} + x^5 + 1$.

1.3 Elemente de logică numerică

Logica propoziţională este o algebră al cărei obiectiv iniţial este modelarea raţionamentului.

Mai recent această algebră şi-a demonstrat utilitatea ca instrument de concepţie (concepţia circuitelor calculatorului).

O a treia utilizare a logicii constă în a servi ca model de calcul pentru limbajele de programare (**Prolog**).

Logica propoziţională este un model matematic care ne permite să raţionăm asupra naturii adevărate sau false a **expresiilor logice**.

O **propoziţie** este un enunţ care poate lua una din cele două valori de adevăr: **adevărat** sau **fals**. Simbolurile care pot reprezenta o propoziţie se numesc **variabile propoziţionale**.

Expresii logice

O primă mulţime de expresii logice se defineşte recursiv astfel:

- sunt operanzi atomici:

- variabilele propoziţionale;
- constantele logice **true** şi **false**;

- Orice operand atomic este expresie logică;
- Dacă E şi F sunt expresii logice atunci **E and F** este expresie logică;
- Dacă E şi F sunt expresii logice atunci **E or F** este expresie logică;
- Dacă E este expresie logică atunci **not E** este expresie logică;

Prezentăm definiţia operatorilor logici **and**, **or** şi **not**.

and	0	1
0	0	0
1	0	1

or	0	1
0	0	1
1	1	1

not	
0	1
1	1

Funcţii booleene

"Semnificaţia**"** unei expresii logice poate fi descrisă formal ca o funcţie care dă o valoare adevărat sau fals pentru expresia întreagă pornind de la valoarea argumentelor, numită funcţie logică sau booleană**.**

Tabele de adevăr

O funcţie booleană poate fi reprezentată în practică printr-o tabelă de adevăr ale cărei linii corespund tuturor combinaţiilor de valori de adevăr pentru argumente. Există o coloană pentru fiecare argument şi una pentru valoarea funcţiei.

Figura următoare prezintă tabelele de adevăr pentru operaţiile logice **and**, **or**, **not**, **xor**.

p	q	p **and** q
0	0	0
0	1	0
1	0	0
1	1	1

p	q	p **or** q
0	0	0
0	1	1
1	0	1
1	1	1

p	**not** p
0	1
1	0

p	q	p **xor** q
0	0	1
0	1	0
1	0	0
1	1	1

Tabela de adevăr a unei funcţii cu k argumente posedă 2^k linii. Fiecare linie asignează pentru funcţie valoarea 0 sau 1, deci există 2^{2k} funcţii.

Operatori logici suplimentari

- **implicaţia** $\rightarrow$ **"** dacă p este adevărat atunci q este adevărat**"**;
- **echivalenţa** $\equiv$ **"** dacă şi numai dacă**"**;
- operatorul **nonand "** not (p **and** q)**"**, notat p **nand** q;
- operatorul **nonor "** not (p **or** q)**"**, notat p **nor** q.

Figura următoare prezintă tabelele de adevăr pentru $\rightarrow$, $\equiv$, **nand**, **nor**.

p	q	$p \to q$
0	0	1
0	1	1
1	0	0
1	1	1

p	q	$p \equiv q$
0	0	1
0	1	0
1	0	0
1	1	1

p	q	p **nand** q
0	0	1
0	1	1
1	0	1
1	1	0

p	q	p **nor** q
0	0	1
0	1	0
1	0	0
1	1	0

Asociativitatea şi precedenţa operatorilor logici

Operatorii logici **and** şi **or** sunt asociativi şi comutativi şi vor fi grupaţi de la stânga la dreapta. Ceilalţi operatori nu sunt asociativi.

Precedenţa operatorilor logici: **not, nand, nor, and, or, $\to$, $\equiv$.**

Funcţii booleene ale expresiilor logice

Vom construi expresii logice pornind de la tabelele de adevăr. Deşi se pot defini o infinitate de expresii logice, de obicei, se va încerca pe cât posibil să se găsească cea mai simplă expresie logică.

Notaţii prescurtate

and se reprezintă prin juxtapunerea operanzilor;

or se reprezintă prin +;

not se reprezintă prin $\neg$ sau prin bararea variabilei.

Construcţia unei expresii logice din tabela de adevăr

Forma normală disjunctivă este o sumă logică de **mintermi** pentru care funcţia ia valoarea logică adevărat (1). Un **minterm** este un produs logic de literale (variabile propoziţionale) ale unei linii, astfel: dacă p are valoarea 0 în coloana k, se utilizează $\overline{p}$, altfel se utilizează *p*.

Forma normală conjunctivă este un produs logic de **maxtermi** pentru care funcţia ia valoarea 0. Un **maxterm** este o sumă logică de literale ale unei linii, astfel: dacă p ia valoarea 0 se utilizează p, altfel $\overline{p}$.

Legi algebrice pentru expresii logice

Legi ale echivalenţei

1. Reflexivitate: $(p \equiv p)$;
2. Comutativitate: $(p \equiv q) \equiv (q \equiv p)$;
3. Tranzitivitate: $(p \equiv q)$ *and* $(q \equiv r) \to (p \equiv r)$;
4. Echivalenţa negaţiilor: $(p \equiv q) \to (\overline{p} \equiv \overline{q})$;

Legi similare aritmeticii

5. Comutativitate **and**: $p \cdot q \equiv q \cdot p$;

6. Asociativitate **and**: $p \cdot (q \cdot r) \equiv (p \cdot q) \cdot r$;
7. Comutativitate **or**: $(p + q) \equiv (q + p)$;
8. Asociativitate **or**: $(p + (q + r)) \equiv ((p + q) + r)$;
9. Distributivitate **and** faţă de **or**: $p \cdot (q + r) \equiv p \cdot q + p \cdot r$;
10. 1 (true) este identitate pentru **and**: $p \cdot 1 \equiv p$;
11. 0 (false) este identitate pentru **or**: $(p + 0) \equiv p$;
12. 0 este adsorbant pentru **and**: $p \cdot 0 \equiv 0$;
13. Eliminare duble negaţii: $\overline{\overline{p}} \equiv p$.

Diferenţe faţă de legile aritmeticii

14. Distributivitate **or** faţă de **and**: $(p + q \cdot r) \equiv (p + q) \cdot (p + r)$;
15. 1 este adsorbant pentru **or**: $1 + p \equiv 1$;
16. Idempotenţa operatorului **and**: $p \cdot p \equiv p$;
17. Idempotenţa operatorului **or**: $p + p \equiv p$;
18. Subsumarea:
    a) $(p + p \cdot q) \equiv p$;
    b) $p \cdot (p + q) \equiv p$.
19. Eliminarea anumitor negaţii:
    a) $p \cdot (\overline{p} + q) \equiv p \cdot q$;
    b) $(p + \overline{p} \cdot q) \equiv p + q$.
20. Legile lui de Morgan:
    a) $\overline{p \cdot q} \equiv \overline{p} + \overline{q}$;
    b) $\overline{p + q} \equiv \overline{p} \cdot \overline{q}$;
    c) $\overline{p_1 \cdot p_2 \cdot \ldots \cdot p_n} \equiv \overline{p}_1 + \overline{p}_2 + \ldots + \overline{p}_n$;
    d) $\overline{p_1 + p_2 + \ldots + p_n} \equiv \overline{p}_1 \cdot \overline{p}_2 \cdot \ldots \cdot \overline{p}_n$.

Legi ale implicaţiei

21. $(p \to q)\ and\ (q \to p) \equiv (p \equiv q)$;
22. $(p \equiv q) \to (p \to q)$;
23. $(p \to q)\ and\ (q \to r) \to (p \to r)$;
24. $(p \to q) \equiv (\overline{p} + q)$.

Tautologii şi metode de demonstraţie

25. Legea terţului exclus: $(p + \bar{p}) \equiv 1$;
26. Analiza de caz: $(p \to q)\, and\, (\bar{p} \to q) \equiv q$;
27. Contrara reciprocei: $(p \to q) \equiv (\bar{q} \to \bar{p})$;
28. Reducere la absurd: $(\bar{p} \to 0) \equiv p$;
29. Demonstraţie prin reducere: $(p \equiv 1) \equiv p$;

Exemple:

1) Funcţii de o variabilă *a*:

a	z_0	z_1	z_2	z_3
0	0	0	1	1
1	0	1	0	1

$z_0 = 0$ constantă;
$z_1 = a$ identitate;
$z_2 = \neg a$ negaţie;
$z_3 = 1$ constantă;

2) Funcţii logice de 2 variabile a şi b:

00	01	10	11	ab
0	0	0	0	F0 = 0
0	0	0	1	F1 = $a \cdot b$
0	0	1	0	F2 = $a \cdot \bar{b}$
0	0	1	1	F3 = a
0	1	0	0	F4 = $\bar{a} \cdot b$
0	1	0	1	F5 = b
0	1	1	0	F6 = $a \oplus b$
0	1	1	1	F7 = $a + b$
1	0	0	0	F8 = $\overline{a + b} = \bar{a} \cdot \bar{b}$
1	0	0	1	F9 = $\overline{a \oplus b}$
1	0	1	0	F10 = $\bar{b}$
1	0	1	1	F11 = $a + \bar{b}$
1	1	0	0	F12 = $\bar{a}$
1	1	0	1	F13 = $\bar{a} + b$
1	1	1	0	F14 = $\neg(ab) = \neg a + \neg b$
1	1	1	1	F15 = 1

Tabelele Karnaugh (TK)

Tabelele sau diagramele lui Karnaugh permit simplificarea funcţiilor logice. Metoda se bazează pe inspectarea vizuală a tabelelor judicios construite (metoda este utilă cu un număr de variabile ≤ 6).

TK se poate considera ca o transformare a tabelei de adevăr.

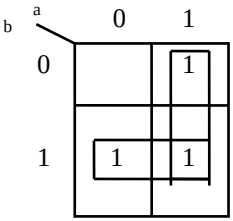
TK cu 2 variabile

Cele 4 căsuţe ale TK corespund celor 4 linii ale tabelei de adevăr:

- fiecare variabilă logică completează o linie sau o coloană;
- un produs de 2 variabile completează o căsuţă;

Pentru a completa TK pornind de la tabela de adevăr se atribuie valoarea 1 căsuţelor corespunzătoare stărilor din intrare în care funcţia are valoarea 1. Metoda de simplificare constă din a încadra căsuţele ocupate, adiacente pe aceeaşi linie sau coloană (suprapunerile sunt permise). Figura următoare prezintă o TK cu 2 variabile.

a	b	z
0	0	0
0	1	1
1	0	1
1	1	1



În urma reducerii avem vom obţine z = a+b.

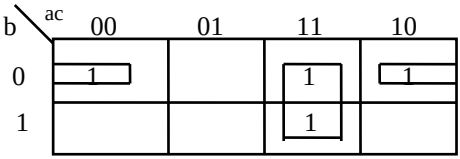
TK cu 3 variabile

Tabela de adevăr a funcţiei logice de 3 variabile se transformă într-o TK cu două dimensiuni, grupând două variabile pe linie sau coloană. Trecerea de la o linie (coloană) la alta diferă printr-o singură variabilă (se consideră tabela ca înfăşurătoarea unui cilindru.

Pentru construirea TK cu 3 variabile:

- fiecare variabilă completează un bloc de 4 căsuţe;
- un produs logic de două variabile completează un bloc de 2 căsuţe;
- un produs logic de 3 variabile completează o căsuţă.

Exemplu: f (a, b, c) = ¬a¬b¬c+a¬bc+a¬b¬c+abc;



În urma reducerii se obţine z = ac+¬b¬c.

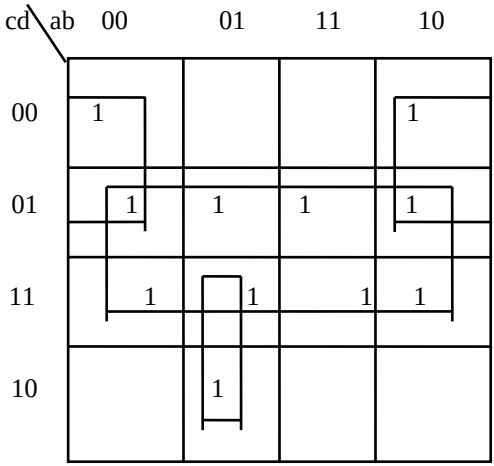
TK cu 4 variabile

Se construieşte TK, precum înfăşurătoarea unui cilindru, atât orizontal cât şi vertical, astfel:

- fiecare variabilă completează un bloc de 8 căsuţe;
- un produs logic de 2 variabile completează un bloc de 4 căsuţe;
- un produs logic de 3 variabile completează un bloc de 2 căsuţe;
- un produs logic de 4 variabile completează o căsuţă.

Exemplu:

z(a,b,c,d)=¬a¬b¬c¬d+¬a¬b¬cd+¬a¬bcd+¬ab¬cd+ab¬cd+
a¬b¬cd+a¬b¬c¬d+¬abcd+abcd+a¬bcd+¬abc¬d.



Expresia simplificată este z = d+¬b¬c+¬abc.

În general, metoda de simplificare a unei funcţii de 4 variabile prin TK este următoarea:

- încadrarea căsuţelor cu 1 care nu sunt adiacente altora cu 1 şi deci nu pot forma blocuri de 2 căsuţe;
- încadrarea căsuţelor care pot forma grupe de 2 căsuţe dar nu pot forma blocuri de 4 căsuţe;
- încadrarea căsuţelor care pot forma grupe de 4 căsuţe dar nu pot forma blocuri de 8 căsuţe;
- încadrarea grupelor de 8 căsuţe adiacente;

Pentru reducerea funcţiilor logice cu mai mult de 4 variabile trebuie create mai multe tabele Karnaugh.