## Conceptul de performanță

Pentru a da o dimensiune numerică performanței $(P)$ și a descrie dependența ei descrescătoare în raport cu timpul de răspuns $(T)$, adoptăm formula:

$$P = \frac{1}{T} \quad (1)$$

Deja putem să comparăm performanțele sau să calculăm un raport al performanțelor unui program cu niște date fixate, pe două mașini $X$ și $Y$:

$$P_X > P_Y \overset{\text{def}}{\Longleftrightarrow} T_X < T_Y, \quad \frac{P_X}{P_Y} = n \overset{\text{def}}{\Longleftrightarrow} \frac{T_Y}{T_X} = n \quad (2)$$

(am notat cu $P_X$, $P_Y$, $T_X$, $T_Y$, performanțele respectiv timpii de răspuns, pe mașinile $X$ și $Y$).

## Măsurarea performanței

Timpul (CPU) se poate măsura în:

- **secunde** (s)
- **cicli de ceas** (c)

Legătura dintre cele două este dată de peroada de ceas (durata ciclului) $D$ (măsurată în secunde (s), nanosecunde (ns), etc.) sau frecvența $F$ (măsurată în herzi (Hz), megaherzi (MHz), etc.), după formula:

$$F = \frac{1}{D}$$

Când măsurăm în cicli nu mai contează fizica circuitelor (durata ciclului) ci doar împărțirea programului în operații mașina (logica programului) și a operațiilor mașină în ciclii (logica circuitelor) - avem astfel posibilitatea să comparăm d.p.v. logic o gamă mai largă de mașini.

## Măsurarea performanței

O mărime care ne dă indicații asupra performanței este **numărul total de cicli executați** $C$.

El depinde de program, de date și de împărțirea instrucțiunilor în cicli (am presupus programul scris în limbaj mașină); nu depinde de durata ciclului.
Performanța este cu atât mai mare cu cât $C$ este mai mic.

Dacă presupunem programul scris într-un limbaj de nivel înalt, contează și împărțirea codului sursă în instrucțiuni mașină - deci intervine și performanța compilatorului.

Atunci putem calcula timpul CPU $T$ după formula:

$$T = C \times D = \frac{C}{F} \quad (3)$$

Această formulă arată că putem reduce timpul $T$ (deci mări performanța) reducând numărul de cicli executați $C$ sau/și durata ciclului $D$.
Este dificil însă de redus simultan ambii factori - de obicei reducera unuia atrage creșterea celuilalt.

## Măsurarea performanței

Alte două mărimi care ne dau indicații asupra performanței sunt **numărul total de instrucțiuni executate** $I$ și **numărul mediu ce cicli per instrucțiune executați** (engl: **cycles per instruction**) $CPI$, calculat după formula:

$$CPI = \frac{C}{I} \quad (4)$$

$I$ depinde de program și de date; nu depinde de împărțirea instrucțiunilor în cicli și nici de durata ciclului.
Performanța este cu atât mai mare cu cât $I$ este mai mic.
$CPI$ depinde de program, de date și de împărțirea instrucțiunilor în cicli; nu depinde de durata ciclului.
Performanța este cu atât mai mare cu cât $CPI$ este mai mic.

## Măsurarea performanţei

Din formulele (3) şi (4) obţinem **ecuaţia elementară a performanţei**:

$$T = I \times CPI \times D = \frac{I \times CPI}{F} \quad (5)$$

Ea leagă cei trei **factori cheie** ai performanţei: numărul de instrucţiuni executate $I$, numărul mediu de cicli pe instrucţiune executaţi $CPI$ şi durata ciclului $D$ (sau frecvenţa $F$).

Există interdependenţe între factori; de exemplu, scăderea $CPI$ poate atrage creşterea $D$ (deoarece la fiecare ciclu se vor executa operaţii mai multe şi/sau mai complexe).

Aceşti factori trebuie consideraţi simultan atunci când analizăm un sistem, altfel putem trage concluzii eronate privind performanţa - a se vedea următorul exerciţiu rezolvat.

## Măsurarea performanţei

O altă mărime care ne dă indicaţii asupra performanţei este **frecvenţa de executare a instrucţiunilor**:

$$FI = \frac{I}{T} \quad (7)$$

Întrucât de obicei valorile lui $FI$ (măsurate în instrucţiuni pe secundă) sunt mari, se obişnuieşte introducerea încă unui factor $10^6$ la numitor, obţinându-se mărimea:

$$MIPS = \frac{I}{T \times 10^6} \quad (8)$$

măsurată în milioane de instrucţiuni pe secundă (**millions instructions per second**).

$FI$ şi $MIPS$ depind de program, de date, de împărţirea instrucţiunilor în cicli şi de durata ciclului.
Performanţa este cu atât mai mare cu cât $FI$ şi $MIPS$ sunt mai mari.

## baza 10 $\to$ baza b; numere întregi

Trecerea din baza 10 într-o bază $b$:

- Cazul numerelor întregi:

Fie $x \in \mathbb{Z}$ şi $b \in \mathbb{N}$, $b \geq 2$, baza.
Considerăm $x$ şi $b$ scrise în baza 10; facem calculele în baza 10.

Regulă:
- împărţim cu rest pe $|x|$ la $b$, apoi câtul la $b$, etc., până obţinem câtul 0;
- luăm resturile în ordine inversă şi le înlocuim cu cifre ale bazei $b$;
- dacă $x < 0$, punem în faţă "$-$".

Observaţie: dacă am continua procedeul după ce am obţinut câtul 0, am obţine noi câturi şi resturi 0, care ar genera cifre 0 în stânga reprezentării, iar acestea nu schimbă semantica reprezentării (nu afectează valoarea reprezentată).

## baza 10 $\to$ baza b; numere întregi

Calculele se pot redacta astfel (`C0`, `C1`, ..., `Cn` sunt cifre ale bazei $b$):

```
 x     |   b
 .     |--------
 .     | cat0 |   b
 .          .  |--------
rest0       .  |  cat1                                             ------------
 C0         .             .            Obtinem reprezentarea:   Cn ... C1 C0
         rest1               .
          C1                    .
                      cat(n-1) |   b
                       .       |--------
                       .       |  0
                       .
                      restn
                       Cn
     <======= citim
```

## baza 10 $\to$ baza b; numere întregi

Exemplu: $(4235)_{16} = ?$

```
4235   |   16
32     |-----
---    |   264 |   16
103        16  |-----
 96        --  |   16   |   16
 --        104     16   |-----
==75        96     --   |   1   |   16
  64        --      0       0   |-----
  --       ==8      0       -   |   0
  11         8              1
   B                        1

    <=======
```

Deci: $(4235)_{16} = \overline{108B}$

## baza 10 $\to$ baza b; numere întregi

Dacă $b$ are (în baza 10) doar o cifră, atunci `catk` și `restk` se pot calcula mintal, iar calculele se pot redacta mai simplu:

```
 x         | rest0  C0     /\
cat0       | rest1  C1     ||
cat1       |               ||
           .               ||                  ------------
           .               ||         Obtinem reprezentarea:   Cn ... C1 C0
           .               ||
cat(n-1)   | restn  Cn     || citim
 0         |
```

## baza 10 $\to$ baza b; numere întregi

Exemplu: $(105)_2 = ?$

```
105  |  1    /\
 52  |  0    ||
 26  |  0    ||
 13  |  1    ||
  6  |  0    ||
  3  |  1    ||
  1  |  1    ||
  0  |
```

(pentru $b < 10$ cifrele bazei $b$ sunt aceleași ca în baza 10 și nu este nevoie să le mai scriem o dată în dreapta).

Deci: $(105)_2 = \overline{1101001}$

## baza 10 $\to$ baza b; numere reale

• Cazul numerelor reale (fracționare):

Fie $x \in \mathbb{R}$ și $b \in \mathbb{N}$, $b \geq 2$, baza.
Considerăm $x$ și $b$ scrise în baza 10; facem calculele în baza 10.

Regulă:
- partea întreagă a lui $|x|$ se reprezintă ca la numere întregi;
- partea fracționară a lui $|x|$ se înmulțește cu $b$ și se ia partea întreagă, apoi partea fracționară rămasă se înmulțește cu $b$ și se ia partea întreagă, etc., până obținem o parte fracționară 0 sau care a mai fost întâlnită (dacă $x \in \mathbb{R} \setminus \mathbb{Q}$ procedeul continuă la infinit, nu este algoritm);
- luăm părțile întregi obținute în ordine directă și le înlocuim cu cifre ale bazei $b$;
- dacă $x < 0$, punem în față "$-$".

## baza 10 $\rightarrow$ baza b; numere reale

Observații:
- regula nu o contrazice pe cea pentru numere întregi ci o completează (afirmă ceva și pentru partea fracționară, care la numere întregi lipsește); cifrele produse de cele două părți ale regulii (pentru partea întreagă, respectiv fracționară) nu se suprapun, deoarece se află în părți diferite ale virgulei;
- cu partea întreagă facem împărțiri succesive la $b$ și obținem cifre de la virgulă spre stânga; cu partea fracționară facem înmulțiri succesive cu $b$ și obținem cifre de la virgulă spre dreapta; per total, cifrele se obțin de la virgulă spre extremități;
- fiecare din părțile fracționare obținute reprezintă un număr real din intervalul $[0, 1)$; când o înmulțim cu $b$, obținem un număr real din intervalul $[0, b)$; luând apoi partea întreagă, obținem un număr întreg din mulțimea $\{0, \ldots, b-1\}$, care corespunde unei cifre în baza $b$ - aceasta este următoarea zecimală (în baza $b$); cu partea fracționară rămasă continuăm procedeul;

## baza 10 $\rightarrow$ baza b; numere reale

- până când se aplică procedeul:

• dacă la un moment dat întâlnim o parte fracționară 0, înmulțind cu $b$ obținem o nouă parte întreagă (i.e. o nouă zecimală) 0 și o nouă parte fracționară 0, etc., deci continuând vom obține noi zecimale 0 nesemnificative; de aceea ne putem opri și conchidem că am obținut o fracție zecimală finită;

• dacă la un moment dat obținem o parte fracționară care a mai fost întâlnită, continuând procedeul vom obține aceeași succesiune de zecimale ca după prima întâlnire; de aceea ne putem opri și conchidem că am obținut o fracție zecimală periodică (simplă sau mixtă); perioada este succesiunea de zecimale generate dupa prima întâlnire;

## baza 10 $\rightarrow$ baza b; numere reale

• întrucât părțile fracționare generate de procedeu reprezintă numere reale din intervalul $[0, 1)$ și există o infinitate de asemenea numere distincte, este posibil ca prin continuarea procedeului să nu obținem nici o dată o parte fracționară 0 sau care a mai fost întâlnită; atunci procedeul nu se termină (nu este algoritm); acest lucru se întâmplă însă doar în cazul numerelor iraționale;

mai exact: dacă $x \in \mathbb{Q}$ atunci în orice bază $b$ reprezentarea lui $x$ va fi o fracție zecimală finită, periodică simplă sau periodică mixtă; dacă $x \in \mathbb{R} \setminus \mathbb{Q}$ atunci în orice baza $b$ reprezentarea lui $x$ va fi o fracție zecimală infinită neperiodică;

deci, dacă într-o aplicație se va cere "reprezentați 7.8 în baza 2", vom ști că procedeul se termină (și obținem o fracție zecimală finită, periodică simplă sau periodică mixtă), deoarece în baza sursă (adică 10) numărul s-a reprezentat printr-o fracție zecimală finită, deci este rațional; nu vom cere "reprezentați $\sqrt{2}$ în baza 2";

atenție însă că un același număr rațional poate avea într-o bază o reprezentare prin fracție zecimală finită, în alta prin fracție zecimală periodică simplă, în alta prin fracție zecimală periodică mixtă.

## baza 10 $\rightarrow$ baza b; numere reale

Exemplu: $(7.8)_2 = ?$

```
7  |  1  /\     2 * 0.8 = 1.6  || citim
3  |  1  ||               -    ||
1  |  1  ||     2 * 0.6 = 1.2  ||
0  |                      -    ||
                2 * 0.2 = 0.4  ||
                          -    ||
                2 * 0.4 = 0.8  \/
                          -
                2 * 0.8   stop, am reintalnit 0.8
```

Deci: $(7.8)_2 = \overline{111.(1100)}$

## baza 10 $\rightarrow$ baza b; numere reale

Dacă $b$ are (în baza 10) doar o cifră, atunci înmulțirea unei părți fracționare cu $b$ se poate face mintal, iar calculele se pot redacta mai simplu:

```
parte_fractionara_0  |  zecimala_1   || citim
parte_fractionara_1  |  zecimala_2   ||
     .               |               ||
     .               |               ||
     .               |               \/
```

În cazul exemplului de mai sus vom avea:

```
0.8  |  1   ||
0.6  |  1   ||
0.2  |  0   ||
0.4  |  0   \/
0.8  |
```

## baza 10 $\rightarrow$ baza b; numere reale

### Exemplu: $(7.8)_5 = ?$

```
7  |  2           0.8  |  4
1  |  1           0    |  stop, am intalnit partea fractionara 0
0  |
```

Deci: $(7.8)_5 = \overline{12.4}$

Observăm că 7.8 se reprezintă în baza 2 prin fracție zecimală periodică, iar în baza 5 (și în baza 10) prin fracție zecimală finită.

## baza b $\rightarrow$ baza 10

Trecerea dintr-o bază $b$ în baza 10 ($b \in \mathbb{N}$, $b \geq 2$):

Regulă:
- înlocuim cifrele cu numerele pe care le reprezintă (scrise în baza 10) și scrierea pozițională cu scriere polinomială în puterile succesive ale lui $b$ (scris tot în baza 10);
- facem calculele în baza 10.

### Exemple:

$(\overline{1101})_2^{-1} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$

$(\overline{A2C})_{16}^{-1} = 10 \times 16^2 + 2 \times 16^1 + 12 \times 16^0 = 2604$

$(\overline{12.4})_5^{-1} = 1 \times 5^1 + 2 \times 5^0 + 4 \times 5^{-1} = 7 + \frac{4}{5} = \frac{39}{5} = 7.8$

Observații:
- în primul exemplu, în stânga primului '=', 0 și 1 înseamnă cifre (în baza 2), iar în dreapta numere (în baza 10); în al doilea exemplu confuzia între cifre și numere este mai mică;
- în ultimul exemplu vedem că pentru cifrele părții fracționare exponenții scad în continuare, la valori negative.

## baza b $\rightarrow$ baza 10

Dacă reprezentarea în baza $b$ de la care pornim este o fracție zecimală periodică (simplă sau mixtă), scrierea polinomială rezultată este infinită.
O asemenea sumă se poate calcula folosind instrumentul matematic al seriilor.

Putem evita folosirea seriilor dacă înlocuim fracția zecimală sursă cu o fracție ordinară (în baza $b$).
Regula de trecere de la fracție zecimală la fracție ordinară în baza $b$ este la fel ca în baza 10, cu diferența că 9 se înlocuiește cu $b - 1$.

### Exemplu:

$(\overline{10.1(011)})_2^{-1} = (\overline{10} + \frac{\overline{1011} - \overline{1}}{\overline{1110}})_2^{-1} = (\overline{10})_2^{-1} + \frac{(\overline{1011})_2^{-1} - (\overline{1})_2^{-1}}{(\overline{1110})_2^{-1}} = 2 + \frac{11-1}{14} =$
$2 + \frac{10}{14} = \frac{38}{14} = \frac{19}{7} = 2.(714285)$

Observație: am folosit și faptul că funcțiile de forma $(.)_b$ și $(.)_b^{-1}$ au proprietăți de morfism (comută cu operațiile aritmetice).

# baza $\mathrm{b}_1 \to$ baza $\mathrm{b}_2$

Trecerea dintr-o bază $b_1$ într-o bază $b_2$ ($b_1, b_2 \in \mathbb{N}$, $b_1, b_2 \geq 2$):

Problema este de aceeași natură ca în cazul trecerilor baza 10 $\to$ baza $b$ și baza $b \to$ baza 10 de mai înainte, deci am putea să aplicăm regulile de acolo. Dar, dacă procedăm ca la trecerea baza 10 $\to$ baza $b$, va trebui să facem calculele în baza $b_1$, iar dacă procedăm ca la trecerea baza $b \to$ baza 10, va trebui să facem calculele în baza $b_2$.

Dacă vrem să facem calculele doar în baza 10, vom trece prin baza 10 compunând cele două reguli.

Regulă:
baza $\mathrm{b}_1 \to$ baza 10 $\to$ baza $\mathrm{b}_2$

# baza $\mathrm{b}_1 \to$ baza $\mathrm{b}_2$

Trecerea baza $\mathrm{b}_1 \to$ baza $\mathrm{b}_2$ se poate face mai simplu, fără a mai trece prin baza 10, dacă una din baze este putere a celeilalte:

- Trecerea $b^k \to b$ $(b, k \in \mathbb{N}\ b \geq 2,\ k \geq 1)$

Regulă:
- înlocuim fiecare cifră a bazei $b^k$ cu câte un grup de $k$ cifre ale bazei $b$;
- eliminăm 0-le extreme (din stânga părții întregi și din dreapta părții fracționare);

- Trecerea $b \to b^k$ $(b, k \in \mathbb{N}\ b \geq 2,\ k \geq 1)$

Regulă:
- grupăm câte $k$ cifre, de la virgulă spre stânga și spre dreapta, completând eventual cu 0-uri grupurile extreme (pentru a avea grupuri de $k$ cifre); aceste 0-uri se pun la stânga în grupul aflat cel mai la stânga al părții întregi și la dreapta în grupul aflat cel mai la dreapta al părții fracționare;
- înlocuim fiecare grup de $k$ cifre ale bazei $b$ cu câte o cifră a bazei $b^k$.

# baza $\mathrm{b}_1 \to$ baza $\mathrm{b}_2$

Observații:

- pentru a aplica regulile de mai înainte, este necesar să știm cele $2^k$ corespondențe posibile între o cifră a bazei $b^k$ și un grup de $k$ cifre ale bazei $b$; acestea se pot determina aplicând trecerea prin baza 10 (baza $\mathrm{b}^k \to$ baza 10 $\to$ baza $\mathrm{b}$) și se pot reține într-un tabel (cu $2^k$ linii);

- explicația intuitivă a acestor reguli este următoarea: dacă în scrierea polinomială în puterile succesive ale lui $b$, cu coeficienți $\in \overline{0, b-1}$, grupăm câte $k$ termeni consecutivi și dăm factor comun puterea lui $b$ cea mai mică, obținem o scriere polinomială în puterile succesive ale lui $b^k$, cu coeficienți $\in \overline{0, b^k - 1}$; din unicitatea acestor scrieri rezultă că ele dau reprezentările în bazele $b$ și $b^k$.

# baza $\mathrm{b}_1 \to$ baza $\mathrm{b}_2$

**Exemplu:** Să se convertească $\overline{1A.C}_{16}$ în baza 2 și $\overline{11011.101}_2$ în baza 16.

Avem $16 = 2^4$, iar grupurile posibile de 4 cifre binare și cifrele hexa corespunzătoare reprezintă numerele $\in \overline{0, 15}$ și sunt cuprinse în tabelul:

| Zec | Bin | Hex | Zec | Bin | Hex | Zec | Bin | Hex | Zec | Bin | Hex |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000 | 0 | 4 | 0100 | 4 | 8 | 1000 | 8 | 12 | 1100 | C |
| 1 | 0001 | 1 | 5 | 0101 | 5 | 9 | 1001 | 9 | 13 | 1101 | D |
| 2 | 0010 | 2 | 6 | 0110 | 6 | 10 | 1010 | A | 14 | 1110 | E |
| 3 | 0011 | 3 | 7 | 0111 | 7 | 11 | 1011 | B | 15 | 1111 | F |

Atunci avem:

$$\overline{1A.C}_{16} \Rightarrow \underbrace{1}\underbrace{A}.\underbrace{C} \Rightarrow \underbrace{0001}\underbrace{1010}.\underbrace{1100} \Rightarrow \cancel{000}11010.11\cancel{00} \Rightarrow \overline{11010.11}_2$$

$$\overline{11011.101}_2 \Rightarrow \underbrace{1}\underbrace{1011}.\underbrace{101} \Rightarrow \underbrace{0001}\underbrace{1011}.\underbrace{1010} \Rightarrow \underbrace{1}\underbrace{B}.\underbrace{A} \Rightarrow \overline{1B.A}_{16}$$

## Operații aritmetice într-o bază *b*

Exemplu: În baza 2, să se adune: `1011 + 110`

Tabla adunării în baza 2 este:

| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 10 |

De exemplu: $\overline{1}+\overline{1}=1+1=2=\overline{10}$.

Atunci, avem:

```
  1 0 1 1 +
    1 1 0
 ---------
1 0 0 0 1
```

Pe poziția unităților am avut $\overline{1}+\overline{0}=\overline{1}$, fără transport; pe următoarea poziție am avut $\overline{1}+\overline{1}=\overline{10}$ (adică numărul 2), s-a păstrat $\overline{0}$ și s-a propagat $\overline{1}$; pe următoarea poziție am avut $\overline{0}+\overline{1}+\overline{1}$ (ultimul $\overline{1}$ provenit din transport) $=\overline{10}$, s-a păstrat $\overline{0}$ și s-a propagat $\overline{1}$; etc.

## Operații aritmetice într-o bază *b*

Exemplu: În baza 16, să se scadă: `BA - 9B`

Avem:

```
 B A -
 9 B
-----
 1 F
```

Pe poziția unităților avem $\overline{A}-\overline{B}=10-11<0$; de aceea, împrumutăm $\overline{1}$ de pe poziția următoare și atunci calculul este $1\times 16+10-11=15=\overline{F}$. Pe poziția următoare avem $\overline{B}-\overline{1}-\overline{9}$ (acel $\overline{1}$ a fost cedat la împrumut) $=11-1-9=1=\overline{1}$.

## Operații aritmetice într-o bază *b*

Exemplu: În baza 2, să se înmulțească: `110.11 x 1.001`
Avem:

```
      1 1 0.1 1 x
        1.0 0 1
     -----------
      1 1 0 1 1
1 1 0 1 1
----------------
1 1 1.1 0 0 1 1
```

Înmulțirea într-o bază oarecare se face tot după reguli asemănătoare ca în baza 10, dar pentru baza 2 aceste reguli se pot simplifica, deoarece singurele cifre sunt $\overline{0}$ și $\overline{1}$, care desemnează respectiv numerele 0 și 1, care sunt factor anulator, respectiv element neutru, la înmulțire; înmulțirea cu un $\overline{0}$ presupune scrierea unui rând de $\overline{0}$-uri, care nu contează la adunare și se pot omite, iar înmulțirea cu un $\overline{1}$ revine la a scrie o copie a deînmulțitului; așadar, pentru a face înmulțirea, este suficient să parcurgem înmulțitorul de la dreapta spre stânga și pentru fiecare $\overline{1}$ întâlnit să mai scriem o copie a deînmulțitului, cu cifra unităților aliniată la acel $\overline{1}$, iar în final să adunăm rândurile scrise - ceea ce am făcut mai sus; în final, numărul de zecimale ale produsului este suma numerelor de zecimale ale factorilor, la fel ca în cazul bazei 10.

## Operații aritmetice într-o bază *b*

Exemplu: În baza 2, să se împartă: `10000.01 : 11`

Avem:

```
1 0 0 0 0.0 1 | 1 1
  1 1         |-----
-----         | 1 0 1.0 1(1 0)
= = 1 0 0
    1 1
    -----
    = = 1 0 1
          1 1
        -----
        = 1 0 0
            1 1
          -----
          = = 1 0
```

## Operații aritmetice într-o bază $b$

Când facem o împărțire, trebuie să urmărim dacă câtul rezultă ca o fracție zecimală finită sau se formează o perioadă. Pentru aceasta, după ce la deîmpărțit am trecut de virgulă și am coborât ultima cifră nenulă, reținem resturile succesive obținute într-o listă; dacă la un moment dat un rest este $\overline{0}$, ne oprim, a rezultat o fracție zecimală finită; dacă la un moment dat un rest se repetă, ne oprim, a rezultat o fracție zecimală periodică, iar perioada începe cu cifra generată la prima apariție a restului care s-a repetat.

În exemplul nostru, după ce la deîmpărțit am trecut de virgulă și am coborât ultima cifră nenulă, am obținut succesiv următoarele resturi: $\overline{10}$, $\overline{1}$, $\overline{10}$ (repetiție).

## Reprezentarea în complement față de 2

Exemplu: Fie $x = 32$, $y = 41$. Calculați $z = x - y$ folosind reprezentarea în complement față de 2 pe 8 biți.

Avem $n = 8$, deci $M = \{-128, \dots, 127\}$.

Deoarece $x, y \geq 0$, avem $[x]_8^s = (x)_2^8$, $[y]_8^s = (y)_2^8$.
Pentru $(x)_2^8$ observăm că $32 = 2^5$, iar pentru $(y)_2^8$ aplicăm regula de conversie baza 10 $\to$ baza 2 din prima secțiune:

| | |
|---|---|
| 41 | 1 |
| 20 | 0 |
| 10 | 0 |
| 5 | 1 |
| 2 | 0 |
| 1 | 1 |
| 0 | |

Atunci:

$$\begin{aligned} [y]_8^s &= 00101001 \\ \boxed{\sim}[y]_8^s &= 11010110 \\ 1 \oplus (\boxed{\sim}[y]_8^s) &= 11010111 \\ [x]_8^s &= 00100000 \\ x \oplus (1 \oplus (\boxed{\sim}[y]_8^s)) &= 11110111 \end{aligned}$$

Avem $11110111 = (247)_2^8$ (aplicând regulile de conversie baza 2 $\to$ baza 10 din prima secțiune) $= [z]_8^s$; deoarece bitul 7 (de semn) din $[z]_8^s$ este 1, înseamnă că $z < 0$, deci $[z]_8^s$ s-a obținut cu al doilea caz al definiției; așadar $[z]_8^s = (256 + z)_2^8 = (247)_2^8$, de unde (aplicând inversa funcției $(.)_2^8$) rezultă că $256 + z = 247$, adică $z = -9$.

Putem verifica, efectuând scăderea în baza 10, că $32 - 41 = -9$.

## Reprezentarea în complement față de 2

Observație: Dacă efectuăm $\boxminus[x]_n^s$, adică $(\boxed{\sim}[x]_n^s) \oplus 1$, pentru $x = -2^{n-1}$, vom obține tot $[x]_n^s$, ceea ce, cu punctul (2) din teorema anterioară, pare să însemne că $-(-2^{n-1}) = -2^{n-1}$ (deoarece au aceeași reprezentare).

De exemplu, pentru $n = 8$ vom obține că $-(-128) = -128$.
Într-adevăr, avem succesiv:

$$[-128]_8^s = 10000000 \overset{\boxed{\sim}}{\to} 01111111 \overset{\oplus 1}{\to} 10000000 = [-128]_8^s.$$

Totuși, nu există nici o contradicție aritmetică sau cu bijectivitatea lui $[.]_n^s$, deoarece formula de la punctul (2) din teorema anterioară se aplică doar dacă $x, -x \in M$, iar în acest caz $-x \notin M$.

## Notația științifică

Notația științifică și aritmetica sistemelor de calcul în virgula mobilă folosesc un mod de reprezentare a unei aproximații a unui număr real care să suporte un domeniu larg de valori. Numerele sunt, în general, reprezentate aproximativ printr-o valoare cu un număr fixat de cifre semnificative care este scalată cu ajutorul unui exponent:

$$\pm\, a \times b^e$$

$a$ este **mantisa** (engl: **mantissa**, **significand**, **coefficient**), $b$ este **baza** de enumerație în care sunt scrise numerele și baza scalării (de obicei 2, 10 sau 16), $e$ este **exponentul**; semnul '$+$' se poate omite.

## Notația științifică

Scalarea cu ajutorul exponentului permite acoperirea unui domeniu larg de valori, atât foarte mici cât și foarte mari, cu aceeași precizie relativă. De exemplu cu 9 cifre semnificative putem descrie, în funcție de exponenții folosiți, atât distanțe în astronomie cu precizie de 1 an lumină, cât și distanțe în fizica atomică cu precizie de 1 femtometru ($10^{-15}$ metri) - ambele precizii fiind acceptabile în domeniul respectiv.

Faptul că se reține un număr fix de cifre pentru mantisă și exponent face ca mulțimea numerelor reale reprezentabile să fie de fapt o mulțime finită de numere raționale, dintr-un interval fixat.
Mai mult, folosirea scalării face ca aceste numere să nu fie uniform spațiate în intervalul respectiv: valorile mici, apropiate de 0, sunt reprezentate cu pași mici (ex. 1 femtometru), în timp ce valorile mari, apropiate de extremele intervalului, sunt reprezentate cu pași mari (ex. 1 an lumină).

Denumirea de **virgulă mobilă** se referă la faptul că **virgula** (engl: **radix point**), numită în funcție de bază și **virgulă zecimală** (engl: **decimal point**), **virgulă binară** (engl: **binary point**), etc., poate fi mutată oriunde în raport cu cifrele semnificative, modificând corespunzător exponentul:

$$0.123 \times 10^2 = 1.230 \times 10^1 = 12.30 \times 10^0 = 123.0 \times 10^{-1}$$

## Notația științifică

În informatică, termenul de **virgulă mobilă** (engl: **floating point**) denumește aritmetica sistemelor de calcul ce folosește reprezentări ale numerelor pentru care virgula binară nu este fixată.

O reprezentare în virgulă mobilă este **notație științifică** dacă are o singură cifră în stânga virgulei și este **notație științifică normalizată** dacă în plus acea cifră este nenulă (i.e nu are 0-uri la început).
De exemplu $1.230 \times 10^1$ este în notație științifică normalizată, dar $0.123 \times 10^2$, $12.30 \times 10^0$, $123.0 \times 10^{-1}$ nu sunt.
Constatăm că notația științifică normalizată pentru un număr real nenul este unică.

## Formatul intern în virgulă mobilă

Reprezentarea în calculator a numerelor reale în virgulă mobilă este reglementată prin standardul **IEEE 754**, adoptat inițial în 1985 (**IEEE 754-1985**) și reactualizat în 2008 (**IEEE 754-2008**). Mai nou, este folosit standardul **ISO/IEC/IEEE 60559:2011**.

Pentru reprezentare (formatul intern) se aleg două dimensiuni $n$ și $k$, $2 \leq k \leq n-2$ și se folosesc locații de $n$ biți $b_{n-1} \dots b_0$ compuse din următoarele câmpuri:
- $b_{n-1}$ ($1$ bit, cel mai semnificativ): semnul;
- $b_{n-2} \dots b_{n-k-1}$ (următorii $k$ biți): caracteristica;
- $b_{n-k-2} \dots b_0$ (ultimii $n$–$k$–$1$ biți, cei mai puțin semnificativi): fracția:

$$\underbrace{b_{n-1}}_{\text{semn (1 bit)}}\ \underbrace{b_{n-2}\ \dots\ b_{n-k-1}}_{\text{caracteristică } (k \text{ biți})}\ \underbrace{b_{n-k-2}\ \dots\ b_0}_{\text{fracție } (n-k-1 \text{ biți})}$$

## Formatul intern în virgulă mobilă

Mulțimea valorilor reprezentabile în acest format este specificată prin intermediul a trei parametri întregi (baza de enumerație și de scalare este întotdeauna $2$):
- $p = n - k$: numărul de cifre ale mantisei (precizia);
- $E_{\min} = -2^{k-1} + 2$: exponentul minim;
- $E_{\max} = 2^{k-1} - 1$: exponentul maxim.

Pentru numerele reale nenule reprezentabile, exponentul $E$ trebuie sa se afle în intervalul de numere întregi $E_{\min} \leq E \leq E_{\max}$, dar se dorește reprezentarea lui ca întreg fără semn, nu prin complement față de doi; de aceea, se mai consideră parametrul:
- $bias = 2^{k-1} - 1$: **bias** (sau **polarizare**);

iar în câmpul de $k$ biți se reprezintă, ca întreg fără semn, valoarea $c = E + bias$ (avem $1 \leq c \leq 2^k - 2$); aceasta s.n. **caracteristică** (sau **exponent biasat** sau **exponent polarizat**).

## Formatul intern în virgulă mobilă

Constatăm că în câmpul caracteristică al formatului pot fi stocate și valorile $c = 0$ (reprezentată cu toți biții $0$) și $c = 2^k - 1$ (reprezentată cu toți biții $1$), care nu pot fi produse de expresiile $E + bias$; ele sunt folosite pentru a codifica valori speciale, ca $\pm 0$, $\pm\infty$, numere denormalizate mici, valori invalide NaN (**Not A Number**).

În ceea ce privește mantisa, numerele vor fi mai întâi scalate a.î. mantisa să aibă în stânga virgulei doar o cifră ($1$ în cazul numerelor normalizate și $0$ în cazul numerelor denormalizate), iar în ultimii $n - k - 1$ biți ai formatului va fi stocată **fracția**, care este partea mantisei din dreapta virgulei (cifra din stânga putându-se deduce din context). În unele texte, prin 'mantisă' este denumită de fapt 'fracția'.

Se mai spune că această reprezentare este **semn și modul** (engl: **sign and magnitude**) deoarece semnul are un bit separat de restul numărului.

## Formatul intern în virgulă mobilă

Valorile reprezentabile în formatul specificat de $n$ și $k$ (sau de $p$, $E_{min}$, $E_{\max}$) și modul lor de reprezentare (codare) sunt următoarele:

- Numerele de forma $(-1)^s \times 2^E \times \overline{1.c_1 \dots c_{p-1}}_2$, unde
  $s \in \{0, 1\}$,
  $E_{\min} \leq E \leq E_{\max}$ este un număr întreg,
  $c_1, \dots, c_{p-1} \in \{0, 1\}$
  (numere reale nenule normalizate).

  Ele sunt reprezentate astfel:
  $b_{n-1} = s$, semnul;
  $b_{n-2} \dots b_{n-k-1} = E + bias$, exponentul biasat, reprezentat ca întreg fără semn; șirul de biți prin care se reprezintă variază de la $0 \dots 01$ (valoarea $1$) la $1 \dots 10$ (valoarea $2^k - 2$);
  $b_{n-k-2} \dots b_0 = c_1 \dots c_{p-1}$, fracția (se omite deci $c_0 = 1$).

## Formatul intern în virgulă mobilă

- Numerele de forma $(-1)^s \times 2^{E_{\min}} \times \overline{0.c_1 \dots c_{p-1}}_2$, unde
  $s \in \{0, 1\}$,
  $c_1, \dots, c_{p-1} \in \{0, 1\}$ și cel puțin unul dintre $c_i$ este $1$
  (numere reale nenule denormalizate mici).

  Ele sunt reprezentate astfel:
  $b_{n-1} = s$, semnul;
  $b_{n-2} \dots b_{n-k-1} = 0$, reprezentat prin șirul de biți $0 \dots 0$;
  $b_{n-k-2} \dots b_0 = c_1 \dots c_{p-1}$, fracția (se omite deci $c_0 = 0$).

## Formatul intern în virgulă mobilă

- Numerele de forma $\pm 0 = (-1)^s \times 2^{E_{\min}} \times \overline{0.0 \dots 0}_2$, unde $s \in \{0, 1\}$.

  Ele sunt reprezentate astfel:
  $b_{n-1} = s$, semnul;
  $b_{n-2} \dots b_{n-k-1} = 0$, reprezentat prin șirul de biți $0 \dots 0$;
  $b_{n-k-2} \dots b_0 = 0 \dots 0$, fracția.

Notăm că numerele de forma $\pm 0$ se reprezină cu toți biții $0$, mai puțin eventual bitul de semn $b_{n-1}$.

Deși reprezentările lui $+0$ și $-0$ sunt diferite (prin bitul de semn), semnul are relevanță în anumite circumstanțe, cum ar fi împărțirea la $0$ (împărțirea $1.0/+0$ produce $+\infty$, împărțirea $1.0/-0$ produce $-\infty$), iar în altele nu.

## Formatul intern în virgulă mobilă

- Valorile $\pm\infty$.

  Ele sunt reprezentate astfel:
  $b_{n-1} = s$, semnul ($0 = +$, $1 = -$);
  $b_{n-2} \dots b_{n-k-1} = 2^k - 1$, reprezentat prin șirul de biți $1 \dots 1$;
  $b_{n-k-2} \ \dots \ b_0 = 0 \ \dots \ 0$, fracția nulă.

- Valori $NaN$ (Not A Number) - sunt entități simbolice codificate în format de virgulă mobilă ce semnifică ideea de valoare invalidă.

  Ele sunt reprezentate astfel:
  $b_{n-1}$ este oarecare;
  $b_{n-2} \dots b_{n-k-1} = 2^k - 1$, reprezentat prin șirul de biți $1 \dots 1$;
  $b_{n-k-2} \ \dots \ b_0$ este o fracție nenulă (cel puțin unul dintre $b_{n-k-2}, \dots, b_0$ este $1$).

## Formatul intern în virgulă mobilă

Există două feluri de $NaN$:

*Signaling NaN* ($sNaN$): semnalează (declanșază) excepția de operație invalidă de fiecare dată când apare ca operand într-o operație; manifestarea excepției poate diferi de la o implementare la alta și poate fi de ignorare sau poate consta în setarea unor flag-uri și/sau executarea unui trap (întrerupere) care lansează o rutină ce primește $NaN$-ul ca parametru;

*Quiet NaN* ($qNaN$): se propagă prin aproape toate operațiile aritmetice fără a semnala (declanșa) excepții, iar rezultatul operațiilor va fi tot un $qNaN$, anume unul dintre $qNaN$-urile date ca operand; ele sunt utile atunci când nu vrem să detectăm/tratăm o eroare chiar la momentul apariției ei ci la un moment ulterior.

Standardul IEEE 754 cere a fi implementat cel puțin un $sNaN$ și cel puțin un $qNaN$; bitul de semn și biții de fracție pot diferi de la o implementare la alta - de exemplu în ei se poate codifica motivul erorii (în cazul $sNaN$-urilor el va fi transmis astfel rutinei de tratare a excepției, iar în cazul $qNaN$-urilor el se va propaga prin operațiile aritmetice până la locul tratării).

## Formatul intern în virgulă mobilă

Aplicând invers regulile de mai sus, putem determinara valoarea $v$ reprezentată printr-un șir de biți $b_{n-1} \ \dots \ b_0$ în formatul specificat de $n$ și $k$. Notăm:
$s =$ numărul $0$ sau $1$ stocat în bitul $b_{n-1}$ (semnul);
$c =$ numărul natural reprezentat ca întreg fără semn în biții $b_{n-2} \ \dots \ b_{n-k-1}$ (caracteristica);
$f =$ șirul de $p-1$ cifre binare stocat în biții $b_{n-k-2} \ \dots \ b_0$ (fracția).

- Dacă $c = 2^k - 1$ (reprezentat ca $1 \dots 1$) și $f \neq 0 \dots 0$, atunci $v$ este un $NaN$ (indiferent de $s$).
- Dacă $c = 2^k - 1$ (reprezentat ca $1 \dots 1$) și $f = 0 \dots 0$, atunci $v = (-1)^s \times \infty$.
- Dacă $0 < c < 2^k - 1$, atunci $v = (-1)^s \times 2^{c-2^{k-1}+1} \times \overline{1.f}_2$ (număr nenul normalizat).
- Dacă $c = 0$ (reprezentat ca $0 \dots 0$) și $f \neq 0 \dots 0$, atunci $v = (-1)^s \times 2^{-2^{k-1}+2} \times \overline{0.f}_2$ (număr nenul denormalizat).
- Dacă $c = 0$ (reprezentat ca $0 \dots 0$) și $f = 0 \dots 0$, atunci $v = (-1)^s \times 0$ (zero).

## Formatul intern în virgulă mobilă

Din aceste reguli deducem că mulțimea numerelor reprezentabile este inclusă în intervalul:

$$[-2^{2^{k-1}-n+k} \times (2^{n-k} - 1), \ +2^{2^{k-1}-n+k} \times (2^{n-k} - 1)]$$

Nu toate numerele reale din acest interval pot fi însă reprezentate, deoarece se rețin doar un numar finit de zecimale în baza 2. Numerele sunt reprezentate cu un anumit pas, care crește cu cât ne apropiem de capetele intervalului. Operațiile aritmetice în virgulă mobilă rotunjesc (și deci alterează) rezultatul a.î. să fie una dintre valorile reprezentabile. De aceea, pasul reprezentării influiențează precizia calculelor.

Întrucât mulțimea valorilor reprezentabile este simetrică față de 0 (dacă un număr $x$ este reprezentabil, atunci și $-x$ este reprezentabil, reprezentarea diferind doar în bitul de semn), vom analiza doar intervalul:

$$[0, \ 2^{2^{k-1}-n+k} \times (2^{n-k} - 1)]$$

## Formatul intern în virgulă mobilă

Cel mai mic număr nenul reprezentabil este:

$$2^{-2^{k-1}+2} \times \underbrace{\overline{0.0\ldots01}_2}_{n-k \text{ cifre ale mantisei}} = 2^{-2^{k-1}-n+k+3},$$

reprezentat: $\underbrace{0}_{\text{s (1 bit)}}\ \underbrace{0\ldots0}_{\text{c (}k\text{ biți)}}\ \underbrace{0\ldots01}_{\text{f(}n-k-1\text{ biți)}}$

Cel mai mare număr nenul reprezentabil este:

$$2^{2^{k-1}-1} \times \underbrace{\overline{1.1\ldots1}_2}_{n-k \text{ cifre ale mantisei}} = 2^{2^{k-1}-n+k} \times (2^{n-k}-1),$$

reprezentat: $\underbrace{0}_{\text{s (1 bit)}}\ \underbrace{1\ldots10}_{\text{c (}k\text{ biți)}}\ \underbrace{1\ldots1}_{\text{f(}n-k-1\text{ biți)}}$

## Formatul intern în virgulă mobilă

Numerele din intervalul $(0,\ 2^{-2^{k-1}-n+k+3})$ sunt prea mici pentru a fi reprezentate.

Numerele din intervalul $[2^{-2^{k-1}-n+k+3},\ 2^{-2^{k-1}+3}]$ sunt reprezentate cu pasul $2^{-2^{k-1}-n+k+3}$.

Pentru orice $i \in \overline{-2^{k-1}+3,\ 2^{k-1}-2}$, numerele din intervalul $[2^i,\ 2^{i+1}]$ sunt reprezentate cu pasul $2^{i-n+k+1}$.

Numerele din intervalul $[2^{2^{k-1}-1},\ 2^{2^{k-1}-n+k} \times (2^{n-k}-1)]$ sunt reprezentate cu pasul $2^{2^{k-1}-n+k}$.

## Formatul intern în virgulă mobilă

Ilustrare grafică:

$\underbrace{0}_{s(1)}\ \underbrace{0\ldots0}_{c(k)}\ \underbrace{0\ldots0}_{f(n\text{-}k\text{-}1)} = 0$

$\vdots$ (numere nereprezentabile)

$\underbrace{0}_{s(1)}\ \underbrace{0\ldots0}_{c(k)}\ \underbrace{0\ldots01}_{f(n\text{-}k\text{-}1)} = 2^{-2^{k-1}+2} \times \overline{0.0\ldots01}_2 = 2^{-2^{k-1}+2} \times 2^{-n+k+1}$
$= 2^{-2^{k-1}-n+k+3}$

(cel mai mic număr nenul reprezentabil)

$\vdots$ pas $2^{-2^{k-1}-n+k+3}$

$\underbrace{0}_{s(1)}\ \underbrace{0\ldots0}_{c(k)}\ \underbrace{1\ldots1}_{f(n\text{-}k\text{-}1)} = 2^{-2^{k-1}+2} \times \overline{0.1\ldots1}_2$
$= 2^{-2^{k-1}+2} \times (2^{n-k-1}-1) \times 2^{-n+k+1}$
$= 2^{-2^{k-1}-n+k+3} \times (2^{n-k-1}-1)$

pas $2^{-2^{k-1}-n+k+3}$

$\underbrace{0}_{s(1)}\ \underbrace{0\ldots01}_{c(k)}\ \underbrace{0\ldots0}_{f(n\text{-}k\text{-}1)} = 2^{-2^{k-1}+2} \times \overline{1.0\ldots0}_2 = 2^{-2^{k-1}+2}$

$\vdots$ pas $2^{-2^{k-1}-n+k+3}$

## Formatul intern în virgulă mobilă

$\underbrace{0}_{s(1)}\ \underbrace{0\ldots01}_{c(k)}\ \underbrace{1\ldots1}_{f(n\text{-}k\text{-}1)} = 2^{-2^{k-1}+2} \times \overline{1.1\ldots1}_2$
$= 2^{-2^{k-1}+2} \times (2^{n-k}-1) \times 2^{-n+k+1}$
$= 2^{-2^{k-1}-n+k+3} \times (2^{n-k}-1)$

pas $2^{-2^{k-1}-n+k+3}$

$\underbrace{0}_{s(1)}\ \underbrace{0\ldots010}_{c(k)}\ \underbrace{0\ldots0}_{f(n\text{-}k\text{-}1)} = 2^{-2^{k-1}+2} \times \overline{10.0\ldots0}_2 = 2^{-2^{k-1}+3} \times \overline{1.0\ldots0}_2$
$= 2^{-2^{k-1}+3}$

$\vdots$ pas $2^{-2^{k-1}-n+k+4}$

$\underbrace{0}_{s(1)}\ \underbrace{0\ldots010}_{c(k)}\ \underbrace{1\ldots1}_{f(n\text{-}k\text{-}1)} = 2^{-2^{k-1}+3} \times \overline{1.1\ldots1}_2$
$= 2^{-2^{k-1}+3} \times (2^{n-k}-1) \times 2^{-n+k+1}$
$= 2^{-2^{k-1}-n+k+4} \times (2^{n-k}-1)$

pas $2^{-2^{k-1}-n+k+4}$

$\underbrace{0}_{s(1)}\ \underbrace{0\ldots011}_{c(k)}\ \underbrace{0\ldots0}_{f(n\text{-}k\text{-}1)} = 2^{-2^{k-1}+3} \times \overline{10.0\ldots0}_2 = 2^{-2^{k-1}+4} \times \overline{1.0\ldots0}_2$
$= 2^{-2^{k-1}+4}$

## Formatul intern în virgulă mobilă

$$\vdots$$

$$\underbrace{0}_{\text{s(1)}}\ \underbrace{1\ldots101}_{\text{c(k)}}\ \underbrace{0\ldots0}_{\text{f(n-k-1)}} = 2^{2^{k-1}-2}\times\overline{1.0\ldots0}_2 = 2^{2^{k-1}-2}$$

$\vdots$ pas $2^{2^{k-1}-n+k-1}$

$$\underbrace{0}_{\text{s(1)}}\ \underbrace{1\ldots101}_{\text{c(k)}}\ \underbrace{1\ldots1}_{\text{f(n-k-1)}} = 2^{2^{k-1}-2}\times\overline{1.1\ldots1}_2 = 2^{2^{k-1}-2}\times(2^{n-k}-1)\times2^{-n+k+1} = 2^{2^{k-1}-n+k-1}\times(2^{n-k}-1)$$

pas $2^{2^{k-1}-n+k-1}$

$$\underbrace{0}_{\text{s(1)}}\ \underbrace{1\ldots10}_{\text{c(k)}}\ \underbrace{0\ldots0}_{\text{f(n-k-1)}} = 2^{2^{k-1}-2}\times\overline{10.0\ldots0}_2 = 2^{2^{k-1}-1}\times\overline{1.0\ldots0}_2 = 2^{2^{k-1}-1}$$

$\vdots$ pas $2^{2^{k-1}-n+k}$

$$\underbrace{0}_{\text{s(1)}}\ \underbrace{1\ldots10}_{\text{c(k)}}\ \underbrace{1\ldots1}_{\text{f(n-k-1)}} = 2^{2^{k-1}-1}\times\overline{1.1\ldots1}_2 = 2^{2^{k-1}-1}\times(2^{n-k}-1)\times2^{-n+k+1} = 2^{2^{k-1}-n+k}\times(2^{n-k}-1)$$

(cel mai mare număr nenul reprezentabil)

## Formatul intern în virgulă mobilă

Observație:

Proiectanții standardului IEEE 754 au dorit o reprezentare a virgulei mobile care să poată fi prelucrată ușor de comparațiile pentru întregi, în special pentru sortare.

Plasarea semnului în bitul cel mai semnificativ permite efectuarea rapidă a testelor '$< 0$', '$> 0$', '$= 0$'.

Plasarea caracteristicii înaintea fracției și faptul că ea este un număr natural (indiferent dacă exponentul este pozitiv sau negativ) ce crește odată cu exponentul permite sortarea numerelor în virgulă mobilă cu ajutorul circuitelor de comparare pentru întregi, deoareace numerele cu exponenți mai mari par mai mari decât numerele cu exponenți mai mici.

Așadar, reprezentarea IEEE 754 poate fi prelucrată cu ajutorul circuitelor de comparare pentru întregi, accelerând sortarea numerelor în virgulă mobilă. Dealtfel, pe ilustrarea grafică de mai înainte se poate observa că șirurile de biți cresc în ordine lexicografică odată cu creșterea numerelor reale reprezentate, ceea ce corespunde creșterii interpretărilor acestor șiruri ca întregi fără semn.

## Formatul intern în virgulă mobilă

Pe aceeași ilustrare grafică se mai poate constata că succesorul lexicografic al șirului de biți prin care s-a reprezentat cel mai mare număr nenul reprezentabil:

$$\underbrace{0}_{\text{s(1)}}\ \underbrace{1\ldots10}_{\text{c(k)}}\ \underbrace{1\ldots1}_{\text{f(n-k-1)}}$$

este șirul de biți prin care se reprezintă $+\infty$:

$$\underbrace{0}_{\text{s(1)}}\ \underbrace{1\ldots1}_{\text{c(k)}}\ \underbrace{0\ldots0}_{\text{f(n-k-1)}}$$

ceea ce exprimă ideea că $+\infty >$ orice număr.

## Formatele single și double

Proiectarea unui sistem de reprezentare în virgulă mobilă presupune o bună alegere a valorilor $n$ (dimensiunea formatului), $k$ (dimensiunea câmpului caracteristică) și $p-1$ (dimensiunea câmpului fracție). Reamintim că $p$ este numărul de cifre semnificative considerate (prima este implicită, restul sunt stocate în câmpul fracție). Avem $n = 1 + k + (p-1) = k + p$.

În general $n$ este un multiplu al dimensiunii cuvântului de memorie (**word**) (ex: 32 biți, 64 biți) și atunci pentru $k$ și $p$ trebuie găsit un compromis, deoarece un bit adăugat la unul dintre câmpuri trebuie luat de la celălalt. Creșterea lui $p$ determină creșterea preciziei numerelor, în timp ce creșterea lui $k$ determină lărgirea domeniului de numere care pot fi reprezentate. Principiile de proiectare ale seturilor de instrucțiuni hardware ne învață că o proiectare bună are nevoie de compromisuri bune.

Două formate în virgulă mobilă descrise în standardul IEEE 754, care se găsește practic în orice calculator conceput dupa anul 1980, sunt **single** și **double** (lor le corespund în limbajul C tipurile float și respectiv double).

# Formatele single și double

Formatul single (precizie simplă): $n = 32$, $k = 8$, $p = 24$

Rezultă: $E_{\min} = -126$, $E_{\max} = 127$, $bias = 127$

Intervalul de numere $\geq 0$ reprezentabile este: $[0,\ 2^{104} \times (2^{24} - 1)]$

Cel mai mic număr nenul reprezentabil este:
$2^{-126} \times \underbrace{\overline{0.0\ldots01}_2}_{24 \text{ cifre ale mantisei}} = 2^{-149}$,

reprezentat: $\underbrace{0}_{\text{s (1 bit)}}\ \underbrace{0\ldots0}_{\text{c (8 biți)}}\ \underbrace{0\ldots01}_{\text{f(23 biți)}}$

Cel mai mare număr nenul reprezentabil este:
$2^{127} \times \underbrace{\overline{1.1\ldots1}_2}_{24 \text{ cifre ale mantisei}} = 2^{104} \times (2^{24} - 1)$,

reprezentat: $\underbrace{0}_{\text{s (1 bit)}}\ \underbrace{1\ldots10}_{\text{c (8 biți)}}\ \underbrace{1\ldots1}_{\text{f(23 biți)}}$

# Formatele single și double

Numerele din intervalul $(0,\ 2^{-149})$ sunt prea mici pentru a fi reprezentate.

Numerele din intervalul $[2^{-149},\ 2^{-125}]$ sunt reprezentate cu pasul $2^{-149}$.

Pentru orice $i \in \overline{-125,\ 126}$, numerele din intervalul $[2^i,\ 2^{i+1}]$ sunt reprezentate cu pasul $2^{i-23}$.

Numerele din intervalul $[2^{127},\ 2^{104} \times (2^{24} - 1)]$ sunt reprezentate cu pasul $2^{104}$.

# Formatele single și double

Ilustrare grafică:

$\underbrace{0}_{\text{s(1)}}\ \underbrace{0\ldots0}_{\text{c(8)}}\ \underbrace{0\ldots0}_{\text{f(23)}} = 0$

⋮ (numere nereprezentabile)

$\underbrace{0}_{\text{s(1)}}\ \underbrace{0\ldots0}_{\text{c(8)}}\ \underbrace{0\ldots01}_{\text{f(23)}} = 2^{-126} \times \overline{0.0\ldots01}_2 = 2^{-126} \times 2^{-23} = 2^{-149}$

(cel mai mic număr nenul reprezentabil)

⋮ pas $2^{-149}$

$\underbrace{0}_{\text{s(1)}}\ \underbrace{0\ldots0}_{\text{c(8)}}\ \underbrace{1\ldots1}_{\text{f(23)}} = 2^{-126} \times \overline{0.1\ldots1}_2 = 2^{-126} \times (2^{23} - 1) \times 2^{-23}$
$= 2^{-149} \times (2^{23} - 1)$

pas $2^{-149}$

$\underbrace{0}_{\text{s(1)}}\ \underbrace{0\ldots01}_{\text{c(8)}}\ \underbrace{0\ldots0}_{\text{f(23)}} = 2^{-126} \times \overline{1.0\ldots0}_2 = 2^{-126}$

⋮ pas $2^{-149}$

# Formatele single și double

$\underbrace{0}_{\text{s(1)}}\ \underbrace{0\ldots01}_{\text{c(8)}}\ \underbrace{1\ldots1}_{\text{f(23)}} = 2^{-126} \times \overline{1.1\ldots1}_2 = 2^{-126} \times (2^{24} - 1) \times 2^{-23}$
$= 2^{-149} \times (2^{24} - 1)$

pas $2^{-149}$

$\underbrace{0}_{\text{s(1)}}\ \underbrace{0\ldots010}_{\text{c(8)}}\ \underbrace{0\ldots0}_{\text{f(23)}} = 2^{-126} \times \overline{10.0\ldots0}_2 = 2^{-125} \times \overline{1.0\ldots0}_2 = 2^{-125}$

⋮ pas $2^{-148}$

$\underbrace{0}_{\text{s(1)}}\ \underbrace{0\ldots010}_{\text{c(8)}}\ \underbrace{1\ldots1}_{\text{f(23)}} = 2^{-125} \times \overline{1.1\ldots1}_2 = 2^{-125} \times (2^{24} - 1) \times 2^{-23}$
$= 2^{-148} \times (2^{24} - 1)$

pas $2^{-148}$

$\underbrace{0}_{\text{s(1)}}\ \underbrace{0\ldots011}_{\text{c(8)}}\ \underbrace{0\ldots0}_{\text{f(23)}} = 2^{-125} \times \overline{10.0\ldots0}_2 = 2^{-124} \times \overline{1.0\ldots0}_2 = 2^{-124}$

## Formatele single și double

$\vdots$

$\underbrace{0}_{s(1)}\ \underbrace{1\dots101}_{c(8)}\ \underbrace{0\dots0}_{f(23)} \ = \ 2^{126}\times\overline{1.0\dots0}_2 \ = \ 2^{126}$

$\vdots$ pas $2^{103}$

$\underbrace{0}_{s(1)}\ \underbrace{1\dots101}_{c(8)}\ \underbrace{1\dots1}_{f(23)} \ = \ 2^{126}\times\overline{1.1\dots1}_2 = 2^{126}\times(2^{24}-1)\times2^{-23}$
$= \ 2^{103}\times(2^{24}-1)$

pas $2^{103}$

$\underbrace{0}_{s(1)}\ \underbrace{1\dots10}_{c(8)}\ \underbrace{0\dots0}_{f(23)} \ = \ 2^{126}\times\overline{10.0\dots0}_2 \ = \ 2^{127}\times\overline{1.0\dots0}_2 \ = \ 2^{127}$

$\vdots$ pas $2^{104}$

$\underbrace{0}_{s(1)}\ \underbrace{1\dots10}_{c(8)}\ \underbrace{1\dots1}_{f(23)} \ = \ 2^{127}\times\overline{1.1\dots1}_2 = 2^{127}\times(2^{24}-1)\times2^{-23}$
$= \ 2^{104}\times(2^{24}-1)$

(cel mai mare număr nenul reprezentabil)

## Formatele single și double

Formatul double (precizie dublă): $n=64$, $k=11$, $p=53$

Rezultă: $E_{\min}=-1022$, $E_{\max}=1023$, $bias=1023$

Intervalul de numere $\geq 0$ reprezentabile este: $[0,\ 2^{971}\times(2^{53}-1)]$

Cel mai mic număr nenul reprezentabil este

$2^{-1022}\times\underbrace{\overline{0.0\dots01}_2}_{53 \text{ cifre ale mantisei}} = 2^{-1074}$,

reprezentat: $\underbrace{0}_{s\ (1\ \text{bit})}\ \underbrace{0\dots0}_{c\ (11\ \text{biți})}\ \underbrace{0\dots01}_{f(52\ \text{biți})}$

Cel mai mare număr reprezentabil este

$2^{1023}\times\underbrace{\overline{1.1\dots1}_2}_{53 \text{ cifre ale mantisei}} = 2^{971}\times(2^{53}-1)$,

reprezentat: $\underbrace{0}_{s\ (1\ \text{bit})}\ \underbrace{1\dots10}_{c\ (11\ \text{biți})}\ \underbrace{1\dots1}_{f(52\ \text{biți})}$

## Formatele single și double

Numerele din intervalul $(0,\ 2^{-1074})$ sunt prea mici pentru a fi reprezentate.

Numerele din intervalul $[2^{-1074},\ 2^{-1021}]$ sunt reprezentate cu pasul $2^{-1074}$.

Pentru orice $i\in\overline{-1021,\ 1022}$, numerele din intervalul $[2^i,\ 2^{i+1}]$ sunt reprezentate cu pasul $2^{i-52}$.

Numerele din intervalul $[2^{1023},\ 2^{971}\times(2^{53}-1)]$ sunt reprezentate cu pasul $2^{971}$.

## Formatele single și double

Ilustrare grafică:

$\underbrace{0}_{s(1)}\ \underbrace{0\dots0}_{c(11)}\ \underbrace{0\dots0}_{f(52)} \ = \ 0$

$\vdots$ (numere nereprezentabile)

$\underbrace{0}_{s(1)}\ \underbrace{0\dots0}_{c(11)}\ \underbrace{0\dots01}_{f(52)} \ = \ 2^{-1022}\times\overline{0.0\dots01}_2 \ = \ 2^{-1022}\times2^{-52} \ = \ 2^{-1074}$

(cel mai mic număr nenul reprezentabil)

$\vdots$ pas $2^{-1074}$

$\underbrace{0}_{s(1)}\ \underbrace{0\dots0}_{c(11)}\ \underbrace{1\dots1}_{f(52)} = 2^{-1022}\times\overline{0.1\dots1}_2 = 2^{-1022}\times(2^{52}-1)\times2^{-52}$
$= \ 2^{-1074}\times(2^{52}-1)$

pas $2^{-1074}$

$\underbrace{0}_{s(1)}\ \underbrace{0\dots01}_{c(11)}\ \underbrace{0\dots0}_{f(52)} \ = \ 2^{-1022}\times\overline{1.0\dots0}_2 \ = \ 2^{-1022}$

$\vdots$ pas $2^{-1074}$

## Formatele single și double

$$\underbrace{0}_{s(1)}\ \underbrace{0\ldots01}_{c(11)}\ \underbrace{1\ldots1}_{f(52)} = 2^{-1022}\times\overline{1.1\ldots1}_2 = 2^{-1022}\times(2^{53}-1)\times2^{-52} = 2^{-1074}\times(2^{53}-1)$$

pas $2^{-1074}$

$$\underbrace{0}_{s(1)}\ \underbrace{0\ldots010}_{c(11)}\ \underbrace{0\ldots0}_{f(52)} = 2^{-1022}\times\overline{10.0\ldots0}_2 = 2^{-1021}\times\overline{1.0\ldots0}_2 = 2^{-1021}$$

$\vdots$ pas $2^{-1073}$

$$\underbrace{0}_{s(1)}\ \underbrace{0\ldots010}_{c(11)}\ \underbrace{1\ldots1}_{f(52)} = 2^{-1021}\times\overline{1.1\ldots1}_2 = 2^{-1021}\times(2^{53}-1)\times2^{-52} = 2^{-1073}\times(2^{53}-1)$$

pas $2^{-1073}$

$$\underbrace{0}_{s(1)}\ \underbrace{0\ldots011}_{c(11)}\ \underbrace{0\ldots0}_{f(52)} = 2^{-1021}\times\overline{10.0\ldots0}_2 = 2^{-1020}\times\overline{1.0\ldots0}_2 = 2^{-1020}$$

## Formatele single și double

$\vdots$

$$\underbrace{0}_{s(1)}\ \underbrace{1\ldots101}_{c(11)}\ \underbrace{0\ldots0}_{f(52)} = 2^{1022}\times\overline{1.0\ldots0}_2 = 2^{1022}$$

$\vdots$ pas $2^{970}$

$$\underbrace{0}_{s(1)}\ \underbrace{1\ldots101}_{c(11)}\ \underbrace{1\ldots1}_{f(52)} = 2^{1022}\times\overline{1.1\ldots1}_2 = 2^{1022}\times(2^{53}-1)\times2^{-52} = 2^{970}\times(2^{53}-1)$$

pas $2^{970}$

$$\underbrace{0}_{s(1)}\ \underbrace{1\ldots10}_{c(11)}\ \underbrace{0\ldots0}_{f(52)} = 2^{1022}\times\overline{10.0\ldots0}_2 = 2^{1023}\times\overline{1.0\ldots0}_2 = 2^{1023}$$

$\vdots$ pas $2^{971}$

$$\underbrace{0}_{s(1)}\ \underbrace{1\ldots10}_{c(11)}\ \underbrace{1\ldots1}_{f(52)} = 2^{1023}\times\overline{1.1\ldots1}_2 = 2^{1023}\times(2^{53}-1)\times2^{-52} = 2^{971}\times(2^{53}-1)$$

(cel mai mare număr nenul reprezentabil)

Formatul double are ca avantaj față de formatul single atât domeniul mai mare al exponentului cât, mai ales, precizia mai mare, datorată mantisei mai mari.

## Formatele single și double

Exemplu: Să se reprezinte în format single și double numărul real
$x = -4.75$.

Reprezentăm $x$ în baza 2:

$4 : 2 = 2$ rest 0
$2 : 2 = 1$ rest 0
$1 : 2 = 0$ rest 1
deci $(4)_2 = \overline{100}$

$0.75 \times 2 = 1.5$
$0.5 \times 2 = 1.0$
deci $(0.75)_2 = \overline{0.11}$

Rezultă $(x)_2 = -\overline{100.11}$

În notația științifică normalizată (în baza 2) avem: $x = -\overline{1.0011}\times 2^2$

Deci:
semnul $= -$
exponentul $= 2$
mantisa $= \overline{1.0011}$

În formatul single, deoarece $-126 \le 2 \le 127$, vom avea:
$s = 1$
$c = (2+127)_2 = (129)_2 = 10000001$
$f = 00110000000000000000000$

## Formatele single și double

Deci reprezentarea binară a lui $x$ ca single este:

$$11000000100110000000000000000000$$

Determinăm și reprezentarea hexa a lui $x$ ca single:

$$\underbrace{1100}_{C}\ \underbrace{0000}_{0}\ \underbrace{1001}_{9}\ \underbrace{1000}_{8}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}$$

deci reprezentarea hexa este:

$$C0980000$$

## Formatele single și double

În formatul double, deoarece $-1022 \le 2 \le 1023$, vom avea:

$$s = 1$$
$$c = (2 + 1023)_2 = (1025)_2 = 10000000001$$
$$f = 0011000000000000000000000000000000000000000000000000$$

Deci reprezentarea binară a lui $x$ ca double este:

$$11000000000100110000000000000000$$
$$00000000000000000000000000000000$$

Determinăm și reprezentarea hexa a lui $x$ ca double:

$$\underbrace{1100}_{\mathrm{C}}\ \underbrace{0000}_{0}\ \underbrace{0001}_{1}\ \underbrace{0011}_{3}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}$$

$$\underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}$$

deci reprezentarea hexa este:

$$\mathrm{C}013000000000000$$

## Formatele single și double

Exemplu: Să se determine valoarea $x$ reprezentată de următorul format single:

$$11000000101000000000000000000000$$

Identificăm valorile câmpurilor componente:

$$\underbrace{1}_{s}\ \underbrace{10000001}_{c}\ \underbrace{01000000000000000000000}_{f}$$

Deci:

$$s = 1$$
$$c = 10000001 = (2^7 + 1)_2 = (129)_2$$
$$f = 01000000000000000000000$$

Întrucât $0 < c < 2^8 - 1 = 255$, rezultă (pentru single *bias* $= 127$):

$$x = (-1)^1 \times 2^{129-127} \times \overline{1.01000000000000000000000}_2 =$$
$$= -2^2 \times (2^0 + 2^{-2}) = -4 \times (1 + 0.25) = -4 \times 1.25 = -5$$

## Adunarea în virgulă mobilă

**Algoritmul de adunare în virgulă mobilă:**

**(1)** Se compară exponenții celor două numere.
Se deplasează spre dreapta cifrele mantisei numărului cu exponent mai mic (deci se deplasează spre stânga virgula sa) până când exponenții celor două numere devin egali.

**(2)** Se adună mantisele. Adunarea mantiselor determină și semnul sumei.

**(3)** Se normalizează suma, fie deplasând dreapta și incrementând exponentul, fie deplasând stânga și decrementând exponentul.
Se testează dacă în urma normalizării sumei s-a produs overflow/underflow (exponentul sumei este în afara domeniului corespunzător formatului).
Dacă da, se semnalează **excepție**.
Dacă nu, se trece la **(4)**.

**(4)** Se rotunjește suma, conform formatului.

**(5)** Se testează dacă suma rotunjită este normalizată (de exemplu, dacă la rotunjire se adună un 1 la un șir de biți 1, rezultatul poate să nu fie normalizat).
Dacă da, **stop**.
Dacă nu, se reia pasul **(3)**.

## Adunarea în virgulă mobilă

Exemplu: Fie $x = 0.5$, $y = -0.4375$. Calculați $x + y$ aplicând algoritmul de adunare în virgulă mobilă pentru formatul single.

Nu este nevoie să determinăm formatul intern (ca single) al celor două numere. Putem opera pe reprezentarea lor matematică în baza 2 în notație științifică, dar vom ține cont de parametrii formatului single: $n = 32$, $k = 8$, $p = 24$, $E_{\min} = -126$, $E_{\max} = bias = 127$.

Reprezentăm $x$, $y$ în baza 2:
Părțile lor întregi sunt $\overline{0}$; pentru părțile lor fracționare, avem:

| | |
|---|---|
| $0.5 \times 2 = 1.0$ | $0.4375 \times 2 = 0.875$ |
| | $0.875 \times 2 = 1.75$ |
| | $0.75 \times 2 = 1.5$ |
| | $0.5 \times 2 = 1.0$ |

Deci $(x)_2 = \overline{0.1}$, $(y)_2 = \overline{-0.0111}$.

În notație științifică normalizată, avem: $x = \overline{1.0} \times 2^{-1}$, $y = -\overline{1.11} \times 2^{-2}$.

Deoarece exponenții $-1$ și $-2$ sunt în intervalul $\{-126, \ldots, 127\}$ iar fracțiile 0 și 11 încap pe 23 biți, numerele $x$ și $y$ se vor reprezenta ca single în formă normalizată, cu $s$, $c$, $f$ deduse din notația științifică normalizată de mai sus (nu mai scriem reprezentările respective).

# Adunarea în virgulă mobilă

Aplicăm pașii algoritmului de adunare în virgulă mobilă descris mai devreme:

**(1)** Avem $-2 < -1$, deci se deplasează spre dreapta cifrele mantisei lui $y$ până ce exponentul său îl egalează pe cel al lui $x$: $y = -\overline{0.111} \times 2^{-1}$.

**(2)** Se adună mantisele: $1.0 - 0.111 = 0.001$, deci suma $x + y = \overline{0.001} \times 2^{-1}$.

**(3)** Se normalizează suma, verificând dacă se produce overflow/underflow:
Avem $\overline{0.001} \times 2^{-1} = \overline{1.0} \times 2^{-4}$ și cum $-126 \leq -4 \leq 127$, nu avem overflow/underflow.

**(4)** Se rotunjește suma, conform formatului single (23 zecimale binare):
Întrucât fracția 0 încape pe 23 biți, rotunjirea nu produce nici o modificare a biților.

**(5)** Suma rezultată este deja normalizată, deci nu se impune încă o normalizare.

În final, am obținut suma $\overline{1.0} \times 2^{-4} = 1 \times 2^0 \times 2^{-4} = 2^{-4} = \frac{1}{16} = 0.0625$.

Putem verifica, efectuând adunarea (scăderea) în baza 10, că $0.5 - 0.4375 = 0.0625$.

# Înmulțirea în virgulă mobilă

**Algoritmul de înmulțire în virgulă mobilă:**

**(1)** Se adună exponenții celor două numere, obținându-se exponentul produsului.
Alternativ, se adună exponenții biasați, scăzând o dată biasul, obținându-se exponentul biasat al produsului.

**(2)** Se înmulțesc mantisele.

**(3)** Se normalizează produsul, dacă este necesar, prin deplasare dreapta și incrementând exponentul.
Se testează dacă în urma normalizării produsului s-a produs overflow/underflow (exponentul produsului este în afara domeniului corespunzător formatului).
Dacă da, se semnalează **excepție**.
Dacă nu, se trece la **(4)**.

**(4)** Se rotunjește produsul, conform formatului.
Se testează dacă produsul rotunjit este normalizat.
Dacă da, se trece la **(5)**.
Dacă nu, se reia pasul **(3)**.

**(5)** Se stabilește semnul produsului ca fiind pozitiv, dacă operanzii au același semn și negativ, dacă operanzii au semn contrar (se face 'xor' între biții de semn ai reprezentările operanzilor).
**Stop**.

# Înmulțirea în virgulă mobilă

**Exemplu:** Fie $x = 0.5$, $y = -0.4375$. Calculați $x \times y$ aplicând algoritmul de înmulțire în virgulă mobilă pentru formatul single.

Ca în exemplul precedent, nu este nevoie să determinăm formatul intern (ca single) al celor două numere, ci putem opera pe reprezentarea lor matematică în baza 2 în notație științifică, ținând cont de parametrii formatului single:
$n = 32$, $k = 8$, $p = 24$, $E_{\min} = -126$, $E_{\max} = bias = 127$.

Am văzut în exemplul precedent că, în notație științifică normalizată, avem:
$x = \overline{1.0} \times 2^{-1}$, $y = -\overline{1.11} \times 2^{-2}$ și că ambele numere se vor reprezenta ca single în formă normalizată, cu $s$, $c$, $f$ deduse din această notație (deoarece exponenții $-1$ și $-2$ sunt în intervalul $\{-126, \ldots, 127\}$ iar fracțiile 0 și 11 încap pe 23 biți).

# Înmulțirea în virgulă mobilă

Aplicăm pașii algoritmului de înmulțire în virgulă mobilă descris mai devreme:

**(1)** Se adună exponenții, obținându-se exponentul produsului:
$-1 + (-2) = -3$.
Alternativ, se adună exponenții biasati (caracteristicile), scăzând o dată biasul, obținându-se exponentul biasat (caracteristica) al produsului:
$(-1 + 127) + (-2 + 127) - 127 = 124$.

**(2)** Se înmulțesc mantisele:

```
    1.0 0 x
    1.1 1
    -----
    1 0 0
  1 0 0
1 0 0
---------
1.1 1 0 0
```

Deci produsul fără semn este (deocamdată): $\overline{1.11} \times 2^{-3}$

**(3)** Se normalizează produsul, verificând dacă se produce overflow/underflow:
Produsul este deja normalizat și, deoarece exponentul său se încadrează în domeniul $-126 \leq -3 \leq 127$ (sau exponentul său biasat se încadrează în domeniul $1 \leq 124 \leq 254$), nu avem overflow/underflow.

## Înmulțirea în virgulă mobilă

**(4)** Se rotunjește produsul, conform formatului single (23 zecimale binare) și se verifică dacă după rotunjire este normalizat:
Întrucât fracția 11 încape pe 23 biți, rotunjirea nu produce modificări, iar produsul este, în continuare, normalizat.

**(5)** Se stabilește semnul produsului ca fiind negativ, deoarece semnele operanzilor diferă.

În final, se obține produsul
$-\overline{1.11} \times 2^{-3} = -(1 \times 2^{0} + 1 \times 2^{-1} + 1 \times 2^{-2}) \times 2^{-3} = -(1 + \frac{1}{2} + \frac{1}{4}) \times \frac{1}{8} = -\frac{7}{4} \times \frac{1}{8} = -\frac{7}{32} = -0.21875.$

Putem verifica, efectuând înmulțirea în baza 10, că $0.5 \times (-0.4375) = -0.21875$.

## Algebra booleană $B_2$

Reamintim câteva dintre formulele anterioare, scrise cu noile notații:

Axiomele prin care se definește o algebră booleană $\langle A, +, \cdot, \bar{}\,, 0, 1\rangle$ sunt:

**asociativitate:** $(x + y) + z = x + (y + z)$, $(xy)z = x(yz)$
**comutativitate:** $x + y = y + x$, $xy = yx$
**absorbție:** $x + xy = x$, $x(x + y) = x$
**distributivitate:** $x(y + z) = xy + xz$, $x + yz = (x + y)(x + z)$
**mărginire:** $0 + x = x$, $0x = 0$, $1 + x = 1$, $1x = x$
**complementare:** $x + \overline{x} = 1$, $x\overline{x} = 0$

Operațiile derivate sunt:

**implicația:** $x \to y \stackrel{d}{=} \overline{x} + y$
**diferența:** $x - y \stackrel{d}{=} x\overline{y}$
**echivalența:** $x \leftrightarrow y \stackrel{d}{=} (x \to y)(y \to x)$
**disjuncția exclusivă (xor):** $x \oplus y \stackrel{d}{=} x\overline{y} + \overline{x}y = (x - y) + (y - x)$

## Algebra booleană $B_2$

Alte proprietăți valabile în algebre booleene:

$x + x = x$, $xx = x$ (**idempotența**, valabilă în orice latice)
$x \to y = \overline{y} \to \overline{x}$, $x - y = \overline{y} - \overline{x}$
$(x \leftrightarrow y) \leftrightarrow z = x \leftrightarrow (y \leftrightarrow z)$, $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
**($\leftrightarrow$ și $\oplus$ sunt asociative)**
$x \leftrightarrow y = y \leftrightarrow x$, $x \oplus y = y \oplus x$ **($\leftrightarrow$ și $\oplus$ sunt comutative)**
$0 \oplus x = x$, $1 \leftrightarrow x = x$, $1 \oplus x = \overline{x}$, $0 \leftrightarrow x = \overline{x}$
$x \oplus y = \overline{x \leftrightarrow y}$, $x \leftrightarrow y = \overline{x \oplus y}$, $x \to y = \overline{x - y}$, $x - y = \overline{x \to y}$
$\overline{\overline{x}} = x$ **(legea dublei negații)**
$x + \overline{x}y = x + y$, $x(\overline{x} + y) = xy$ **(absorbția booleană)**
$\overline{x + y} = \overline{x}\,\overline{y}$, $\overline{xy} = \overline{x} + \overline{y}$ sau $\overline{\overline{x} + \overline{y}} = xy$, $\overline{\overline{x}\,\overline{y}} = x + y$ **(legile lui de Morgan)**
$x + y = 1$ și $xy = 0$ implică $y = \overline{x}$ **(unicitatea complementului)**
$x + y = 0$ d.d. $x = y = 0$, $xy = 1$ d.d. $x = y = 1$

## Algebra booleană $B_2$

Algebra booleană $\langle B_2, +, \cdot, \bar{}\,, 0, 1\rangle$ este caracterizată prin următoarele:

• Mulțimea subiacentă se reduce la cele două elemente distinse, minim și maxim, iar acestea sunt diferite între ele: $B_2 = \{0, 1\}$, $0 \neq 1$. De obicei, elementele 0/1 au semnificația valorilor de adevăr din logica clasică *fals*/*adevărat* (*false*/*true*); de aceea, terminologia legată de această algebră este preluată din logică: 0 și 1 sunt valori de adevăr, operațiile $+, \cdot, \bar{}$ s.n. respectiv "sau", "și", "not" ("or", "and", "not"), tabelele prin care sunt definite ele s.n. tabele de adevăr, etc.. Alteori, elementele 0/1 pot semnifica valorile unor biți, numerele 0/1, cifrele binare, etc.

• Operațiile și relația de ordine pe $B_2$ sunt definite astfel:

| $x$ | $y$ | $x+y$ | $xy$ | $x \to y$ | $x - y$ | $x \leftrightarrow y$ | $x \oplus y$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

| $x$ | $\overline{x}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

Operația 0-ară 0 este elementul 0,
Operația 0-ară 1 este elementul 1,
Relația de ordine este: $0 \leq 1$.

## Algebra booleană $B_2$

Singurele funcții constante : ${B_2}^n \longrightarrow B_2$, $n \geq 1$, sunt:

funcția constantă 0: $0(x_1, \ldots, x_n) \stackrel{d}{=} 0$

funcția constantă 1: $1(x_1, \ldots, x_n) \stackrel{d}{=} 1$

Acestea sunt elementele 0, respectiv 1, ale algebrei booleene ${B_2}^{{B_2}^n}$ și astfel orice expresie algebrică booleană de funcții din ${B_2}^{{B_2}^n}$ se poate transforma într-una echivalentă fără aceste elemente.

**Th:** Orice funcție $f : {B_2}^n \longrightarrow B_2$, $n \geq 1$, este funcție booleană (i.e. se poate scrie ca o expresie algebrică booleană de funcții constante și funcții proiecție).

Astfel, în cazul $B_2$ putem elimina (i), (ii), (iii) din definiția funcțiilor booleene și să numim funcție booleană orice funcție : ${B_2}^n \longrightarrow B_2$, $n \geq 1$.

## Algebra booleană $B_2$

În cele ce urmează, vom folosi însă o definiție mai generală:

**Def:** S.n. **funcție booleană** orice funcție $f : {B_2}^n \longrightarrow {B_2}^k$, $n, k \geq 1$.
Dacă $k = 1$, $f$ se mai numește **funcție booleană scalară**.
Dacă $k > 1$, $f$ se mai numește **funcție booleană vectorială**.
**Componentele** unei funcții booleene vectoriale $f : {B_2}^n \longrightarrow {B_2}^k$ sunt funcțiile $f_i \stackrel{d}{=} \pi_i \circ f : {B_2}^n \longrightarrow B_2$, unde $\pi_i : {B_2}^k \longrightarrow B_2$ este funcția proiecție, $1 \leq i \leq k$.

**Observație:** O funcție booleană vectorială $f : {B_2}^n \longrightarrow {B_2}^k$ se poate asimila cu sistemul componentelor sale: $f = (f_1, \ldots, f_k)$ (pe elemente: $f(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n))$), iar componentele $f_i : {B_2}^n \longrightarrow B_2$ sunt funcții booleene scalare.

## Algebra booleană $B_2$

O funcție booleană poate fi dată în mai multe feluri:

- Informal (dar suficient de precis):

  Fie funcția booleană care asociază unui sistem de trei biți bitul majoritar.

- Printr-o formulă, de exemplu o expresie algebrică booleană:
  Fie $f : {B_2}^3 \longrightarrow {B_2}^4$, $f(x, y, z) = (z(\overline{y \oplus \overline{z}} + x\overline{y}),\ x + z,\ z \oplus \overline{xy},\ y)$

- Printr-un tabel (daca funcția este definită pe ${B_2}^n$, tabelul are $2^n$ linii):
  Fie $f : {B_2}^3 \longrightarrow {B_2}^2$, dată prin tabelul:

| $x$ | $y$ | $z$ | $f_1(x,y,z)$ | $f_2(x,y,z)$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |

## Algebra booleană $B_2$

Când descriem o funcție booleană printr-un tabel, trebuie să fixăm o ordine a semnificațiilor variabilelor și o ordine de amplasare a coloanelor acestora, în concordanță cu semnificația; în exemplul anterior, ordinea semnificațiilor variabilelor este: $x > y > z$ ($x$ este mai semnificativ decât $y$, care este mai semnificativ decât $z$), iar coloanele variabilelor au fost amplasate în ordinea descrescătoare a semnificațiilor, de la stânga la dreapta:

| $x$ | $y$ | $z$ | $f_1(x,y,z)$ | $f_2(x,y,z)$ |
|---|---|---|---|---|

De asemenea, este bine să generăm sistemele de valori ale variabilelor pe linii după o anumită regulă; în exemplul anterior, aceste sisteme de valori au fost generate în ordine lexicografică, de sus în jos.

## Algebra booleană $B_2$

Când procedăm astfel, tabelul are următoarele proprietăți:

• Pe prima coloană, cea a variabilei celei mai semnificative, valorile alternează cu o perioadă mare: avem jumătate 0, apoi jumătate 1; pe coloana a doua, cea a variabilei următoare ca semnificație, perioada se înjumătățește: un sfert 0, apoi un sfert 1, apoi un sfert 0, apoi un sfert 1; pe coloana variabilei următoare ca semnificație, perioada se înjumătățește iar: o optime 0, apoi o optime 1, etc..

• Pe fiecare linie, sistemul valorilor variabilelor constituie transcrierea în baza 2 a numărului de ordine al liniei, numărând de la 0, de sus în jos; de exemplu, sistemul valorilor variabilelor de pe linia 3 este 011 (transcrierea binară a lui 3):

| | $x$ | $y$ | $z$ | $f_1(x,y,z)$ | $f_2(x,y,z)$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | 1 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 |
| 6 | 1 | 1 | 0 | 1 | 1 |
| 7 | 1 | 1 | 1 | 0 | 0 |

Ambele proprietăți vor avea semnificație și se vor dovedi utile la realizarea circuitelor logice.

## Algebra booleană $B_2$

**Observație:**

Dacă am fixat ordinea semnificațiilor variabilelor, cea de amplasare a coloanelor acestora și regula după care generăm sistemele de valori ale variabilelor pe linii, pentru a descrie o funcție booleană scalară $f : {B_2}^n \longrightarrow B_2$, $n \geq 1$, este suficient să dăm sistemul de valori din coloana funcției (deoarece putem deduce pentru ce valori ale variabilelor corespunde o anumită valoare a funcției).
Un asemenea sistem de valori este transcrierea în baza 2 a unui număr întreg de la 0 la $2^{2^n} - 1$, iar acest număr este unic determinat de $f$.

În exemplul de mai sus, $f_1$ poate fi descrisă de sistemul de 8 valori (octetul) 10001010 (de exemplu, bitul 3 (numărând de la stânga și de la 0) este 0 și este valoarea $f_1(x,y,z)$ corespunzătoare liniei 3 din tabel, care corespunde valorilor variabilelor ce transcriu în baza 2 numărul 3: $x = 0$, $y = 1$, $z = 1$). Acest octet este transcrierea în baza 2 a numărului 81 (ținând cont că bitul scris în stânga este cel de rang 0, care corespunde unităților).

Astfel, funcțiile booleene scalare de $n$ variabile, $n \geq 1$, sunt în corespondență bijectivă cu numerele $0, \ldots, 2^{2^n} - 1$, ceea ce ne permite să mutăm studiul acestor funcții într-un cadru aritmetic.
În particular, numărul funcțiilor booleene scalare de $n$ variabile, $n \geq 1$, este $2^{2^n}$.

Numărul funcțiilor booleene $f : {B_2}^n \longrightarrow {B_2}^k$, $n, k \geq 1$, este $(2^{2^n})^k$.

## Algebra booleană $B_2$

Pentru a trece de la un mod de descriere a unei funcții booleene la altul, putem proceda astfel:

• Trecerea de la tabel la formulă (expresie):

Pentru fiecare componentă a funcției, din tabel se poate obține FND sau FNC (vom vedea mai târziu).

• Trecerea de la formulă (expresie) la tabel:

De la stânga la dreapta, asociem coloane variabilelor în ordinea descrescătoare a semnificației, apoi subexpresiilor din formulă, în ordinea crescătoare a agregării lor din alte subexpresii, până obținem coloane corespunzătoare componentelor funcției.

Procedând astfel, putem completa tabelul pe coloane de la stânga la dreapta, fiecare nouă coloană obținându-se din niște coloane deja completate, folosind o singură operație booleană.

## Algebra booleană $B_2$

Astfel, pentru funcția din exemplul anterior:

$$f : {B_2}^3 \longrightarrow {B_2}^4,\ f(x,y,z) = (\underbrace{z(\overline{y \oplus \overline{z}} + x\overline{y})}_{f_1}, \underbrace{x + z}_{f_2}, \underbrace{z \oplus \overline{xy}}_{f_3}, \underbrace{y}_{f_4})$$

avem:

| $x$ | $y = f_4$ | $z$ | $\overline{y}$ | $\overline{z}$ | $x\overline{y}$ | $y \oplus \overline{z}$ | $\overline{y \oplus \overline{z}}$ | $\overline{y \oplus \overline{z}} + x\overline{y}$ | $f_1$ | $f_2$ | $xy$ | $\overline{xy}$ | $f_3$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

De exemplu, coloana a 9-a ($\overline{y \oplus \overline{z}} + x\overline{y}$) s-a calculat din coloanele a 8-a ($\overline{y \oplus \overline{z}}$) și a 6-a ($x\overline{y}$) folosind operația $+$.

## Algebra booleană $B_2$

### Sfat:

La completarea unei coloane, în loc să urmărim pe fiecare linie valorile sursă pentru a calcula valoarea rezultată, se poate dovedi mai util să deducem o regulă de aranjare a valorilor din coloana rezultată din regulile de aranjare a valorilor în coloanele sursă și să completăm noua coloană direct.

De exemplu, observăm că în coloana $y$ alternează doi de 0, doi de 1, ...; atunci deducem că în coloana $\overline{y}$ alternează doi de 1, doi de 0, ... și putem completa coloana $\overline{y}$ direct, fără a mai privi la fiecare linie în coloana $y$.

În cazul coloanei $x\overline{y}$, observăm că $x$ începe cu 4 linii 0, care este element anulator la conjuncție, deci și coloana $x\overline{y}$ începe tot cu 4 linii 0; apoi $x$ are 4 linii 1, care este element neutru la conjuncție, deci următoarele 4 linii din coloana $x\overline{y}$ se copiază din coloana $\overline{y}$, unde deja am văzut că ultimile 4 linii alternează doi de 1, doi de 0.

## Algebra booleană $B_2$

În cazul $B_2$, teorema și observațiile referitoare la FND și FNC se simplifică, deoarece orice funcție $f : {B_2}^n \longrightarrow B_2$, $n \geq 1$, este funcție booleană și pentru orice $\alpha_1, \dots, \alpha_n \in \{0, 1\} = B_2$, $f(\alpha_1, \dots, \alpha_n)$ poate fi doar 0 sau 1:

**Th:** Pentru orice funcție $f : {B_2}^n \longrightarrow B_2$, $n \geq 1$, (i.e. orice funcție booleană scalară) avem:

**(2)** $\forall x_1, \dots, x_n \in B_2,\ f(x_1, \dots, x_n) = \sum\limits_{\substack{\alpha_1, \dots, \alpha_n \in \{0,1\} \\ f(\alpha_1, \dots, \alpha_n) = 1}} x_1^{\alpha_1} \cdots x_n^{\alpha_n}$

**(forma normală disjunctivă, FND)**

**(3)** $\forall x_1, \dots, x_n \in B_2,\ f(x_1, \dots, x_n) = \prod\limits_{\substack{\alpha_1, \dots, \alpha_n \in \{0,1\} \\ f(\overline{\alpha_1}, \dots, \overline{\alpha_n}) = 0}} (x_1^{\alpha_1} + \dots + x_n^{\alpha_n})$

**(forma normală conjunctivă, FNC)**

**(3')** $\forall x_1, \dots, x_n \in B_2,\ f(x_1, \dots, x_n) = \prod\limits_{\substack{\alpha_1, \dots, \alpha_n \in \{0,1\} \\ f(\alpha_1, \dots, \alpha_n) = 0}} (x_1^{\overline{\alpha_1}} + \dots + x_n^{\overline{\alpha_n}})$

**(forma normală conjunctivă, FNC)**

unde am notat: $x_i^0 = \overline{x_i}$, $x_i^1 = x_i$, $1 \leq i \leq n$.
Scrierea în FND, FNC, este unică, abstracție făcând de o permutare a factorilor/termenilor pe baza asociativității și comutativității lui $\cdot/+$.

Pentru realizarea circuitelor logice ne va fi mai utilă FNC dată de (3'), de aceea, în cele ce urmează, pe aceasta o vom numi FNC.

## Algebra booleană $B_2$

Pentru a determina FND și FNC pentru o funcție booleană scalară $f : {B_2}^n \longrightarrow B_2$, $n \geq 1$, putem proceda astfel:

• Dacă funcția este dată printr-un tabel:

Determinarea FND:
- Considerăm liniile tabelului corespunzătoare valorii $f(\alpha_1, \dots, \alpha_n) = 1$;
- Pentru fiecare linie de forma $\alpha_1, \dots, \alpha_n, 1 (= f)$, considerăm produsul (conjuncția) $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ (deci, pentru orice $1 \leq i \leq n$, $x_i$ apare negat dacă $\alpha_i = 0$ și nenegat dacă $\alpha_i = 1$);
- Considerăm suma (disjuncția) produselor.

Determinarea FNC:
- Considerăm liniile tabelului corespunzătoare valorii $f(\alpha_1, \dots, \alpha_n) = 0$;
- Pentru fiecare linie de forma $\alpha_1, \dots, \alpha_n, 0 (= f)$, considerăm suma (disjuncția) $x_1^{\overline{\alpha_1}} + \dots + x_n^{\overline{\alpha_n}}$ (deci, pentru orice $1 \leq i \leq n$, $x_i$ apare negat dacă $\alpha_i = 1$ și nenegat dacă $\alpha_i = 0$);
- Considerăm produsul (conjuncția) sumelor.

## Algebra booleană $B_2$

Exemplu: Determinați FND, FNC, pentru $f : {B_2}^3 \longrightarrow B_2$, dată prin tabelul:

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Numerotând liniile tabelului de sus în jos de la 0 la 7, liniile corespunzătoare valorii $f = 1$ sunt: 1, 2, 4, 6, 7, iar cele corespunzătoare valorii $f = 0$ sunt: 0, 3, 5. Rezultă:

FND: $f(x, y, z) = \overline{x}\ \overline{y}\ z + \overline{x}\ y\ \overline{z} + x\ \overline{y}\ \overline{z} + x\ y\ \overline{z} + x\ y\ z$

FNC: $f(x, y, z) = (x + y + z)(x + \overline{y} + \overline{z})(\overline{x} + y + \overline{z})$

## Algebra booleană $B_2$

Constatăm că sistemul de negări/nenegări de deasupra unui produs/sume transcrie sistemul de valori 0 și 1 ale variabilelor din linia corespunzătoare produsului/sumei respective, care la rândul lui transcrie în baza 2 numărul de ordine al liniei; diferența este că la FND se pune $\bar{\ }$ la 0, iar la FNC se pune $\bar{\ }$ la 1:

| | $x$ | $y$ | $z$ | $f(x,y,z)$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 |

FND: $\overset{1}{\overline{x}\,\overline{y}\,z} + \overset{2}{\overline{x}\,y\,\overline{z}} + \overset{4}{x\,\overline{y}\,\overline{z}} + \overset{6}{x\,y\,\overline{z}} + \overset{7}{x\,y\,z}$

FNC: $\overset{0}{(x+y+z)}\overset{3}{(x+\overline{y}+\overline{z})}\overset{5}{(\overline{x}+y+\overline{z})}$

Astfel, în cazul FND, sistemul ␣$\bar{\ }\bar{\ }$ de deasupra produsului $x\,\overline{y}\,\overline{z}$ transcrie sistemul de valori ale variabilelor din linia corespunzătoare, 100 ($\bar{\ } = 0$), care transcrie în baza 2 numărul liniei, 4.

Similar, în cazul FNC, sistemul ␣$\bar{\ }\bar{\ }$ de deasupra sumei $x+\overline{y}+\overline{z}$ transcrie sistemul de valori ale variabilelor din linia corespunzătoare, 011 ($\bar{\ } = 1$), care transcrie în baza 2 numărul liniei, 3.

## Algebra booleană $B_2$

Astfel, dacă o funcție booleană scalară $f : {B_2}^n \longrightarrow B_2$, $n \geq 1$, este descrisă printr-un tabel construit cu respectarea convențiilor anterioare privind ordinea semnificațiilor variabilelor, cea de amplasare a coloanelor acestora și regula după care generăm sistemele de valori ale variabilelor pe linii, putem determina FND și FNC ale funcției mai repede, fără să urmărim pe fiecare linie valorile variabilelor, astfel:

• FND: scriem atâtea produse $x_1 \cdots x_n$, cu $+$ între ele, câte linii din tabel conțin $f = 1$, apoi transcriem numerele de ordine ale acestor linii prin sisteme de ␣ și $\bar{\ }$ deaspra lor ($\bar{\ } = 0$).

• FNC: scriem atâtea sume $(x_1 + \cdots + x_n)$, cu $\cdot$ între ele, câte linii din tabel conțin $f = 0$, apoi transcriem numerele de ordine ale acestor linii prin sisteme de ␣ și $\bar{\ }$ deaspra lor ($\bar{\ } = 1$).

## Algebra booleană $B_2$

**Observație:**

Pentru o funcție booleană scalară, numărul de produse ale FND nu trebuie să coincidă neapărat cu numărul de sume ale FNC (în exemplul precedent, FND avea 5 produse, FNC avea 3 sume); astfel, FND și FNC nu sunt neapărat la fel de complexe.

În practică, putem prefera pe cea mai simplă dintre ele (urmărim coloana valorilor funcției și determinăm care valoare apare mai rar: 0 sau 1).

Pentru realizarea circuitelor logice însă, va avea o deosebită importanță FND.

## Algebra booleană $B_2$

• Dacă funcția este dată printr-o formulă (expresie):

Determinarea FND:

- Aplicând proprietățile de calcul într-o algebră booleană (de exemplu legile de distributivitate), transformăm expresia prin care este dată funcția a.î. să fie o sumă (disjuncție) de produse (conjuncții) de variabile cu/fără $\bar{\ }$.

- La fiecare produs, înmulțim (conjugăm) câte un factor de forma $(t + \overline{t})$, pentru fiecare variabilă $t$ care lipsește din produsul respectiv (în fapt, am înmulțit niște 1, care nu afectează valoarea produsului).

- Desfacem parantezele, folosind distributivitatea lui $\cdot$ față de $+$ (și alte proprietăți de calcul într-o algebră booleană), a.î. expresia să devină din nou o sumă de produse; acum toate produsele vor conține toate variabilele, dar unele produse pot să se repete.

- Eliminăm produsele duplicat, folosind idempotența lui $+$ (și alte proprietăți de calcul într-o algebră booleană).

## Algebra booleană $B_2$

Determinarea FNC:

- Aplicând proprietățile de calcul într-o algebră booleană (de exemplu legile de distributivitate), transformăm expresia prin care este dată funcția a.î. să fie un produs (conjuncție) de sume (disjuncții) de variabile cu/fără $\bar{}$.

- La fiecare sumă, adunăm (disjungem) câte un termen de forma $t\bar{t}$, pentru fiecare variabilă $t$ care lipsește din suma respectivă (în fapt, am adunat niște 0, care nu afectează valoarea sumei).

- Folosind distributivitatea lui $+$ față de $\cdot$ (și alte proprietăți de calcul într-o algebră booleană), transformăm expresia a.î. să devină din nou un produs de sume; acum toate sumele vor conține toate variabilele, dar unele sume pot să se repete.

- Eliminăm sumele duplicat, folosind idempotența lui $\cdot$ (și alte proprietăți de calcul într-o algebră booleană).

## Algebra booleană $B_2$

Exemplu: Determinați FND, FNC pentru $f : {B_2}^3 \longrightarrow B_2$,
$f(x,y,z) = \overline{\overline{xy} \oplus x} + z$:

FND:
$f(x,y,z) =$
(aducem $f$ la forma unei sume de produse)
$\overline{\overline{xy}\ \overline{x} + \overline{\overline{xy}}x} + z = \overline{(\overline{x}+\overline{y})\overline{x} + xyx} + z = \overline{\overline{x} + xy} + z = \overline{\overline{x} + y} + z =$
$\overline{\overline{x}}\ \overline{y} + z = x\ \overline{y} + z =$
(înmulțim cu $(t+\bar{t})$ pentru fiecare variabilă $t$ lipsă)
$x\overline{y}(z+\overline{z}) + (x+\overline{x})(y+\overline{y})z =$
(desfacem parantezele și rescriem ca sumă de produse)
$x\overline{y}z + x\overline{y}\ \overline{z} + xyz + x\overline{y}z + \overline{x}yz + \overline{x}\ \overline{y}z =$
(eliminăm produsele duplicat)
$x\overline{y}z + x\overline{y}\ \overline{z} + xyz + \overline{x}yz + \overline{x}\ \overline{y}z$

## Algebra booleană $B_2$

FNC:
$f(x,y,z) = x\ \overline{y} + z =$ (am calculat deja)
(aducem $f$ la forma unui produs de sume)
$(x+z)(\overline{y}+z) =$
(adunăm cu $t\ \bar{t}$ pentru fiecare variabilă $t$ lipsă)
$(x + y\ \overline{y} + z)(x\ \overline{x} + \overline{y} + z) =$
(rescriem ca produs de sume)
$(x+y+z)(x+\overline{y}+z)(x+\overline{y}+z)(\overline{x}+\overline{y}+z) =$
(eliminăm sumele duplicat)
$(x+y+z)(x+\overline{y}+z)(\overline{x}+\overline{y}+z)$

Exercițiu: obțineți FND, FNC, pentru această funcție construind tabelul și apoi folosind acest tabel (trebuie să rezulte aceleași expresii pentru FND, FNC, abstracție făcând de o permutare a factorilor/termenilor pe baza asociativității și comutativității lui $\cdot/+$).

## Algebra booleană $B_2$

Teorema precedentă și exemplele anterioare ne oferă și o regulă după care recunoaștem dacă o formulă (expresie) este o FND sau FNC:

- O expresie algebrică booleană în variabilele $x_1, \ldots, x_n$ este o FND, dacă:
  – este o sumă de produse unice (care nu se repetă) de variabile cu/fără $\bar{}$;
  – fiecare produs conține toate variabilele $x_1, \ldots, x_n$.

Exemple:
Expresia $x + \overline{y}z$ în variabilele $x, y, z$ nu este o FND
(deși este o sumă de produse, nu toate produsele conțin toate variabilele $x, y, z$).
Expresia $x\overline{y}z + \overline{x}\ \overline{y}z$ în variabilele $x, y, z$ este o FND
(produsele conțin toate variabilele $x, y, z$, chiar dacă nu sunt toate cele $2^3 = 8$ produse posibile în aceste variabile).

## Algebra booleană $B_2$

- O expresie algebrică booleană în variabilele $x_1, \dots, x_n$ este o FNC, dacă:
  – este un produs de sume unice (care nu se repetă) de variabile cu/fără $^{-}$;
  – fiecare sumă conține toate variabilele $x_1, \dots, x_n$.

Exemple:
Expresia $x(\overline{y} + z)$ în variabilele $x, y, z$ nu este o FNC
(deși este un produs de sume, nu toate sumele conțin toate variabilele $x, y, z$).
Expresia $(x + \overline{y} + z)(\overline{x} + \overline{y} + z)$ în variabilele $x, y, z$ este o FNC
(sumele conțin toate variabilele $x, y, z$, chiar dacă nu sunt toate cele $2^3 = 8$ sume posibile în aceste variabile).

## Algebra booleană $B_2$

**Observație:**

O expresie algebrică booleană în variabilele $x_1, \dots, x_n$ care este o sumă de produse de variabile cu/fără $^{-}$ și care conține toate cele $2^n$ produse posibile este FND a funcției constante 1 (corespunde unui tabel care are 1 în toata coloana valorilor funcției).
Astfel, dacă ea apare ca factor într-un produs, se poate elimina.

De exemplu, avem:
$xz+x\overline{z}+zxy+zx\overline{y}+z\overline{x}y+z\overline{x}\,\overline{y} = x(z+\overline{z})+z(xy+x\overline{y}+\overline{x}y+\overline{x}\,\overline{y}) = x\cdot 1+z\cdot 1 = x+z$

Similar, o expresie algebrică booleană în variabilele $x_1, \dots, x_n$ care este un produs de sume de variabile cu/fără $^{-}$ și care conține toate cele $2^n$ sume posibile este FNC a funcției constante 0 (corespunde unui tabel care are 0 în toata coloana valorilor funcției).
Astfel, dacă ea apare ca termen într-o sumă, se poate elimina.

## Algebra booleană $B_2$

Reamintim, cu noile notații, și teorema care dă o descriere recursivă a funcțiilor booleene scalare, prin funcții care au cu o variabilă mai puțin:

**Th:** Dacă $f : {B_2}^n \longrightarrow B_2$, $n \geq 1$, este o funcție booleană scalară și $1 \leq i \leq n$, atunci:
$\forall x_1, \dots, x_n \in B_2$,
$f(x_1, \dots, x_n) = x_i f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) + \overline{x_i} f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$

## Algebra booleană $B_2$

Dacă notăm variabilele unei funcții booleene în ordinea descrescătoare a semnificației cu $x_{n-1}$, ..., $x_0$, $n \geq 1$, și folosim teorema anterioară pentru $i = 0$:

$$f(x_{n-1}, \dots, x_0) = f(x_{n-1}, \dots, x_1, 0)\overline{x_0} + f(x_{n-1}, \dots, x_1, 1)x_0$$

putem descrie o funcție booleană $f = (f_1, \dots, f_p) : {B_2}^n \longrightarrow {B_2}^p$, $n, p \geq 1$, printr-un tabel mai compact, având doar $2^{n-1}$ linii, însă valorile din tabel nu vor mai fi doar 0 și 1, ci și $\overline{x_0}$ sau $x_0$.

În acest scop, scriem valorile componentelor funcției pentru valori succesive ale variabilei celei mai puțin semnificative $x_0$ pe aceeași linie (conform regulilor de organizare a tabelului de valori descrise mai devreme, ele apar pe linii succesive în tabelul extins) iar în niște coloane separate (care vor fi coloanele rezultat) vom exprima unitar aceste valori ca funcții de $x_0$, folosind formula de recurență de mai sus.

# Algebra booleană $B_2$

Mai exact, fiecare pereche de linii succesive:

$$2^n \text{ linii} \left\{ \begin{array}{cccc|ccc} x_{n-1} & \cdots & x_1 & x_0 & f_1 & \cdots & f_p \\ \hline & \vdots & & & & \vdots & \\ a_{n-1} & \cdots & a_1 & 0 & b_1 & \cdots & b_p \\ a_{n-1} & \cdots & a_1 & 1 & c_1 & \cdots & c_p \\ & \vdots & & & & \vdots & \end{array} \right.$$

se înlocuiește cu:

$$2^{n-1} \text{ linii} \left\{ \begin{array}{ccc|cc|ccc} x_{n-1} & \cdots & x_1 & \multicolumn{2}{c|}{\overbrace{0 \qquad\qquad 1}^{x_0}} & f_1 & \cdots & f_p \\ \hline & \vdots & & \vdots & \vdots & & \vdots & \\ a_{n-1} & \cdots & a_1 & b_1 \cdots b_p & c_1 \cdots c_p & d_1 & \cdots & d_p \\ & \vdots & & \vdots & \vdots & & \vdots & \end{array} \right.$$

unde pentru orice $1 \leq i \leq p$, avem $d_i = b_i\overline{x_0} + c_i x_0$, adică avem corespondența:

| $b_i$ | $c_i$ | $d_i$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | $x_0$ |
| 1 | 0 | $\overline{x_0}$ |
| 1 | 1 | 1 |

# Algebra booleană $B_2$

**Exemplu:** Pentru funcția booleană $f : {B_2}^3 \longrightarrow {B_2}^2$ dată prin tabelul extins:

| $x$ | $y$ | $z$ | $f1$ | $f2$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

vom avea tabelul compact:

$$\begin{array}{cc|cc|cc} x & y & \multicolumn{2}{c|}{\overbrace{0 \qquad\quad 1}^{z}} & f1 & f2 \\ \hline 0 & 0 & 0\,0 & 0\,0 & 0 & 0 \\ 0 & 1 & 1\,1 & 1\,0 & 1 & \overline{z} \\ 1 & 0 & 1\,0 & 0\,1 & \overline{z} & z \\ 1 & 1 & 0\,1 & 1\,0 & z & \overline{z} \end{array}$$

Acest tabel ne arată, de exemplu, că pentru orice $z \in \{0,1\}$ avem:

$$f_1(0,0,z) = 0, \quad f_2(0,1,z) = \overline{z}$$

# Algebra booleană $B_2$

Din teorema referitoare la FND și formula de recurență pentru funcții booleene scalare în cazul general, deducem o regulă după care putem obține o scriere a funcției ca sumă de produse (disjuncție de conjuncții), pornind de la tabelul compact:

Dacă $f : {B_2}^n \longrightarrow B_2$ este o funcție booleană scalară, $n \geq 1$, atunci pentru orice $x_{n-1}, \ldots, x_0 \in B_2$ avem:

$$\begin{aligned} f(x_{n-1}, \ldots, x_0) &= \\ \sum_{\alpha_{n-1},\ldots,\alpha_0 \in B_2} & f(\alpha_{n-1}, \ldots, \alpha_0) x_{n-1}^{\alpha_{n-1}} \cdots x_0^{\alpha_0} = \\ \sum_{\alpha_{n-1},\ldots,\alpha_1 \in B_2} & x_{n-1}^{\alpha_{n-1}} \cdots x_1^{\alpha_1} (f(\alpha_{n-1}, \ldots, \alpha_1, 0)\overline{x_0} + f(\alpha_{n-1}, \ldots, \alpha_1, 1)x_0) = \\ \sum_{\alpha_{n-1},\ldots,\alpha_1 \in B_2} & x_{n-1}^{\alpha_{n-1}} \cdots x_1^{\alpha_1} E(\alpha_{n-1}, \ldots, \alpha_1) \end{aligned}$$

unde $E(\alpha_{n-1}, \ldots, \alpha_1) = f(\alpha_{n-1}, \ldots, \alpha_1, 0)\overline{x_0} + f(\alpha_{n-1}, \ldots, \alpha_1, 1)x_0$ este valoarea care apare în coloana rezultat din tabelul compact al lui $f$, pe linia corespunzătoare valorilor $\alpha_{n-1}$, ..., $\alpha_1$ ale variabilelor $x_{n-1}$, ..., $x_1$; întrucât $f$ ia valori în $B_2$, $E(\alpha_{n-1}, \ldots, \alpha_1)$ poate fi doar 0, 1, $x_0$ sau $\overline{x_0}$.

# Algebra booleană $B_2$

Rezultă următoarea regulă de a obține scrierea funcției ca sumă de produse pornind de la tabelul compact (dacă funcția este vectorială, regula se aplică pentru fiecare componentă în parte):

- Considerăm liniile tabelului corespunzătoare valorii $E(\alpha_{n-1}, \ldots, \alpha_1) \neq 0$.
- Pentru fiecare linie de forma $\alpha_{n-1}, \ldots, \alpha_1, E$ (unde $E$ poate fi 1, $x_0$ sau $\overline{x_0}$), considerăm produsul (conjuncția) $x_{n-1}^{\alpha_{n-1}} \cdots x_1^{\alpha_1} E$.
  Deci, pentru orice $n-1 \geq i \geq 1$, $x_i$ apare negat dacă $\alpha_i = 0$ și nenegat dacă $\alpha_i = 1$, ca la FND, iar $E$ se copiază ca atare din tabel; dacă $E = 1$, se poate omite.
- Considerăm suma (disjuncția) produselor.

**Exemplu:** Pentru funcția $f : {B_2}^3 \longrightarrow {B_2}^2$ din exemplul precedent, avem:

$$f_1(x,y,z) = \overline{x}y + x\overline{y}\ \overline{z} + xyz$$
$$f_2(x,y,z) = \overline{x}y\overline{z} + x\overline{y}z + xy\overline{z}$$

Aceste scrieri nu sunt neapărat FND, de exemplu termenul $\overline{x}y$ nu conține toate variabilele.

## Algebra booleană $B_2$

Scrierea ca sumă de produse se poate simplifica în continuare, folosind proprietatea menționată mai devreme, că suma tuturor produselor cu/fără $\bar{\ }$ formate cu niște variabile date este egală cu 1.

Mai exact, dacă în notațiile precedente $E(\alpha_{n-1},\dots,\alpha_1)$ are o aceeași valoare $y$ pentru niște $\alpha_{i_k}$, ..., $\alpha_{i_1}$ fixate ($n-1\geq i_k>\cdots>i_1\geq 1$, $k\geq 1$) și toate valorile posibile 0 sau 1 ale celorlalte $\alpha_i$ ($n-1\geq i\geq 1$), atunci în scrierea:

$$f(x_{n-1},\dots,x_0)=\sum_{\alpha_{n-1},\dots,\alpha_1\in B_2} x_{n-1}^{\alpha_{n-1}}\cdots x_1^{\alpha_1}E(\alpha_{n-1},\dots,\alpha_1)$$

putem da factor comun și înlocui:

$$\sum_{\substack{\alpha_i\in B_2\\ i\neq i_k,\dots,i_1}} x_{n-1}^{\alpha_{n-1}}\cdots x_{i_k+1}^{\alpha_{i_k+1}}x_{i_k}^{\alpha_{i_k}}x_{i_k-1}^{\alpha_{i_k-1}}\cdots x_{i_1+1}^{\alpha_{i_1+1}}x_{i_1}^{\alpha_{i_1}}x_{i_1-1}^{\alpha_{i_1-1}}\cdots x_1^{\alpha_1}y=$$

$$\Big(\sum_{\substack{\alpha_i\in B_2\\ i\neq i_k,\dots,i_1}} x_{n-1}^{\alpha_{n-1}}\cdots x_{i_k+1}^{\alpha_{i_k+1}}x_{i_k-1}^{\alpha_{i_k-1}}\cdots x_{i_1+1}^{\alpha_{i_1+1}}x_{i_1-1}^{\alpha_{i_1-1}}\cdots x_1^{\alpha_1}\Big)x_{i_k}^{\alpha_{i_k}}\cdots x_{i_1}^{\alpha_{i_1}}y=$$

$$1\cdot x_{i_k}^{\alpha_{i_k}}\cdots x_{i_1}^{\alpha_{i_1}}y=x_{i_k}^{\alpha_{i_k}}\cdots x_{i_1}^{\alpha_{i_1}}y$$

Dacă $E(\alpha_{n-1},\dots,\alpha_1)$ are o aceeași valoare $y$ pentru toate valorile posibile 0 sau 1 ale tuturor $\alpha_i$ ($n-1\geq i\geq 1$), atunci:

$$f(x_{n-1},\dots,x_0)=\sum_{\alpha_{n-1},\dots,\alpha_1\in B_2} x_{n-1}^{\alpha_{n-1}}\cdots x_1^{\alpha_1}y=y$$

## Algebra booleană $B_2$

**Exemplu:** Pentru funcția $f:{B_2}^3\longrightarrow {B_2}^2$ din exemplele precedente, avem:

$$f_2(x,y,z)=y\overline{z}+x\overline{y}z$$

deoarece avem aceeași valoare $f_2(=E)=\overline{z}$ pentru $y=1$ și toate valorile posibile 0 sau 1 ale lui $x$.

Aceasta corespunde următorului calcul prin care simplificăm formula obținută anterior:

$$f_2(x,y,z)=\overline{x}y\overline{z}+x\overline{y}z+xy\overline{z}=(\overline{x}+x)y\overline{z}+x\overline{y}z=1\cdot y\overline{z}+x\overline{y}z=y\overline{z}+x\overline{y}z$$

## Algebra booleană $B_2$

Așadar, dacă în tabelul compact un grup de linii conține aceleași valori într-o coloană rezultat (a unei componente a funcției) și în niște coloane ale variabilelor, iar în celelalte coloane ale variabilelor sunt parcurse toate combinațiile posibile de 0 sau 1, acel grup de linii furnizează un singur termen în scrierea ca sumă de produse a componentei respective a funcției, iar acest termen conține doar variabilele cu valoare comună (cu / fără $\bar{\ }$) și valoarea comună rezultat:

| | $x_{n-1}$ | $\cdots$ | $x_{i_k}$ | $\cdots$ | $x_{i_1}$ | $\cdots$ | $x_1$ | $\cdots$ | $f_j$ | $\cdots$ |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | $\vdots$ | | | | | $\vdots$ | |
| (*) | | $\cdots$ | $\alpha_{i_k}$ | $\cdots$ | $\alpha_{i_1}$ | $\cdots$ | | | $y$ | |
| (*) | | | | $\vdots$ | | | | | $\vdots$ | |
| (*) | | $\cdots$ | $\alpha_{i_k}$ | $\cdots$ | $\alpha_{i_1}$ | $\cdots$ | | | $y$ | |
| | | | | $\vdots$ | | | | | $\vdots$ | |

Grupul de $2^{n-k}$ linii (*) de mai sus nu sunt neapărat consecutive în tabel și conțin aceleași valori în coloanele variabilelor $x_{i_k}$, ..., $x_{i_1}$ și în coloana rezultat $f_j$, iar în coloanele celorlalte $n-k$ variabile sunt parcurse toate cele $2^{n-k}$ combinații posibile de 0 sau 1; acest grup de linii furnizează un singur termen, $x_{i_k}^{\alpha_{i_k}}\cdots x_{i_1}^{\alpha_{i_1}}y$, în scrierea ca sumă de produse a componentei $f_j$.

## Algebra booleană $B_2$

**Exemplu:** Pentru funcția booleană $f:{B_2}^3\longrightarrow {B_2}^4$, în variabilele $x$, $y$, $z$, dată prin tabelul compact (am numerotat liniile):

| | $x$ | $y$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | $\overline{z}$ | 0 | 1 | $\overline{z}$ |
| 1 | 0 | 1 | $\overline{z}$ | $z$ | $z$ | $\overline{z}$ |
| 2 | 1 | 0 | 0 | $z$ | $\overline{z}$ | $\overline{z}$ |
| 3 | 1 | 1 | $z$ | 1 | $z$ | $\overline{z}$ |

avem:

$f_1(x,y,z)=\overline{x}\ \overline{z}+xyz$
(liniile 0, 1 au furnizat un singur termen, $\overline{x}\ \overline{z}$, deoarece în coloana rezultat $f_1$ și în coloana variabilei $x$ avem aceleași valori, iar în coloana variabilei rămase $y$ sunt parcurse toate cele 2 combinații posibile de 0 sau 1)

$f_2(x,y,z)=\overline{x}yz+x\overline{y}z+xy$
(liniile 1, 2 nu pot furniza un singur termen, căci deși în coloana rezultat $f_2$ avem aceeași valoare $z$, în coloanele variabilelor rămase $x$, $y$ nu sunt parcurse toate cele 4 combinații posibile de 0 sau 1)

# Algebra booleană $B_2$

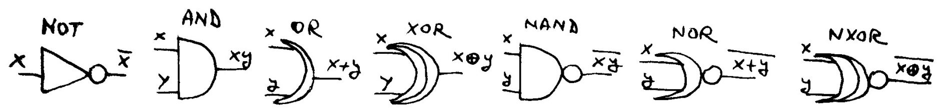
$f_3(x,y,z) = \overline{x}\ \overline{y} + yz + x\overline{y}\ \overline{z}$
(liniile 1, 3 au furnizat un singur termen)

$f_4(x,y,z) = \overline{z}$
(toate cele 4 linii au furnizat un singur termen, $\overline{z}$, deoarece în coloana rezultat $f_4$ avem aceeași valoare $\overline{z}$, iar în coloanele variabilelor rămase $x$, $y$ sunt parcurse toate cele 4 combinații posibile de 0 sau 1).

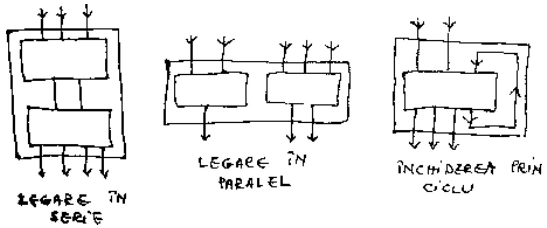
Observăm că în cazul unui tabel compact cu 4 linii numerotate ca mai sus, formula unei componente se simplifica dacă valoarea ei se repetă în liniile:
0 și 1 (dispare variabila a doua),
2 și 3 (dispare variabila a doua),
0 și 2 (dispare prima variabilă),
1 și 3 (dispare prima variabilă),
0, 1, 2, 3 (dispar ambele variabile);
nu se simplifică dacă valoarea se repetă doar în liniile 0 și 3 sau 1 și 2.

# Circuite logice

Porțile sunt:




Operațiile de combinare sunt:

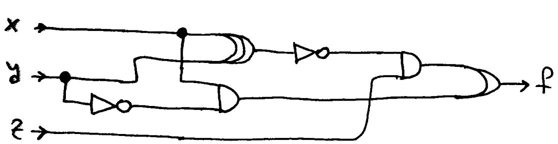



Desenarea săgeților ">" pe liniile de intrare sau ieșire nu este obligatorie, dacă este evident sensul de circulație al datelor; altfel, pe o linie se pot pune oricâte săgeți.

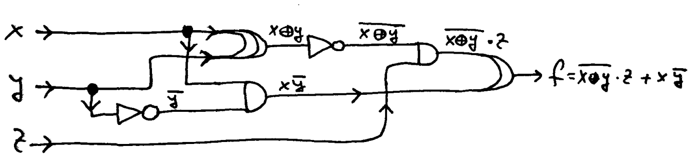
# Circuite logice

În general, circuitul care implementează o formulă booleană se poate construi combinând porțile în aceeași ordine în care se compun operațiile booleene implementate de ele pentru a se obține formula.

De exemplu, formula $f(x,y,z) = \overline{x \oplus y}z + x\overline{y}$ poate fi implementată prin circuitul:

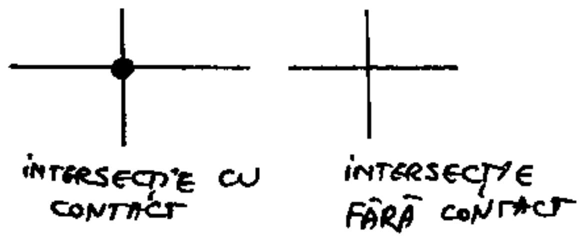


Putem spori claritatea desenului dacă pe anumite linii (nu neapărat pe toate) adăugam săgeți ">" și/sau notăm în dreptul lor formula de calcul a valorii emise pe acolo, de exemplu:

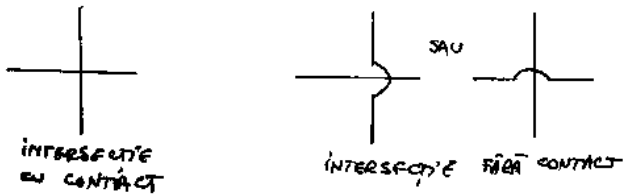


# Circuite logice

În reprezentările grafice ale circuitelor, avem nevoie să distingem între o intersecție de linii fără contact și una cu contact. În figurile anterioare am folosit următoarea simbolizare, pe care o vom utiliza și în continuare:

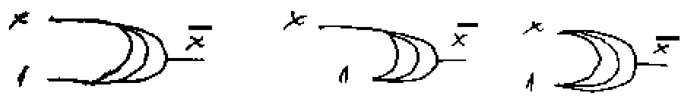



O altă variantă de simbolizare, dar pe care nu o vom folosi, este:



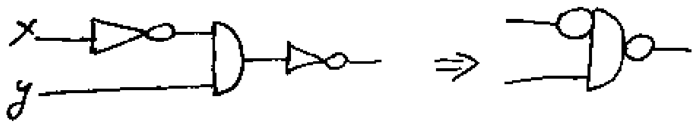
## Circuite logice

Intrările se pot nota în dreptul unor scurte linii de intrare sau direct lângă poarta în care intră; unele intrări sunt fixate pe 0 sau 1, iar acestea se vor nota ca atare; de exemplu:

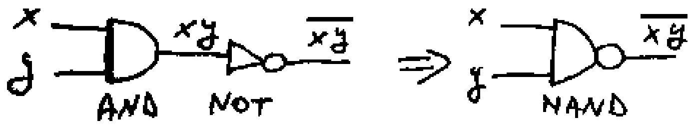


## Circuite logice

O simplificare folosită uneori în reprezentarea grafică a circuitelor logice este următoarea: dacă o poartă "NOT" este legată în serie cu o poartă "OR", "AND", "XOR", nu se mai desenează triunghiul iar cerculețul este lipit de poarta respectivă; de exemplu:

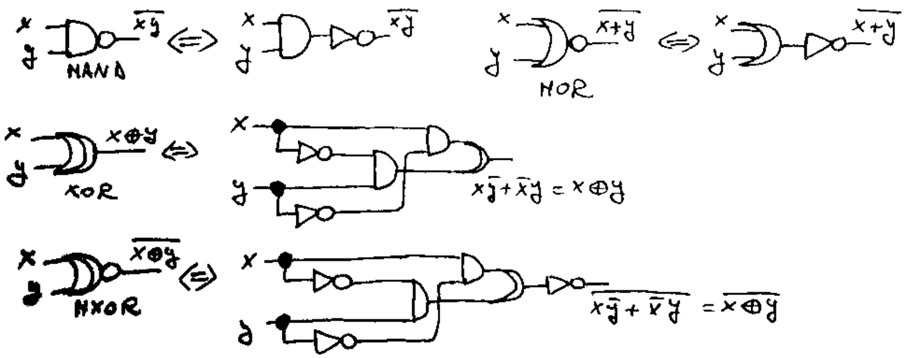


Această simplificare stă la baza simbolurilor folosite pentru porțile "NAND", "NOR", "NXOR"; de exemplu:




## Circuite logice

Constatăm că pentru implementarea diverselor formule de calcul logic nu sunt necesare toate porțile considerate; de exemplu, sunt suficiente porțile "NOT", "AND", "OR", restul porților putându-se exprima în funcție de acestea:




## Circuite logice

Două justificări pentru luarea în considerare a mai multor porți sunt următoarele:

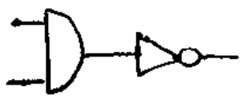
• Anumite porțiuni de circuit (**blocuri**) apar foarte frecvent în structura diverselor circuite; atunci, în loc să le desenăm detaliat de fiecare dată, le asociem un simbol și desenăm simbolul. Astfel, circuitele sunt mai ușor de desenat și de înțeles.

Situația seamănă cu cea din programare când un anumit fragment de cod se repetă de multe ori, eventual cu alte variabile/valori, și în loc să-l rescriem de fiecare dată, preferăm să-l încorporăm într-o procedură, eventual cu parametri, și să apelăm procedura.

## Circuite logice

• Realzarea unui circuit logic prin combinări de porți poate fi asemănată cu realizarea unui circuit electronic prin montarea unor compomente electronice (tranzistori, rezistori, condensatori, diode, etc.) pe o placă cu contacte (cablaj imprimat).

Procesarea efectuată de o componentă electronică se bazează pe fenomene fizice foarte rapide care au loc în interiorul componentei - de exemplu interacțiunea dintre mai multe straturi de semiconductori - le vom numi fenomene de tip (1). Comunicarea între componente se bazează pe fenomene de circulație a curentului electric prin liniile plăcii de contacte - le vom numi fenomene de tip (2). De exemplu, în circuitul:



procesarea presupune un fenomen de tip (1) desfășurat în interiorul lui "AND", apoi un fenomen de tip (2) pentru comunicarea cu "NOT", apoi un fenomen de tip (1) în interiorul lui "NOT".

## Circuite logice

Dacă acest circuit apare frecvent ca bloc în alte circuite, preferăm să încorporăm toată procesarea sa, printr-un fenomen de tip (1), în interiorul unei singure componente (poartă) de un tip nou, simbolizate:

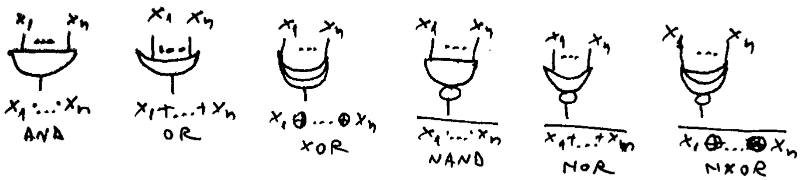
Astfel, toate circuitele care vor conține noua poartă în locul blocului anterior vor funcționa mai repede.

Această abordare este întâlnită în cazul multor circuite electronice care apar frecvent ca parte a altor circuite: în loc să se construiască circuitul de fiecare dată din mai multe componente simple montate pe placa cu contacte, se construiește ca un circuit integrat (chip) care se montează ca o singură componentă pe placa respectivă. Utilizarea frecventă a acestuia justifica fabricarea sa în serie ca un nou tip de componentă electronică.
Circuitele care conțin asemenea chip-uri sunt mai mici ca gabarit, mai ieftine și mai rapide decât cele realizate din componente simple și multe.

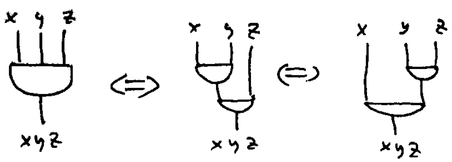
## Circuite logice

În cele ce urmează vor exista și alte circuite, mai complexe, care sunt des întâlnite ca blocuri în alte circuite și de aceea vor avea un simbol propriu, care va fi folosit în locul circuitului detaliat: decodificator, multiplexor, sumator, etc.. Din punct de vedere tehnic, ele se pot realiza ca tipuri distincte de componenete electronice, alături de porți.

Printre acestea, sunt circuitele "AND", "OR", "XOR", "NAND", "NOR", "NXOR" cu mai multe intrări:

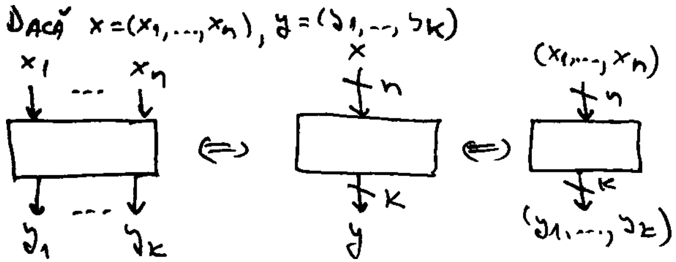



Realizarea lor se bazează pe asociativitatea și comutativitatea operațiilor $\cdot$, $+$, $\oplus$. De exemplu:

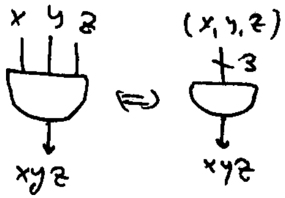


## Circuite logice

Uneori, la desenarea circuitelor logice cu multe linii se folosește următoarea simbolizare mai simplă:




De exemplu:



## Circuite logice

Subliniem că termenul de "circuit logic" se referă la o anumită logică de organizare și funcționare și un anumit scop la care este folosit circuitul, nu și la mijloacele tehnice prin care este el construit.

Cel mai bine, ne imaginăm circuitele logice ca fiind niște niște circuite conceptuale, prin care circulă valori de adevăr; ele descriu în mod abstract un calcul logic.

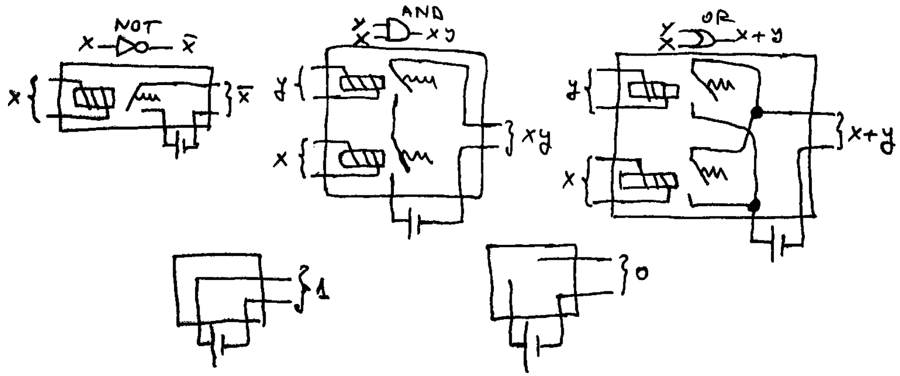
Circuitele logice se pot realiza prin diverse mijloace tehnice: circuite electronice, relee și contacte, angrenaje cu roți dințate, sisteme de pârghii, frânghii și scripeți, conducte de apă și robinete, etc.

Nu trebuie să confundăm însă circuitul logic cu un anumit mod de realizare tehnică a sa.

## Circuite logice

**Exemplu:** Realizare tehnică cu relee și contacte:

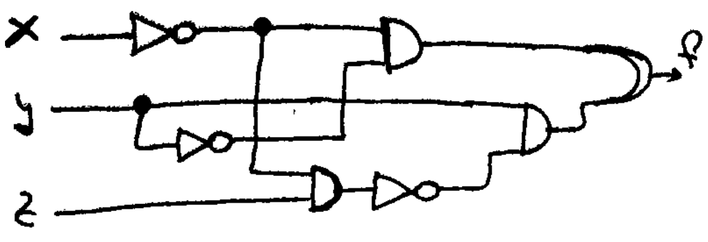
Realizarea porților "NOT", "AND", "OR" și a intrărilor constante 0 și 1:




Deci, liniile circuitului sunt perechi de linii electrice, 0 = fără tensiune, 1 = sub tensiune. Fiecare poartă poate avea propria sursă de curent, sau toate porțile pot fi conectate la o aceeași sursă și nu contează respectarea unei polarități +/−.

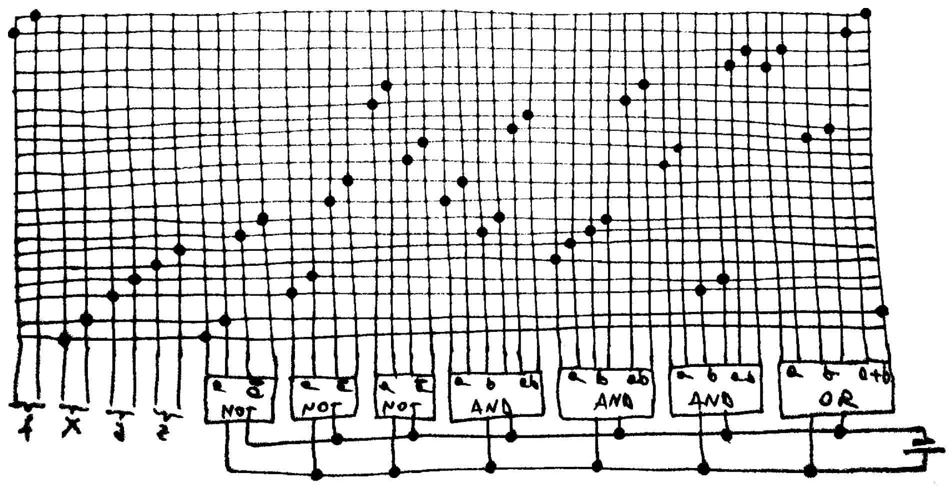
## Circuite logice

Circuitul logic care implementează formula $f(x, y, z) = \overline{x}\,\overline{y} + y\,\overline{\overline{x}\,z}$ este:



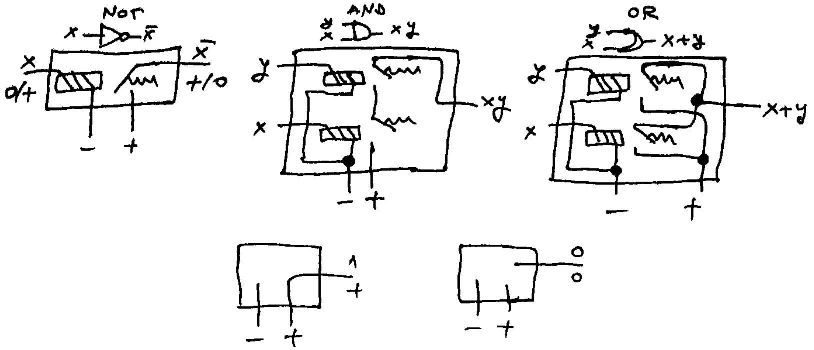
## Circuite logice

Realizarea tehnică a circuitului este:



## Circuite logice

O variantă de realizare cu relee și contacte a porților "NOT", "AND", "OR" și a intrărilor constante 0 și 1, care necesită doar o linie elecrică pentru o linie logică, este:

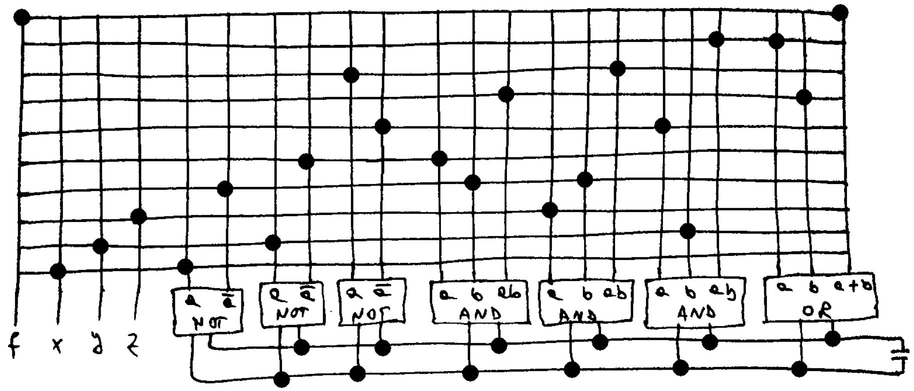



Așadar, 0 = fără tensiune, 1 = tensiune +.

Acum însă toate porțile trebuie conectate la o aceeași sursă de curent și trebuie respectată polaritatea $+/-$.

## Circuite logice

Realizarea tehnică a circuitului pentru formula $f(x, y, z) = \overline{x}\,\overline{y} + y\,\overline{\overline{x}\,z}$ este:




## Circuite logice

De obicei, circuitele logice sunt realizate tehnic prin mijloace electronice (circuite electronice cu chip-uri), deoarece oferă gabarit și cost redus și viteză de funcționare mare; de aceea simbolistica și terminologia sunt preluate din electronică.

Când realizăm circuitele logice prin mijloace electronice putem modela în diverse moduri valorile 0/1; de exemplu:
- 0 = nu trece curentul, 1 = trece curentul;
- 0 = trece curentul, 1 = nu trece curentul;
- și la 0 și la 1 trece curentul, dar are altă tensiune, modulare (transportă un alt tip de semnal).

De obicei este folosită ultima variantă.

## Circuite logice

Mai exact:

• Circuitele electronice din interiorul calculatorului modern sunt **circuite digitale**.
Electronica digitală operează cu doar două niveluri de tensiune electrică importante: o **tensiune înaltă (nivel înalt)** și o **tensiune joasă (nivel jos)**.
Celelalte valori de tensiune sunt temporare și apar în timpul tranziției între cele două valori (o deficiență în proiectarea digitală poate fi prelevarea unui semnal care nu este în mod clar nici înalt nici jos).

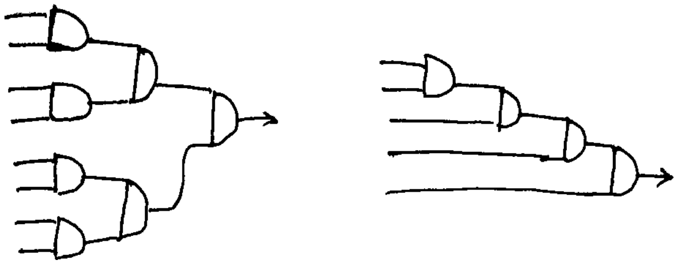
• În diverse categorii de dispozitive logice, valorile și relațiile dintre cele două valori de tensiune diferă.
Astfel, în loc să se facă referire la valorile de tensiune, se discută despre semnale care sunt (logic) **adevărate**, sau 1, sau **activate (asserted)** și despre semnale care sunt (logic) **false**, sau 0, sau **dezactivate (deasserted)**.
Valorile 0 și 1 sunt **complemente** sau **inverse** una celeilalte.

# Circuite logice

Dacă schimbăm valorile pe liniile de intrare, vom avea alte valori pe liniile de ieșire, rezultate în urma procesării (calculului) efectuate de circuit. Valorile rezultat nu apar și nu rămân stabile însă imediat ci după un anumit interval de timp, necesitat de structura circuitului și natura fenomenelor fizice folosite; până atunci, valorile pe liniile de ieșire pot fluctua și, în orice caz, nu sunt relevante.

Acest interval de timp este cu atât mai lung cu cât circuitul este mai dezvoltat pe verticală (are mai multe niveluri de legare în serie).

De exemplu, în figura de mai jos, circuitul din stânga este mai rapid decât cel din dreapta, deși are mai multe porți:



# Circuite logice

Pe lângă porțile "NOT", "AND", "OR", "XOR", "NAND", "NOR", "NXOR", prezentate mai devreme, uneori mai sunt considerate și alte porți, a căror utilitate este legată de mijloacele tehnice prin care sunt construite circuitele (fenomenele fizice folosite); de exemplu, în electronica digitală se mai folosesc:

• **Bufferul:**

Simbol și tabelă de valori:


| IN | OUT |
|---|---|
| 0 | 0 |
| 1 | 1 |

Așadar, bufferul transmite exact valoarea de la intrare la ieșire, cu o mică întârziere cauzată de procesarea sa internă.
Deși nu efectuează un calcul semnificativ (implementează formula $f(x) = x$), bufferul are mai multe utilități:

– Permite izolarea altor porțiuni de circuit unele de altele, împiedicând impedanța unui circuit să afecteze impedanța altuia.
În electronică, **impedanța (impedance)** este o mărime fizică care generealizează rezistența electrică (distincția între cele două se manifestă în cazul curentului alternativ), se notează cu $Z$ și se măsoară în **ohmi ($\Omega$)**.

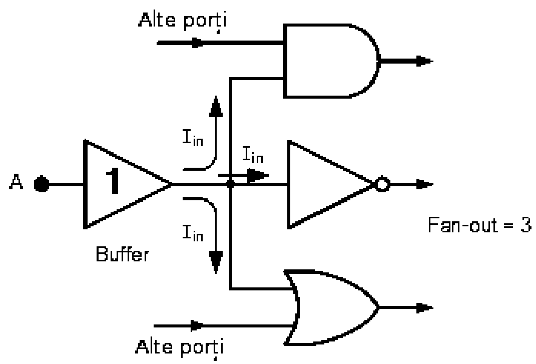
# Circuite logice

– Permite dirijarea unor încărcări mari de curent, cum ar fi cele folosite de comutatoarele cu tranzistori, sau comandarea unui LED, deoarece poate furniza la ieșire curenți mult mai mari decât necesită ca semnal de intrare.
Cu alte cuvinte, bufferul poate fi folosit pentru amplificarea puterii semnalului digital, având o **capacitate fan-out (fan-out capability)** ridicată. Parametrul fan-out al unei porți sau bloc de circuit descrie capacitatea acestuia de a furniza la ieșire curenți mari, oferind o amplificare mai mare semnalului de intrare.

Această proprietate ne permite și să legăm ieșirea unui bloc la intrarea mai multor alte blocuri.

# Circuite logice

Într-adevăr, pentru a funcționa corect, fiecare intrare consumă o anumită cantitate de curent din ieșirea respectivă, a.î. distribuirea ieșirii la mai multe blocuri crește încărcarea blocului sursă. Atunci, inserarea unui buffer între blocul sursă și blocurile destinație poate rezolva problema:




Parametrul "fan-out" este numărul de încărcări paralele care pot fi dirijate simultan de către o singură poartă. Acționând ca o sursă de curent, un buffer poate avea un rating "fan-out" înalt de până la 20 porți din aceeași familie logică.

## Circuite logice

Dacă o poartă are un rating "fan-out" înalt (sursă de curent), ea trebuie să aibă de asemenea un rating fan-in înalt (consumator de curent), Totuşi, întârzierea cauzată de procesarea internă a porţii (propagation delay) se deteriorează rapid ca funcţie de "fan-in", astfel că porţile cu "fan-in" mai mare decât 4 trebuie evitate.

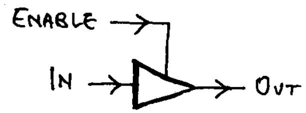
Notăm că proprietăţi asemănătoare bufferului, cum ar fi efectul de amplificare a semnalului, îl au şi alte porţi, ca de exemplu "NOT":  - de aceea, poarta "NOT" se mai numeşte şi **buffer inversor**, sau doar **inversor (inverter)**.

## Circuite logice

• **Bufferul cu 3 stări (Tri-state Buffer):** Este un tip de buffer a cărui ieşire poate fi, la cerere, deconectată "electronic" de la circuitele la care este legată.

Mai exact, el poate genera la ieşire un semnal care nu este logic nici 0, nici 1, iar care d.p.v. funcţional se comportă ca şi când linia de ieşire ar fi deconectată de la intrare (se produce o condiţie de circuit deschis); d.p.v. tehnic, poarta se va comporta ca o componentă electronică cu impedanţă foarte mare, sau ca un contact electric întrerupt (ca rezultat, nu se consumă curent de la sursă); de aceea, această valoare de ieşire s.n. **impedanţă înaltă (Hi-Z)**.

Simbol şi tabel de valori:



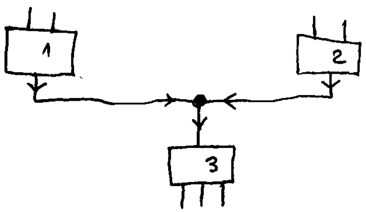

| ENABLE | IN | OUT |
|---|---|---|
| 0 | 0 | Hi-Z |
| 0 | 1 | Hi-Z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Putem considera Hi-Z ca o a treia valoare de adevăr.

## Circuite logice

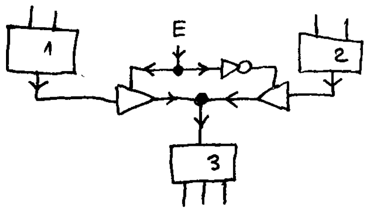
Bufferul cu 3 stări este folosit atunci când, din motive tehnice, dorim să decuplăm funcţional un bloc de la circuit, fără să-l eliminăm fizic (circuitul să se comporte însă ca şi când acel bloc n-ar exista).

De exemplu, într-un circuit logic nu se permite / nu se dă sens contactului între linii care aduc valori (în funcţie de modul tehnic de realizare, întâlnirea între cele două semnale poate avea efecte imprevizibile (de exemplu, un scurtcircuit):

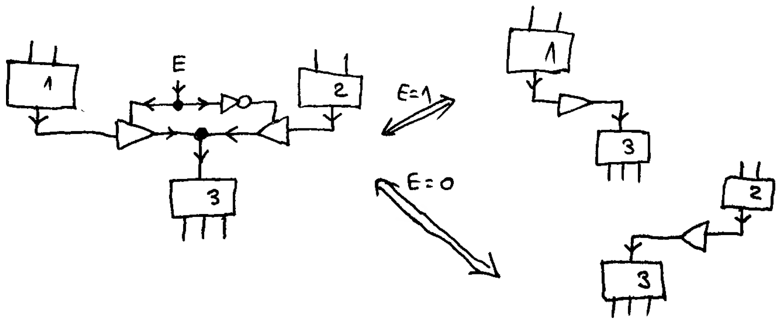



## Circuite logice

Putem însă conecta alternativ mai multe ieşiri la o aceeaşi linie, folosind buffere cu 3 stări:




Atunci:



## Circuite logice

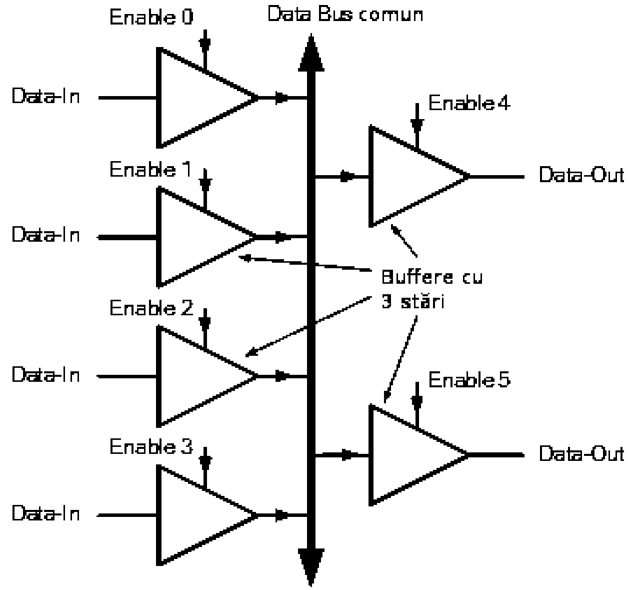
Bufferul cu 3 stări este folosit în multe circuite deoarece permit ca mai multe dispozitive logice să fie conectate la o aceeași linie sau bus fără distrugere fizică sau pierderi de date.

De exemplu, putem avea o linie de date sau bus de date la care sunt conectate memorii, dispozitive de I/O, alte dispozitive periferice sau procesor. Fiecare dintre aceste dispozitive este capabil să emită sau să recepționeze date unul de la altul prin acest unic bus de date în același timp, creând o așa zisă **dispută (contention)**.

Disputele apar când mai multe dispozitive sunt conectate împreună, deoarece unele intenționează să emită la ieșire o tensiune de nivel înalt, altele una de nivel jos; dacă aceste dispozitive încep să emită sau să recepționeze date în același timp, poate apărea un scurtcircuit atunci când un dispozitiv emite spre bus o valoare 1 (care poate însemna tensiunea sursei de curent), în timp ce altul este pe valoarea 0 (care poate însemna legătura cu pământul, ground); acesta poate cauza distrugerea fizică a unor dispozitive sau pierderi de date.

## Circuite logice

Controlul bus-ului de date poate fi realizat folosind buffere cu 3 stări:

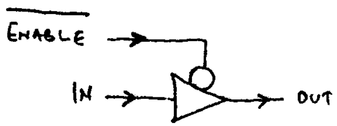



## Circuite logice

Există două tipuri de buffer cu 3 stări: unul a cărui ieșire este controlată de un semnal de control activ la nivel înalt (Active-HIGH) și unul care este controlat de un semnal de control activ la nivel jos (Active-LOW).

Varianta descrisă până acum este bufferul cu 3 stări activ la nivel înalt (Active HIGH Tri-state Buffer): el copiază intrarea la ieșire atunci când semnalul *Enable* are valoarea 1 (altfel furnizează la ieșire HI-Z).

Bufferul cu 3 stări activ la nivel jos (Active LOW Tri-state Buffer) copiază intrarea la ieșire atunci când semnalul *Enable* are valoarea 0 (altfel furnizează la ieșire HI-Z).
Simbol și tabel de valori:




| ENABLE | IN | OUT |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | Hi-Z |
| 1 | 1 | Hi-Z |

## Circuite logice

Blocurile logice sunt împărțite în două categorii, după cum au sau nu au memorie.

Blocurile fără memorie se zic **combinaționale**; la un asemenea bloc, ieșirea depinde doar de intrarea curentă și poate fi descrisă printr-un tabel de adevăr; astfel, circuitul implementează de fapt o funcție booleană; blocurile combinaționale sunt organizate ca circuite logice fără cicluri (0-DS).

În blocurile cu memorie, ieșirea depinde atât de intrarea curentă cât și de valoarea curent păstrată în memorie și care definește **starea** blocului logic; logica care include stări este **logica secvențială**; blocurile cu memorie sunt organizate ca circuite logice cu cel puțin un nivel de cicluri ($n$-DS, $n \geq 1$).

# Circuite logice

O modelare teoretică a circuitelor logice care evidențiază organizarea (structura) acestora se face cu ajutorul grafurilor orientate.

**Def:** Un **circuit** este un graf orientat cu cel puțin o intrare și cel puțin o ieșire, care are două tipuri de noduri: **conectori** și **porți**.

Intrările unui circuit primesc **semnale**, sub forma unor sisteme de valori din mulțimea $\{0,1\}^n$ ($n$ fiind numărul de intrări ale circuitului).

Ieșirile unui circuit vor furniza, de asemenea, semnale.

TODO: Descrierea structurii de graf a unui circuit logic printr-o formulă de Network Algebra.

# Sisteme digitale

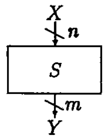
O modelare teoretică a circuitelor logice care evidențiază funcționalitatea acestora o constituie sistemele digitale.

**Def:** Un **sistem digital (digital system, DS)** este o structură $\langle X, Y, f\rangle$, unde $X = V^n$, $Y = V^m$, $f : X \longrightarrow Y$, $n, m \in \mathbf{N}$, iar $V$ este un **alfabet** finit, nevid; $f$ s.n. **funcție de transfer**.

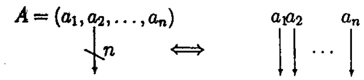
Considerăm doar sisteme digitale **binare**, i.e. cu $V = \{0,1\} = B_2$.

Cazurile $n = 0$ sau $m = 0$ corspund unor sisteme digitale speciale, de **ieșire**, respectiv **intrare**, care vor fi studiate separat.
Deocamdată presupunem $n, m \neq 0$.

Simbol:




unde



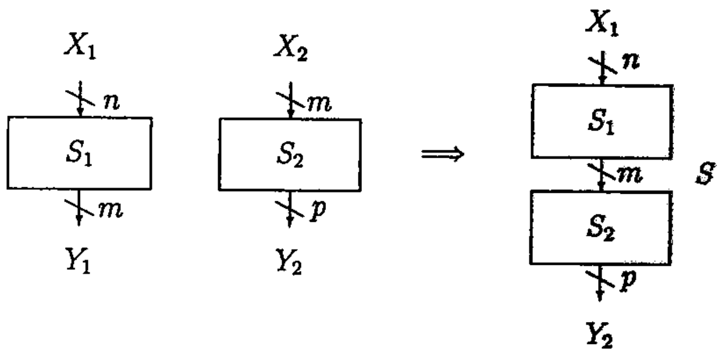

(flux de $n$ date).

# Sisteme digitale

**Operații** cu sisteme digitale:

- **Extensia serială:**

$$\left.\begin{array}{l} S_1 = \langle X_1, Y_1, f\rangle \\ S_2 = \langle X_2, Y_2, g\rangle \\ Y_1 = X_2 \end{array}\right\} \Longrightarrow S = \langle X_1, Y_2, g \circ f\rangle$$

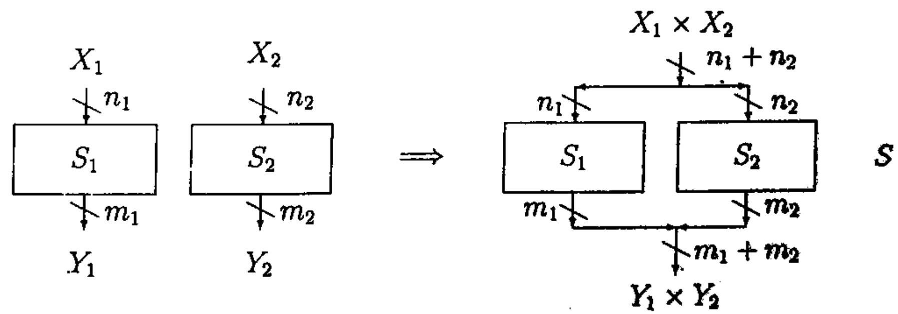



# Sisteme digitale

- **Extensia paralelă:**

$$\left.\begin{array}{l} S_1 = \langle X_1, Y_1, f_1\rangle \\ S_2 = \langle X_2, Y_2, f_2\rangle \end{array}\right\} \Longrightarrow S = S_1 \times S_2 = \langle X_1 \times X_2, Y_1 \times Y_2, f_{12}\rangle,$$

unde $f_{12}(x_1, x_2) = (f_1(x_1), f_2(x_2))$

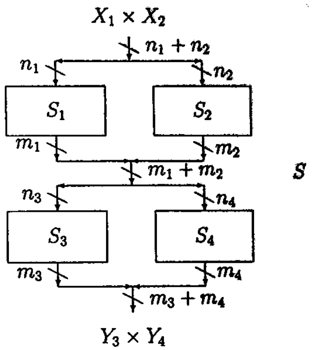



Observăm că într-o extensie paralelă cele două sisteme nu interacționează.

## Sisteme digitale

- **Extensia serial-paralelă:**

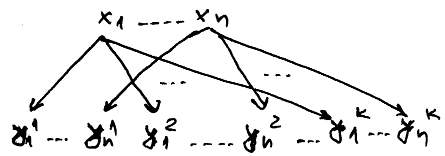
$S_i = \langle X_i, Y_i, f_i \rangle$, $X_i = \{0,1\}^{n_i}$, $Y_i = \{0,1\}^{m_i}$, $1 \le i \le 4$ și $m_1 + m_2 = n_3 + n_4$
$\Longrightarrow$
$S = \langle X_1 \times X_2, Y_3 \times Y_4, f \rangle$, unde $f = f_{34} \circ f_{12} : X_1 \times X_2 \longrightarrow Y_3 \times Y_4$
($f_{12}$, $f_{34}$ sunt funcțiile de transfer ale extensiilor paralele $S_1 \times S_2$, resp. $S_3 \times S_4$)



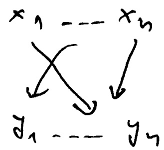

Notăm că ieșirile lui $S_1$ pot fi distribuite atât unor intrări ale lui $S_3$ cât și unor intrări ale lui $S_4$; la fel și ieșirile lui $S_2$.

## Sisteme digitale

Pentru a modela diversele posibilități de ramificare sau permutare a unor linii de circuit, se pot folosi niște sisteme speciale, care sunt angajate și ele în extensiile pe care le facem:



$S = (X^n, X^{n \cdot k}, f)$
$f(x_1, \dots, x_n) = (x_1, \dots, x_n, \dots, x_1, \dots, x_n)$

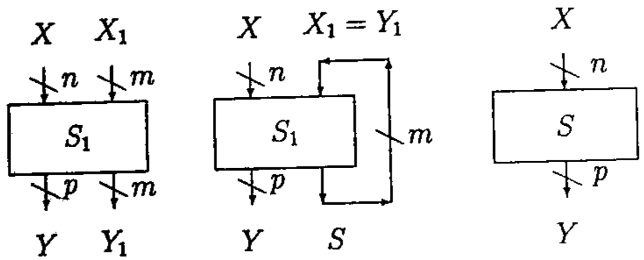


$S = (X^n, X^n, f)$
$f(x_1, \dots, x_n) = (x_{\sigma(1)}, \dots, x_{\sigma(n)})$
$\sigma$ permutare de ordin $n$

## Sisteme digitale

- **Închiderea prin ciclu:**

$S = \langle X \times X_1, Y \times Y_1, h \rangle$, unde:
$X_1 = Y_1$ iar $h = (f, f_1)$, $f : X \times X_1 \longrightarrow Y$, $f_1 : X \times X_1 \longrightarrow Y_1$
$\Longrightarrow$
$S' = \langle X, Y, g \rangle$, unde $g : X \longrightarrow Y$ este definintă prin $g(x) = f(x, f_1(x,y))$;
$f_1$ s.n.**funcție de tranziție** și verifică definiția recursivă $y = f_1(x, f_1(x,y))$




Observăm că apare o variabilă nouă "ascunsă" care ia valori în mulțimea $Q = X_1 = Y_1$; atunci $f : X \times Q \longrightarrow Y$, $f_1 : X \times Q \longrightarrow Q$.

## Sisteme digitale

Mulțimea $Q$ caracterizează comportarea internă a sistemului, care se mai numește **stare**.

Uneori, elementele legate de stare sunt introduse în definiția sistemului respectiv, a.î. el se va scrie: $S' = \langle X, Y, Q, f_1, g \rangle$.

Efectul fundamental al comportării interne constă în evoluția sistemului pe spațiul de valori $Q$, fără modificări ale intrării $X$.

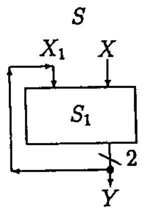
Pentru un $a \in X$ aplicat constant la intrare, ieșirea poate prezenta variații. De aceea, spunem că **autonomia** unui sistem crește ca urmare a introducerii lui într-un ciclu.

**Def:** Comportarea unui sistem digital este **autonomă** d.d. pentru o intrare constantă ieșirea are un comportament dinamic.

**Exemplu:** Fie sistemul $S_1 = \langle X \times X_1, Y, f \rangle$, unde $X = \{0, 1\}$, $X_1 = Y = \{0, 1\}^2$, iar $f$ este definită prin tabelul:

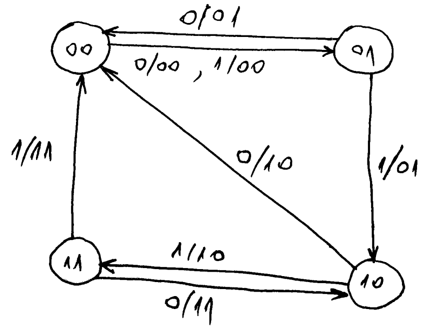
| $X \times X_1$ | $Y$ | $X \times X_1$ | $Y$ |
|---|---|---|---|
| 0 00 | 01 | 1 00 | 01 |
| 0 01 | 00 | 1 01 | 10 |
| 0 10 | 00 | 1 10 | 11 |
| 0 11 | 10 | 1 11 | 00 |

Prin ramificarea în două a ieșirilor $Y$ și apoi identificarea (legarea serială) a uneia dintre ramificări cu $X_1$ se obține un nou sistem $S$:






Funcționalitatea sa este ilustrată mai jos (se face abstracție de timpul real cât este aplicat un semnal la intrare); în cerculețe este notată starea curentă (elementul lui $X_1 = Y$); pe fiecare săgeată sunt notate intrarea curentă (elementul lui $X$) "/" ieșirea curentă (elementul lui $Y$):




De exemplu, din starea "01", cu intrarea "0", se trece în starea "00" și se emite ieșirea "01".
Dependența ieșirii de intrare și stare este dată de tabelul lui $f$ de mai devreme.



**Observație:** Sistemul inițial $S$, fără cicluri, nu avea autonomie - ieșirea sa curentă depindea doar de intrare. După închiderea acestuia printr-un ciclu, sistemul obținut $S'$ capătă, în funcție de tranziție, un comportament al ieșirii autonom de intrare.
De exemplu, intrarea (constantă) $11\ldots1$ în sistemul $S'$ va determina ieșirea ciclică: $01\,10\,11\,00\,01\,\ldots$.



Mai notăm că funcționalitatea circuitelor logice (sistemelelor digitale) fără cicluri nu implementează ideea de etapizare - d.p.v. logic, este o funcționare atemporală, la niște intrări se asociază niște ieșiri, pe baza unui tabel; ieșirea curentă depinde doar de intrarea curentă.

Evident, utilizatorul poate folosi circuitul de mai multe ori, dar această etapizare este a utilizatorului, nu a circuitului - el nu reține informații de la o etapă la alta. De exemplu, un calculator de buzunar simplu (neprogramabil și fără memorie) efectuează fiecare operație care i se comandă ca și cum ar fi prima.

## Sisteme digitale

La circuitele logice (sistemele digitale) cu cicluri, ieșirea curentă depinde atât de intrarea curentă cât și de o informație existentă în circuit (ca stare), introdusă acolo la o operare anterioară; de asemenea, în urma operației efectuate se actualizează și informația (starea) internă.

Astfel, funcționalitatea circuitelor cu cicluri implementează ideea de etapizare. Ele sunt folosite în sisteme unde funcționarea este împărțită în etape ce se succed logic; controlul este asigurat cu ajutorul unui circuit special numit **ceas**, care emite un semnal pulsatoriu în timp, iar aceste pulsații declanșază etapele; la fiecare etapă, blocurile logice componente preiau o intrare, efectuează o operație și produc un rezultat care depinde atât de intrarea preluată cât și de starea lor curentă; din acest rezultat este produsă o ieșire și o nouă stare.

În proiectarea unor asemenea sisteme trebuie rezolvate probleme suplimentare legate de durata fizică a ciclului de ceas (pentru a permite blocurilor logice să efectueze operațiile implementate de ele), sincronizarea blocurilor logice (rezultatul furnizat la ieșirea unui bloc să fie disponibil atunci când îl solicită ca intrare blocul următor), etc.

## Sisteme digitale

În teoria circuitelor logice:

• **Logica combinațională** se referă la un tip de circuit a cărui ieșire depinde doar de intrarea curentă.

• **Logica secvențială** se referă la un tip de circuit a cărui ieșire depinde nu numai de intrarea curentă ci și de istoricul intrărilor sale anterioare.

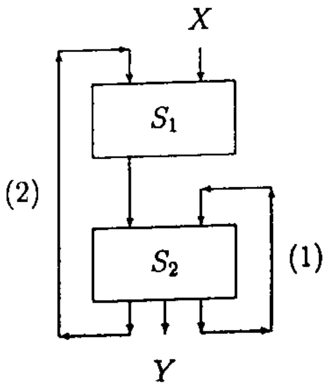
Aceasta înseamnă că logica secvențială are **stare (memorie)**, în timp ce logica combinațională nu.
Cu alte cuvinte, logica secvențială este logică combinațională cu memorie.

Din cele de mai sus, rezultă că circuitele logice (sistemele digitale) fără cicluri sunt prezente în logica combinațională, iar cele cu cicluri în logica secvențială.

## Sisteme digitale

**Def:** Un ciclu $A$ este **inclus** în alt ciclu $B$ dacă $A$ aparține unui sistem care face parte dintr-o extensie serială închisă prin ciclul $B$.
Spunem că $A$ este subciclu al lui $B$.

**Exemplu:** În figura de mai jos, (1) este subciclu al lui (2):



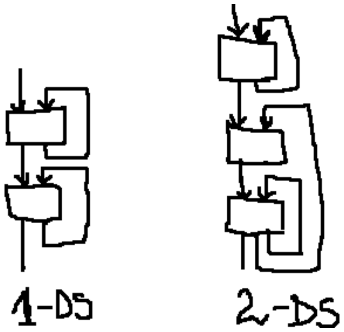

## Sisteme digitale

**Def:** Un **sistem digital de ordin** $n$ **(**$n$**-digital system,** $n$**-DS)**, $n \geq 0$, se definește recursiv astfel:

1. Orice circuit combinațional (fără cicluri) este un 0-DS.
2. Un $(n+1)$-DS se obține dintr-un $n$-DS adăugând un ciclu care include toate ciclurile anterioare.
3. Orice $n$-DS se obține aplicând de un număr finit de ori regulile anterioare.

# Sisteme digitale

**Obs:** Un $n$-DS are $n$ niveluri de cicluri incluse unele în altele, nu $n$ cicluri. De exemplu:




(am presupus că blocurile simbolizate nu conțin cicluri).

# Sisteme digitale

Vom arăta următoarea corespondență ierarhică pentru $n$-DS, care corespunde arhitecturii unui calculator:

- 0-DS: circuite combinaționale, funcții booleene;
- 1-DS: memorii;
- 2-DS: automate finite;
- 3-DS: procesoare;
- 4-DS: calculatoare.

# Circuite combinaționale

Sistemele 0-DS sunt circuite logice fără cicluri.

La aceste circuite, ieșirea curentă depinde doar de intrarea curentă, deci ele nu au memorie și astfel sunt prezente în logica combinațională; de aceea se mai numesc și **circuite combinaționale (combinational logic circuit, CLC)**.

Dependența ieșirii curente de intrarea curentă a unui 0-DS poate fi descrisă printr-un tabel de valori, astfel că un 0-DS implementează o funcție booleană. Așa cum vom vedea mai târziu, orice funcție booleană poate fi implementată printr-un 0-DS, deci avem o echivalență între 0-DS și funcții booleene.

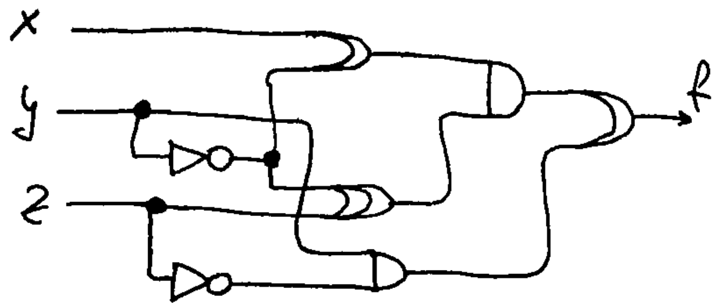
Există mai multe modalități de a implementa o funcție booleană ca un 0-DS: implementare directă a unei formule date, minimizarea numărului de operații și implementarea printr-un circuit minimal, implementarea printr-un circuit general construit după un anumit tipar, implementarea cu ajutorul unor blocuri CLC speciale: codificator, multiplexor, etc.

# Implementare directă

## Implementare directă:

O funcție booleană dată printr-o formulă poate fi implementată combinând porțile în aceeași ordine în care se compun operațiile booleene implementate de ele pentru a se obține formula.

**Exemplu:** Funcția $f : {B_2}^3 \longrightarrow B_2$, $f(x, y, z) = (x + \overline{y})(z \oplus \overline{y}) + y\overline{z}$ poate fi implementată prin circuitul:



# Circuit minimal

## Circuit minimal:

O funcție booleană poate fi implementată descriind-o mai întâi printr-o formulă cu număr minim de operații și implementând apoi direct această formulă (circuitul va avea un număr minim de porți).

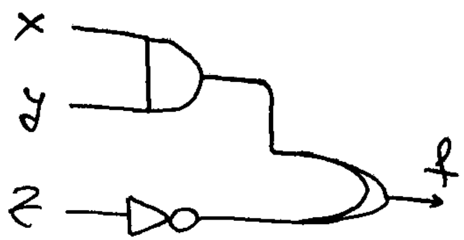
Problema generală a minimizării circuitului este considerată **intractabilă (intractable)**, i.e. poate fi rezolvată în teorie (de exemplu, fiind dat un timp lung, dar finit), dar în practică durează prea mult pentru ca soluția ei să fie utilă.

Există însă metode euristice eficiente, ca **hărțile Karnaugh (Karnaugh maps)** și **algoritmul QuineMcCluskey**.

În multe cazuri poate fi util să încercăm să transformăm formula, aplicând proprietăți de algebră booleană, până când aceasta nu mai conține 0 sau 1 iar variabilele nu se mai repetă.

# Circuit minimal

**Exemplu:** Pentru funcția din exemplul anterior, avem:
$f(x,y,z) = (x+\overline{y})(z\oplus\overline{y}) + y\overline{z} = (x+\overline{y})(z\overline{\overline{y}} + \overline{z}\,\overline{y}) + y\overline{z} =$
$(x+\overline{y})(yz+\overline{y}\,\overline{z}) + y\overline{z} = xyz + x\overline{y}\,\overline{z} + \overline{y}yz + \overline{y}\,\overline{y}\,\overline{z} + y\overline{z} =$
$xyz + x\overline{y}\,\overline{z} + \overline{y}\,\overline{z} + y\overline{z} = xyz + \overline{y}\,\overline{z} + y\overline{z} = xyz + (\overline{y}+y)\overline{z} = xyz + \overline{z} = xy + \overline{z}$
Astfel, ea poate fi implementată prin circuitul:




# Circuit general

## Circuit general:

O funcție booleană poate fi implementată printr-un circuit general, construit după un anumit tipar, pornind de la o formulă standard a funcției - de exemplu, scrierea ei ca sumă de produse sau ca FND.

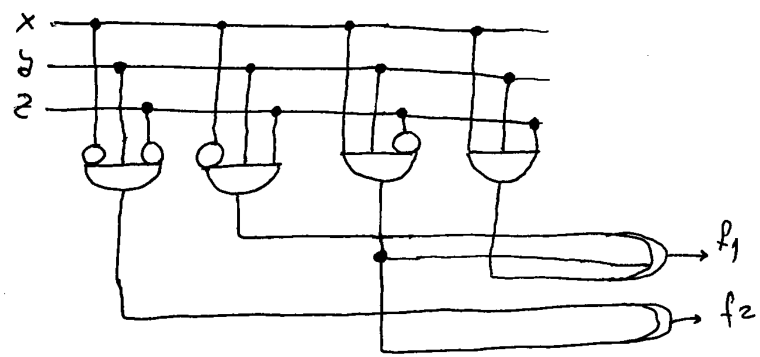
**Exemplu:** Fie funcția $f = (f_1, f_2) : {B_2}^3 \longrightarrow {B_2}^2$ ($f_1, f_2 : {B_2}^3 \longrightarrow B_2$) dată prin tabelul:

| $x$ | $y$ | $z$ | $f_1$ | $f_2$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

Din tabel rezultă că $f_1$, $f_2$ au respectiv următoarele FND:
$f_1(x,y,z) = \overline{x}yz + xy\overline{z} + xyz$
$f_2(x,y,z) = \overline{x}y\overline{z} + xy\overline{z}$

# Circuit general

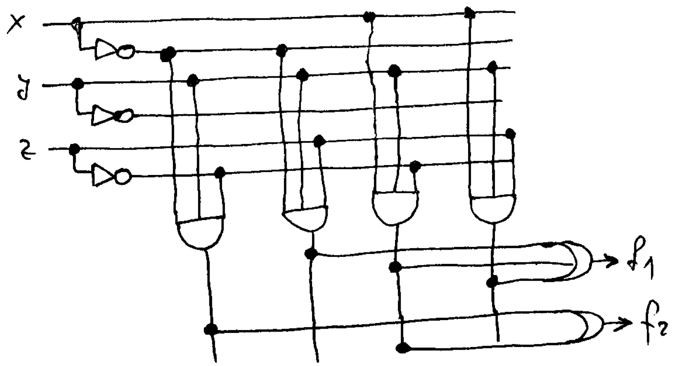
Atunci, funcția $f$ poate fi implementată prin circuitul:




Observăm că am folosit două porți "NOT" pentru a calcula aceeași valoare $\overline{x}$ și două porți "NOT" pentru a calcula aceeași valoare $\overline{z}$.

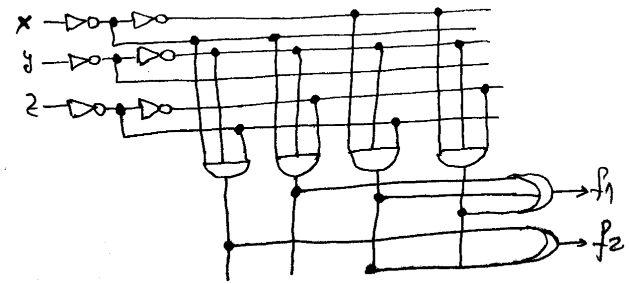
## Circuit general

Putem evita folosirea mai multor porți "NOT" pentru negarea aceleiași variabile construind circuitul astfel:




## Circuit general

O variantă a circuitului de mai sus, pe care o vom prefera în cele ce urmează, este următoarea:



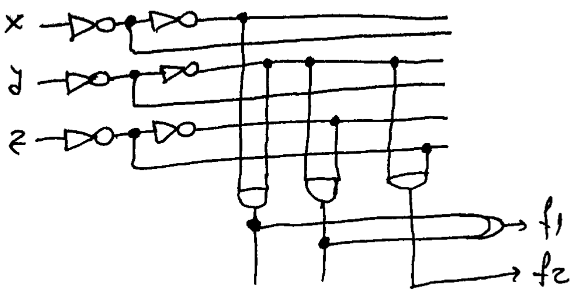

Plasarea porților "NOT" imediat la intrările $x$, $y$, $z$ are următoarele avantaje (datorate efectului de buffer al porților "NOT"):

- se împiedică pătrunderea semnalelor perturbate adânc în circuit (din porțile "NOT" respective va ieși un semnal clar 0 sau 1);
- se izolează circuitul de alte circuite aflate la intrare, evitând influiența negativă a impedanței acestora;
- semnalul este amplificat și astfel poate fi distribuit mai multor porți "AND" (un circuit cu $n$ variabile de intrare poate avea până la $2^n$ porți "AND").

## Circuit general

Putem minimiza scrierea funcției ca sumă de produse (să avem cât mai puține operații $+$ și $\cdot$) înainte de a o implementa prin circuitul general. Astfel, vom obține un circuit general minimal.

**Exemplu:** Pentru funcția $f$ din exemplul anterior, avem:
$f_1(x,y,z) = \overline{x}yz + xy\overline{z} + xyz = \overline{x}yz + xy(\overline{z}+z) = \overline{x}yz + xy =$
$(\overline{x}z + x)y = (z+x)y = xy + yz$
$f_2(x,y,z) = \overline{x}y\overline{z} + xy\overline{z} = (\overline{x}+x)y\overline{z} = y\overline{z}$
Atunci, $f$ poate fi implementată prin circuitul:




## Circuit general

Toate variantele de circuit general prezentate până acum, cu excepția primeia, au trei părți:
- un bloc care calculează variabilele și negațiile lor;
- un bloc care calculează produse "AND";
- un bloc care calculează sume "OR"

(prima variantă constructivă calculează negațiile în blocul al doilea).

Structura primului bloc depinde doar de numărul de variabile, în timp ce structura blocurilor al doilea și al treilea depinde de funcție.
Dacă ne imaginăm că implementăm funcțiile cu un număr fixat de variabile prin plăci de extensie inserate într-un soclu pe o placă de bază, primul bloc ar putea fi integrat în soclu (pe placa de bază), iar blocurile al doilea și al treilea pe placa de extensie.

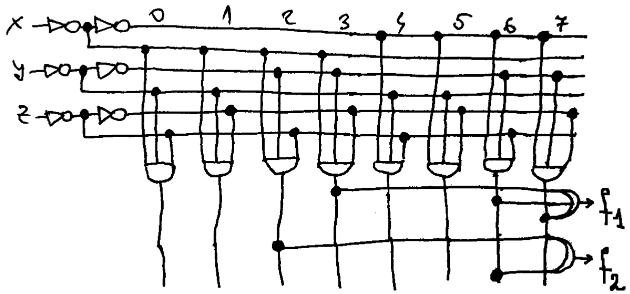
Putem standardiza și mai mult circuitul general și să integrăm mai mult în soclu dacă în blocul al doilea am calcula toate cele $2^n$ produse posibile cu cele $n$ variabile date (ele corespund liniilor din tabelul de valori al funcției) - atunci structura primelor două blocuri va depinde doar de numărul de variabile (iar ele vor putea fi integrate în soclu) și doar structura celui de-al treilea bloc va depinde de funcție (iar el va fi integrat în placa de extensie).
În acest caz, suma de produse implementată de circuit este FND.

# Circuit general

**Exemplu:** Pentru funcția $f$ din exemplele anterioare, ale cărei componente au, reamintim, respectiv următoarele FND:
$f_1(x,y,z) = \overline{x}yz + xy\overline{z} + xyz, \quad f_2(x,y,z) = \overline{x}y\overline{z} + xy\overline{z}$
obținem circuitul următor (am numerotat liniile din tabel, termenii din FND și produsele din blocul "AND" pentru a observa mai ușor corespondențele):

| | x | y | z | $f_1$ | $f_2$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 |
| 6 | 1 | 1 | 0 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 | 0 |

FND:
$f_1(x,y,z) = \overset{3}{\overline{x}yz} + \overset{6}{xy\overline{z}} + \overset{7}{xyz}$
$f_2(x,y,z) = \overset{2}{\overline{x}y\overline{z}} + \overset{6}{xy\overline{z}}$




# Circuit general

Putem desena mai rapid circuitul, observând următoarele proprietăți:

– Punctele de contact ale produselor din blocul "AND" sunt în concordanță cu aparițiile cu/fără ¯ ale variabilelor în termenii corespunzători din FND, care sunt în concordanță cu valorile 0/1 ale acestor variabile în tabel, care traduc în binar numerele de ordine ale liniilor tabelului; de aceea, ne putem gândi că transcriem în binar aceste numere de ordine direct prin puncte de contact (așa cum s-au transcris prin sistemele de ˍ și ¯ deasupra termenilor din FND): 0 înseamnă contact pe linia de jos, 1 înseamnă contact pe linia de sus (în perechea de linii care furnizează valoarea unei variabile și negația ei);

– Modul în care alternează valorile în tabel în coloanele variabilelor, cu perioade care se înjumătățesc odată cu scăderea semnificației variabilelor (în coloana variabilei celei mai semnificative, jumătate 0, jumătate 1, în coloana variabilei următoare ca semnificație, un sfert 0, un sfert 1, etc.) și care ne permite să completăm tabelul pe coloane, ne permite să plasăm punctele de contact în blocul "AND" pe linii: în fiecare produs, variabila cea mai semnificativă are jumătate din contacte pe linia de jos, jumătate pe linia de sus, variabila următoare ca semnificație are un sfert din contacte pe linia de jos, un sfert pe linia de sus, apoi iar un sfert pe linia de jos, un sfert pe linia de sus, etc..

– Sumele din blocul "OR" au punctele de contact la produsele care corespund liniilor din tabel unde componenta corespunzătoare a funcției are valoarea 1.

# Circuit general

Constatăm că orice funcție booleană se poate implementa printr-un circuit general, în oricare din variantele constructive prezentate mai sus, de aceea circuitele de acest tip au statut de **circuite universale**.

Totodată, vedem cum orice funcție booleană poate fi implementată printr-un 0-DS, ceea ce încheie justificarea echivalenței între 0-DS și funcții booleene, afirmată mai devreme.

# PLA și PROM

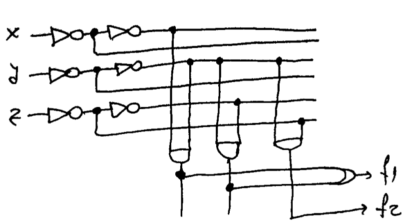
## PLA și PROM:

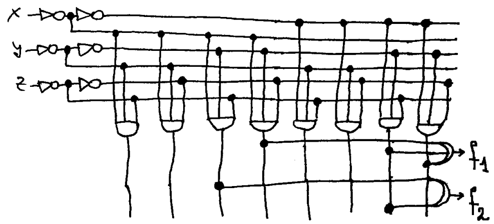
Circuitele generale prezentate mai devreme se pot desena folosind următoarele convenții grafice mai simple:
– Blocul care furnizează valorile variabilelor și negațiile lor se desenează la fel.
– Fiecare produs din blocul "AND" (care acum se mai numește și **planul "AND"**) se desenează printr-o linie verticală care are contacte pe aceleași linii ca și produsul.
– Fiecare sumă din blocul "OR" (care acum se mai numește și **planul "OR"**) se desenează printr-o linie orizontală care are contacte pe aceleași linii ca și suma.

Așadar, liniile desenate au semantici diferite, în funcție de poziția lor pe desen: cele orizontale din blocul de sus sunt linii de circuit, cele verticale din planul "AND" sunt produse, cele orizontale din planul "OR" sunt sume.

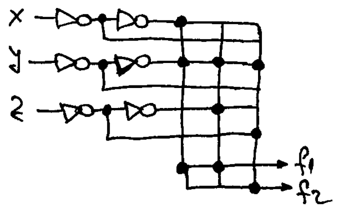
# PLA și PROM

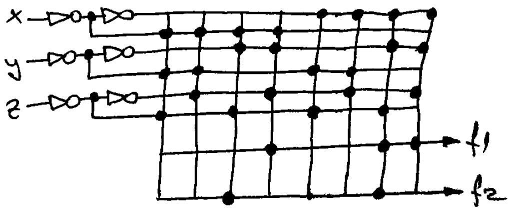
**Exemplu:** Următoarele circuite care implementează funcția $f$ din exemplele anterioare (cel cu număr minim de produse/sume și cel cu toate produsele posibile):




se vor desena astfel:




# PLA și PROM

Circuitele generale desenate după convențiile mai simple de mai sus s.n. **PLA (Programmable Logic Array)**; cel care are în planul "AND" toate produsele posibile cu variabilele date se mai numește și **PROM (Programmable Read-only Memory)** (deci un PROM este un caz particular de PLA). De obicei însă, denumirea de PLA este folosită doar pentru circuitul cu număr minim de produse/sume.

Din punct de vedere tehnic, având în vedere gradul ridicat de standardizare, aceste circuite se pot construi după tehnologii diferite, mai simple, decât circuitele alcătuite din porți și având o organizare oarecare.

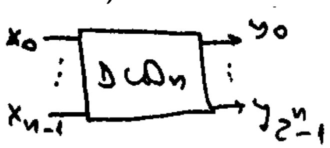
Întrucât pot implementa orice funcție booleană, PLA și PROM sunt circuite universale.

# Decodificator

## Decodificator:

Un **decodificator (decoder)** cu selector pe $n$ biți $DCD_n$, $n \geq 1$, este un circuit care transformă un cod numeric $k$ pe $n$ biți într-o alegere fizică, a liniei de ieșire cu numărul $k$ (prin care trimite 1).
De fapt, acesta este un caz particular de decodificator, numit **decodificator linie (line decoder)**.

Simbol:

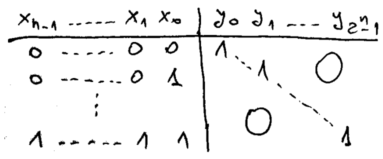



El primește ca intrare un sistem de valori $x_{n-1}, \ldots, x_0$, numit **selector**, care este reprezentarea în calculator a unui număr natural $k \in \{0, 2^n - 1\}$ ca întreg fără semn și furnizează ca ieșire un sistem de valori $y_0, \ldots, y_{2^n-1}$ a.î. $y_k = 1$ și $y_i = 0$ pentru $i \neq k$.

Liniile de ieșire $y_0, \ldots, y_{2^n-1}$ pot fi conectate la diverse echipamente (circuite) și astfel va fi activat echipamentul cu numărul $k$.
Deci, cu ajutorul unui decodificator, putem activa diverse echipamente, selectabile printr-un cod numeric.

# Decodificator

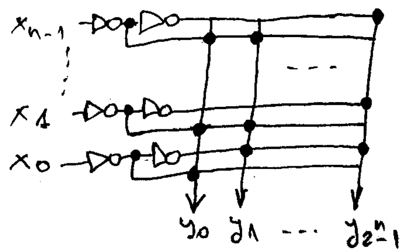
$DCD_n$ implementează o funcție $f : {B_2}^n \longrightarrow {B_2}^{2^n}$ având următorul tabel de valori:




Așadar, coloana fiecărei componente $y_k$ a funcției, $0 \leq k \leq 2^n - 1$, conține o singură valoare 1, anume pe linia $k$.

Atunci, dacă implementăm această funcție ca PROM, fiecare produs din planul "AND" va participa la o singură sumă.

În consecință, un decodificator este planul "AND" dintr-un PROM:



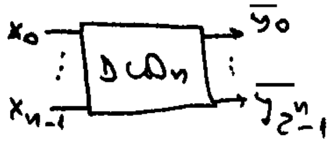
# Decodificator

Pentru $n = 1$ obținem **decodificatorul elementar (elementary decoder)**, *EDCD*; simbolizare:


Construcție:

# Decodificator

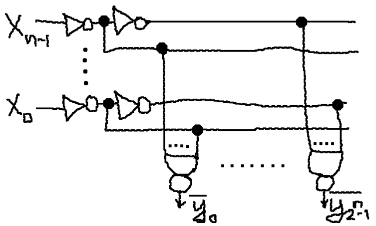
Uneori este utilă o variantă a decodificatorului, în care ieșirile sunt complementate:



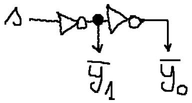

El transformă selectorul $(x_{n-1}, \ldots, x_0) = [k]_n^u$ în sistemul de valori $y_0, \ldots, y_{2^n-1}$ a.î. $y_k = 0$ și $y_i = 1$ pentru $i \neq k$.

# Decodificator

Pentru construcție, în planul "AND" al PROM-ului se înlocuiesc porțile "AND" cu porți "NAND":



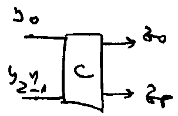
În cazul EDCD, se poate proceda astfel:



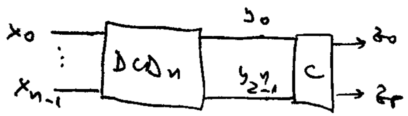
# Codificator

## Codificator:

Un $(2^n, p)$ - **codificator (encoder)** este un circuit cu $2^n$ intrări dintre care la fiecare moment doar una este activă (i.e. are valoarea 1) și care generează la ieșire o configurație binară oarecare de lungime $p$. Simbol:



Pentru a se garanta că dintre cele $2^n$ intrări la fiecare moment exact una este activă, codificatorul este însoțit întotdeauna de un decodificator:

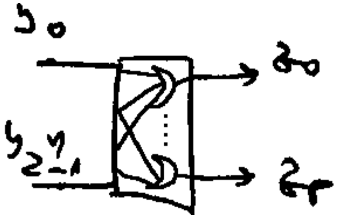



Circuitul rezultat transformă: $(x_{n-1}, \ldots, x_0) = [k]_n^u$, $0 \leq k \leq 2^n - 1 \Rightarrow$ $(y_0, \ldots, y_{2^n-1}) = (0, \ldots, 0, 1, 0, \ldots, 0)$ (1 este pe poziția $k$) $\Rightarrow (z_1, \ldots, z_p)$, unde pentru orice $1 \leq i \leq p$, $z_i = z_i(k) = z_i(x_{n-1}, \ldots, x_0) : {B_2}^n \longrightarrow B_2$ este o funcție booleană scalară oarecare, deci $z = (z_1, \ldots, z_p) : {B_2}^n \longrightarrow {B_2}^p$ este o funcție booleană vectorială oarecare.

# Codificator

Întrucât am văzut că orice funcție booleană se poate implementa într-un mod standard ca PROM iar decodificatorul este planul "AND" al acestuia, rezultă că codificatorul poate fi construit ca planul său "OR".

Mai exact, codificatorul se construiește ca un sistem de porți "OR", fiecare furnizând ca ieșire câte un $z_i$, $1 \leq i \leq p$, și primind ca intrare acei $y_k$ pentru care, dacă valoarea este 1, atunci și $z_i = 1$:



Așadar, codificatorul este planul "OR" dintr-un PROM.

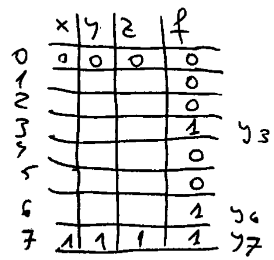
Întrucât am văzut că PROM-ul este un circuit universal, rezultă că și codificatorul (însoțit de decodificator) este un circuit universal, i.e. poate implementa orice funcție booleană.

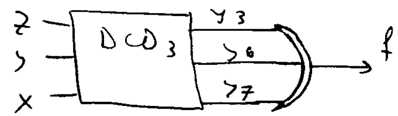
# Codificator

**Exemplu:** Implementați cu ajutorul unui codificator funcția booleană $f : {B_2}^3 \longrightarrow B_2$ dată prin tabelul:

| $x$ | $y$ | $z$ | $f$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Codificatorul va conține un "OR" ce furnizează ca ieșire $f$ și are ca intrări ieșirile $y_k$ al $DCD_3$ ce corespund liniilor cu numerele $k$ din tabel pentru care $f = 1$:

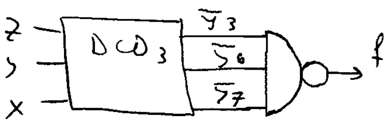




# Codificator

Dacă folosim un decodificator cu ieșirile negate, în construcția codificatorului se înlocuiesc porțile "OR" cu porți "NAND".

**Exemplu:** Pentru funcția $f$ din exemplul anterior, obținem implementarea:

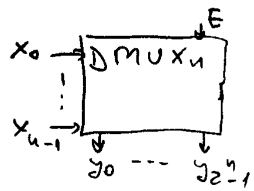


# Demultiplexor

## Demultiplexor:

Un **demultiplexor (demultiplexer)** cu selector pe $n$ biți $DMUX_n$, $n \geq 1$, este un comutator de tip "one into many", care care poate conecta o intrare unică la o ieșire selectabilă printr-un cod numeric.

Simbol:



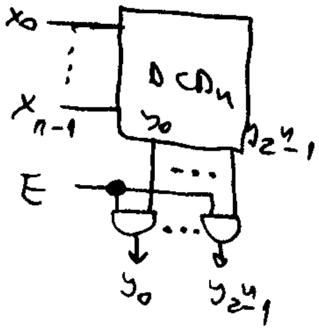
El primește ca intrare:
- un sistem de valori $x_{n-1}, \ldots, x_0$, numit **selector**, care este reprezentarea în calculator a unui număr natural $k \in \{0, 2^n - 1\}$ ca întreg fără semn;
- o valoare $E \in \{0, 1\}$;

și furnizează ca ieșire:
- un sistem de valori $y_0, \ldots, y_{2^n-1}$ a.î. $y_k = E$ și $y_i = 0$ pentru $i \neq k$.

# Demultiplexor

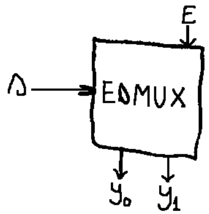
Constatăm că demultiplexorul este o generalizare a decodificatorului; mai exact, un $DCD_n$ este un $DMUX_n$ unde am fixat $E = 1$.

De aceea, demultiplexorul se construiește ușor cu ajutorul unui decodificator - se conjugă toate ieșirile decodificatorului cu $E$:

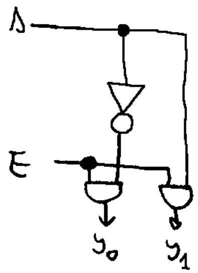



# Demultiplexor

Pentru $n = 1$ obținem **demultiplexorul elementar (elementary demultiplexer)**, $EDMUX$; simbolizare:



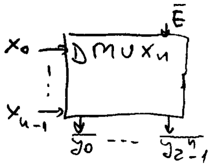

Construcție:



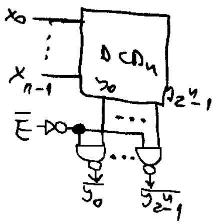

# Demultiplexor

Putem construi o variantă a demultiplexorului, în care intrarea $E$ și ieșirile $y_0, \ldots, y_{2^n-1}$ sunt negate.

Simbolizare:




Construcție:



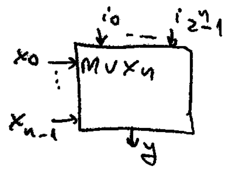

Această variantă are avantajul că și intrarea $E$ va fi protejată iar semnalul va fi amplificat înainte de combinarea cu cele $2^n$ ieșiri ale decodificatorului, datorită efectului de buffer al porții "NOT".

# Multiplexor

## Multiplexor:

Un **multiplexor (multiplexer)** cu selector pe $n$ biți $MUX_n$, $n \geq 1$, este un comutator de tip "many into one", care care poate conecta o intrare selectabilă printr-un cod numeric la o ieșire unică.

Simbol:



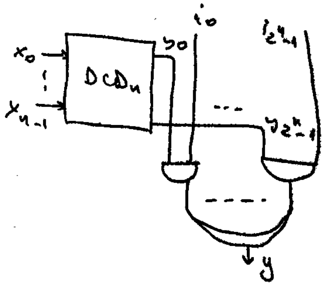

El primește ca intrare:

- un sistem de valori $x_{n-1}, \ldots, x_0$, numit **selector**, care este reprezentarea în calculator a unui număr natural $k \in \{0, 2^n - 1\}$ ca întreg fără semn;
- un sistem de valori oarecare $i_0, \ldots, i_{2^n-1}$;

și furnizează ca ieșire: $i_k$.

# Multiplexor

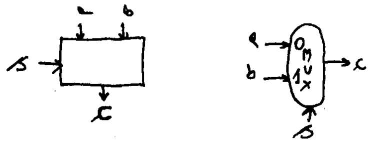
Construcție:



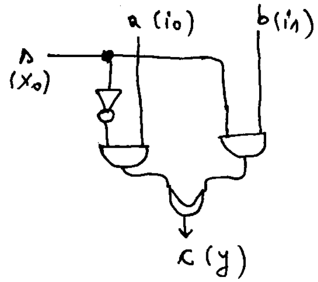

Decodificatorul transformă selectorul $k = (x_{n-1}, \ldots, x_0)$ în sistemul de ieșiri $0, \ldots, 0, 1, 0, \ldots, 0$, unde 1 este pe poziția $k$; aceste ieșiri sunt conjugate cu respectiv intrările $i_0, \ldots, i_{2^n-1}$, rezultând sistemul de valori $0, \ldots, 0, i_k, 0, \ldots, 0$, unde $i_k$ este pe poziția $k$; aceste valori intră într-o poartă "OR", care efectuează $0 + \cdots + i_k + \cdots + 0$, rezultând în final $i_k$.

# Multiplexor

Pentru $n = 1$ obținem **multiplexorul elementar (elementary multiplexer)**, *EMUX*; variante de simbolizare (în primul caz subînțelegem că intrarea corespunzătoare selectorului $s = 0$ este în stânga):

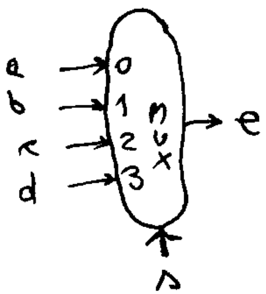


Construcție:



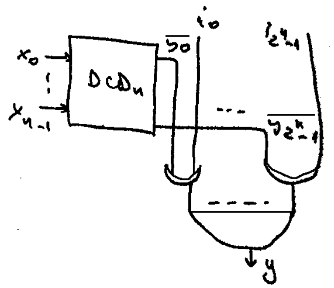

# Multiplexor

De asemenea, vom folosi și $MUX_2$, pentru care avem și simbolizarea:




# Multiplexor

Dacă folosim un decodificator cu ieșirile negate, în construcția multiplexorului se interschimbă porțile nou adăugate "AND" și "OR":




Acum, decodificatorul transformă selectorul $k = (x_{n-1}, \ldots, x_0)$ în sistemul de ieșiri $1, \ldots, 1, 0, 1, \ldots, 1$, unde 0 este pe poziția $k$; aceste ieșiri sunt disjunse cu respectiv intrările $i_0, \ldots, i_{2^n-1}$, rezultând sistemul de valori $1, \ldots, 1, i_k, 1, \ldots, 1$, unde $i_k$ este pe poziția $k$; aceste valori intră într-o poartă "AND", care efectuează $1 \cdot \ldots \cdot i_k \cdot \ldots \cdot 1$, rezultând în final $i_k$.

# Multiplexor

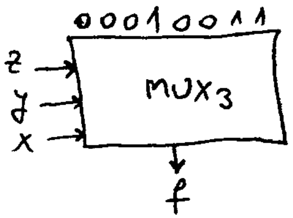
Multiplexorul este și el un circuit universal, deoarece putem implementa orice funție booleană scalară cu un multiplexor.

**Exemplu:** Implementați funcția booleană următoare cu un multiplexor:

$f : B_2^3 \to B_2$ dată prin tabelul:

| x | y | z | f |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Implementare:



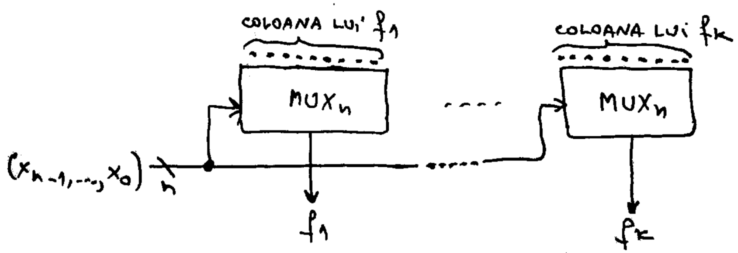

# Multiplexor

Intuitiv, fiecare sistem de valori ale variabilelor $x, y, z$, care este reprezentarea unui număr natural $0 \leq k \leq 7$, selectează:
- din tabel, valoarea lui $f$ din linia $k$;
- cu muliplexorul, intrarea $i_k$ de sus.

Așadar, multiplexorul implementează pe $f$ dacă plasăm coloana valorilor lui $f$ ca sistem de intrări $i_0, \ldots, i_7$; întrucât sistemul de valori $0, 0, 0$ ale variabilelor selectează valoarea din linia de sus a tabelului și intrarea $i$ din stânga a multiplexorului, coloana valorilor lui $f$ trebuie plasată orizontal, cu partea de sus spre stânga; notăm că variabila cea mai semnificativă, $x$, a fost desenată, conform convențiilor adoptate, în tabel în stânga și la multiplexor jos.

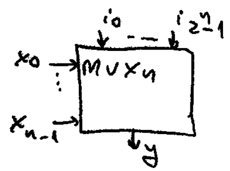
# Multiplexor

O funcție booleană vectorială $f : B_2{}^n \longrightarrow B_2{}^k$, $k > 1$, poate fi implemetată printr-un sistem de $k$ multiplexoare $MUX_n$, fiecare calculând câte una din componentele funcției, $f_1, \ldots, f_k$; liniile corespunzătoare variabilelor lui $f$ intră simultan ca selector în toate multiplexoarele, iar acestea au ca sisteme de intrări $i$ coloanele de valori ale componentelor lui $f$ pe care le calculează:

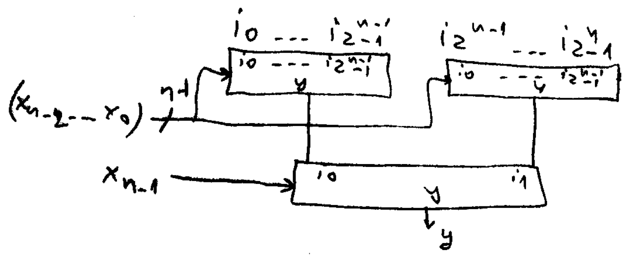



# Multiplexor

$MUX_n$ admite și o construcție recursivă; mai exact:




se poate construi ca:



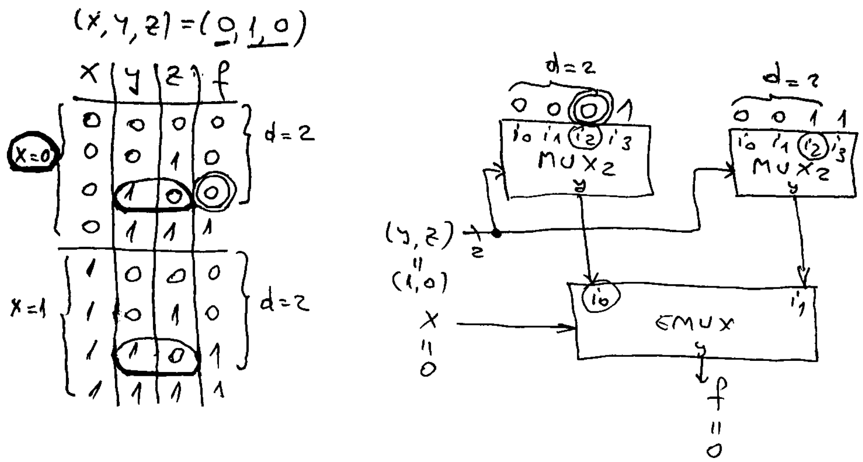
# Multiplexor

În interiorul dreptunghiurilor am notat intrările/ieșirile proprii ale multiplexoarelor (parametrii formali), iar lângă dreptunghiuri am notat liniile conectate la ele (parametri actuali).

De exemplu, intrarea $i_{2^{n-1}}$ a lui $MUX_n$ este conectată la intrarea $i_0$ a lui $MUX_{n-1}$ din dreapta.

# Multiplexor

Echivalența celor două circuite este ușor de înțeles dacă facem o comparație cu tablul de valori al funcției implementate; vom ilustra pentru funcția din exemplul anterior, presupunând că $(x, y, z) = (0, 1, 0)$:




# Multiplexor

Ieșirea $y$ a lui $MUX_3$ este valoarea finală a lui $f$; după valoarea variabilei celei mai semnificative, $x$, ea este aleasă din prima sau a doua jumătate a tabelului, respectiv din ceea ce furnizează primul sau al doilea $MUX_2$; întrucât $x = 0$, se alege din prima jumătate a tebelului, respectiv ce furnizează $MUX_2$ din stânga.

Sistemul valorilor celor mai puțin semnificative variabile, $(y, z)$, apare de două ori în tabel, o dată în prima jumătate, o dată în a doua jumătate, și în fiecare caz selectează valoarea lui $f$ de pe o linie aflată la aceeași distanță $d$ față de începutul jumătății; corepunzător, sistemul $(y, z)$ intră ca selector în ambele $MUX_2$ și în fiecare caz selectează intrarea $i$ cu același indice $d$; întrucât $(y, z) = (1, 0)$, se alege din fiecare jumătate a tabelului valoarea lui $f$ din linia 2 (numărând de la 0) și de la fiecare $MUX_2$ intrarea $i_2$ ($d = 2 = \overline{10}_2$). Întrucât variabila cea mai semnificativă este $x = 0$, prima dintre cele două valori, adică 0, este emisă ca valoare finală.

# Multiplexor

Proprietatea algebrică pe care se bazează construcția recursivă de mai sus este teorema care dă o descriere recursivă a unei funcții booleene scalare cu $n$ variabile prin două funcții booleene scalare cu $n - 1$ variabile:

$\forall x_{n-1}, \ldots, x_0 \in B_2$,
$f(x_{n-1}, \ldots, x_0) = \overline{x_{n-1}} f(0, x_{n-2}, \ldots, x_0) + x_{n-1} f(1, x_{n-2}, \ldots, x_0)$

$MUX_n$ construit recursiv implementează $f : {B_2}^n \longrightarrow B_2$;

$MUX_{n-1}$ din stânga (a cărui ieșire este selectată de $EMUX$ pentru $x_{n-1} = 0$) implementează $f_0 : {B_2}^{n-1} \longrightarrow B_2$, $f_0(x_{n-2}, \ldots, x_0) = f(0, x_{n-2}, \ldots, x_0)$;
$MUX_{n-1}$ din dreapta (a cărui ieșire este selectată de $EMUX$ pentru $x_{n-1} = 1$) implementează $f_1 : {B_2}^{n-1} \longrightarrow B_2$, $f_1(x_{n-2}, \ldots, x_0) = f(1, x_{n-2}, \ldots, x_0)$.

Formula de recursie din teoremă este funcția implementată de un $EMUX$: $EMUX(s, a, b) = \bar{s}a + sb$.

# Multiplexor

**Observație:** Și alte circuite $0-DS$, ca de exemplu $DCD_n$ sau $DMUX_n$, admit o construcție recursivă (exercițiu).

# Multiplexor

Putem continua construcția recursivă a lui $MUX_n$ de mai devreme, exprimând fiecare $MUX_{n-1}$ prin câte două $MUX_{n-2}$ și un *EMUX*, ș.a.m.d., până obținem un circuit, asemănător unui arbore, alcătuit doar din *EMUX*-uri.

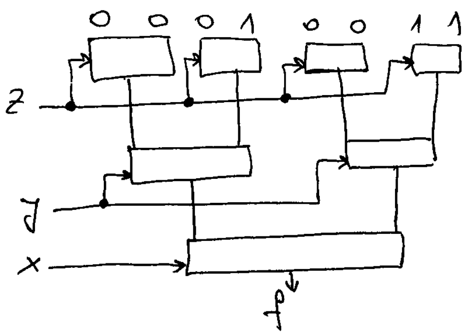
Astfel, întrucât orice funcție booleană scalară se poate implementa cu un multiplexor iar orice multiplexor se poate înlocui cu un arbore de *EMUX*-uri, rezultă că orice funcție booleană scalară se poate implementa cu un circuit alcătuit doar din *EMUX*-uri.
Deci, și *EMUX* este circuit universal.

**Observație:** Alte circuite universale sunt *NAND* și *NOR* (putem implementa orice funcție booleană folosind doar *NAND*-uri sau doar *NOR*-uri) - exercițiu.

# Multiplexor

**Exemplu:** Funcția din exemplul precedent se poate implementa cu *EMUX*-uri astfel:

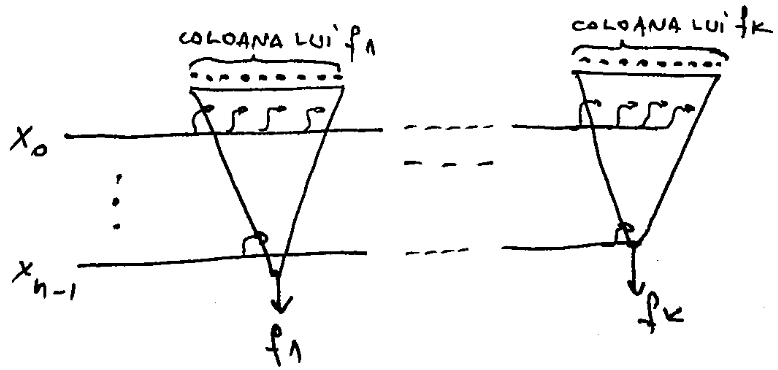
| x | y | z | f |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



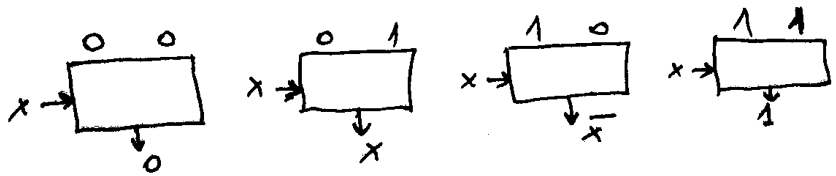

Dacă păstrăm convențiile anterioare privind poziția intrărilor și ieșirilor pe desenul *EMUX*-urilor, nu mai este nevoie să notăm aceste intrări și ieșiri.

# Multiplexor

O funcție booleană vectorială $f : {B_2}^n \longrightarrow {B_2}^k$, $k > 1$, poate fi implemetată printr-un sistem de $k$ arbori de *EMUX*-uri, fiecare calculând câte una din componentele funcției, $f_1, \ldots, f_k$; fiecare linie corespunzătoare unei variabile a lui $f$ intră ca selector în *EMUX*-urile de pe un același rând în toți arborii, iar aceștia au ca sisteme de intrări $i$ coloanele de valori ale componentelor lui $f$ pe care le calculează:



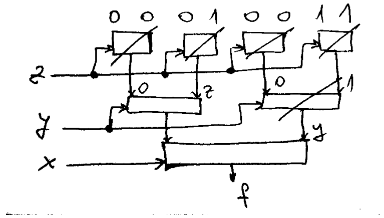
Numărul *EMUX*-urilor prin care se implementează o funcție booleană poate fi redus, dacă observăm că:



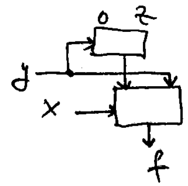
Astfel, dacă acceptăm și prezența porților "NOT", *EMUX*-urile aflate pe primul rând al arborilor dispar, iar uneori dispar și *EMUX*-uri de pe rândurile următoare.



**Exemplu:** Reduceți la maxim numărul de *EMUX* prin care se implementează funcția $f$ din exemplele anterioare:



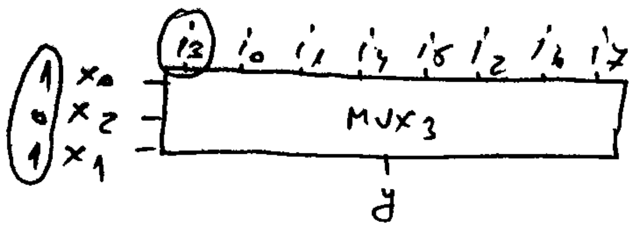
Desenul care păstrează doar *EMUX*-urile rămase este:



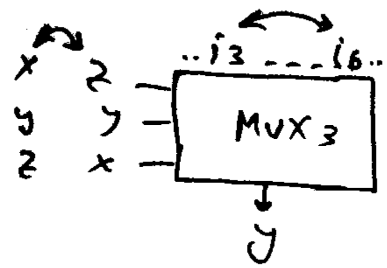
**Sfat:** Desenul circuitului redus este bine să fie realizat de jos în sus, deoarece avem o perspectivă mai clară care linii trebuie șterse sau redirecționate.

**Observație:**

În notațiile și simbolizările grafice folosite până acum, dacă dăm nume indexate variabilelor, intrărilor, ieșirilor, nu mai este nevoie să respectăm o anumită ordine de scriere/desenare a acestora. De exemplu, în cazul unui $MUX_3$, vom ști că intrarea $i_3$ corespunde valorii selectorului $x_2 = 0$, $x_1 = 1$, $x_0 = 1$, indiferent unde sunt scrise/desenate aceste linii:




Dacă folosim nume neindexate, contează ordinea semnificațiilor variabilelor și cea de scriere/desenare în funcție de semnificație. De exemplu, dacă inversăm ordinea liniilor selectorului unui $MUX_3$, intrarea $i_3$ se interschimbă cu $i_6$:



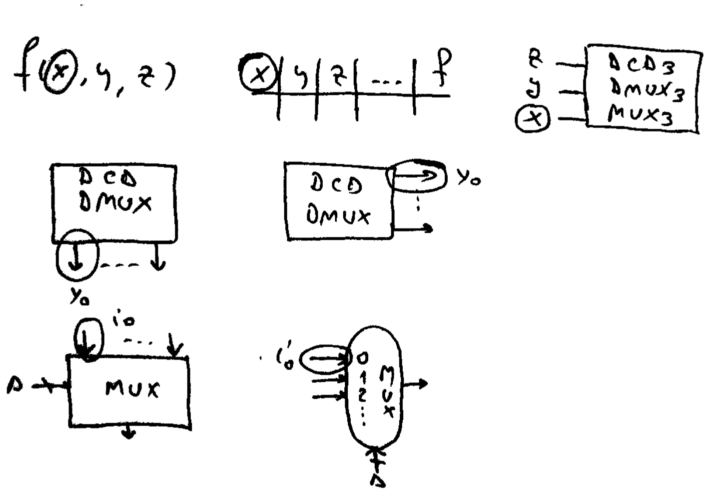

Oricare convenție de ordonare este bună, dar trebuie aleasă una și păstrată cu consecvență.

Convenția de ordonare folosită în acest curs este:

- variabila cea mai semnificativă este scrisă în stânga notației cu paranteze a funcției, în coloana din stânga a tabelului de valori și în partea de jos a selectorului blocurilor decodificator, codificator, demultiplexor, multiplexor;

- ieșirea $y$ corespunzătoare valorii $0 = (0, \ldots, 0)$ a selectorului blocurilor decodificator și demultiplexor este scrisă în stânga sau sus;

- intrarea $i$ corespunzătoare valorii $0 = (0, \ldots, 0)$ a selectorului blocului multiplexor este scrisă în stânga sau sus.




# Cicluri

Sistemele 1-DS sunt sisteme 0-DS închise prin exterior printr-un ciclu (mai general, pot avea mai multe cicluri, dar un singur nivel de cicluri).

Apare un prim grad de autonomie a circuitului, prin **stare** - ea depinde doar parțial de intrare, ceea ce conduce la o independență parțială a ieșirii de intrare. Evoluția ieșirilor rămâne sub controlul intrărilor, dar stările dau o autonomie parțială.

O altă caracteristică a 1-DS este că **pot păstra informația** de intrare pentru o perioadă determinată de timp, proprietate specifică **memoriilor** - de aceea, sistemele 1-DS sunt folosite pentru a construi diverse circuite de memorie (RAM, regiștrii, etc.).

Informația memorată este pusă la dispoziția diverselor circuite în anumite faze de lucru, de aceea trebuie să existe o sincronizare a operațiilor în desfășurare, a.î. ele să poată conlucra corect și eficient.
Pentru aceasta se folosește un dispozitiv general de control al circuitelor, **ceasul (CK)** - el este un circuit bistabil ce asigură o discretizare a timpului de calcul, făcând posibilă definirea unor noțiuni temporale, ca: moment actual, tact, istoric, dezvoltare ulterioară, etc.

# Cicluri

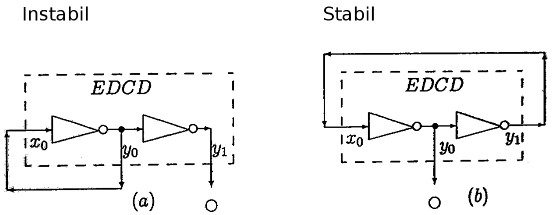
Există două tipuri de cicluri ce închid un CLC:

- **Cicluri stabile:** conțin un număr par de complementări;
  ele generează o stare stabilă și sunt utile în construcția circuitelor digitale.
- **Cicluri instabile:** conțin un număr impar de complementări;
  ele generează o stare instabilă la ieșire și pot fi folosite la construcția ceasului.

Pentru a fi stabil, un circuit trebuie să treacă printr-un număr par de complementări pentru toate combinațiile binare aplicate la intrare (altfel, pentru anumite combinații, se destabilizează).

# Cicluri

**Exemplu:** În figura de mai jos, circuitul (a) conține un ciclu instabil, circuitul (b) conține un ciclu stabil:

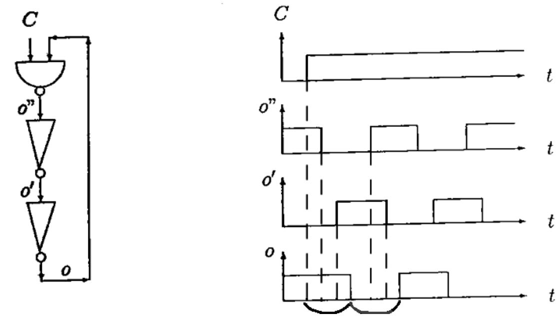



În cazul (a), dacă avem de exemplu starea $y_0 = 0 = x_0$, atunci vom obține la ieșire $y_1 = 0$ și noua stare $y_0 = 1 = x_0$, apoi vom obține la ieșire $y_1 = 1$ și noua stare $y_0 = 0 = x_0$, etc.; astfel, cele două ieșiri ale decodificatorului sunt instabile, comutând de pe 0 pe 1 și invers.
Momentul de schimbare a valorii de ieșire (din 0 în 1 și invers) definește **frecvența** circuitului și s.n. **tact**.

În cazul (b), dacă avem de exemplu $y_1 = 0 = x_0$, atunci starea $y_1$ va fi fixată la valoarea 0 iar la ieșire vom avea constant $y_0 = 1$; similar, dacă $y_1 = 1 = x_0$ (la ieșire vom avea constant $y_0 = 0$); deci, acest circuit are două stări stabile.
Deocamdată însă nu știm cum să comutăm între stări, circuitul neavând o intrare prin care să putem controla schimbarea.

# Cicluri

**Exemplu:** Considerăm următorul circuit cu 3 niveluri de complementare (ciclu instabil):

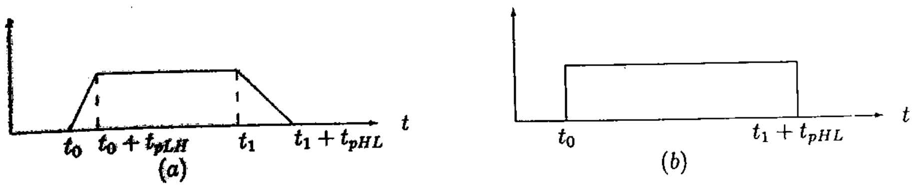



Dacă la intrare aplicăm comanda $C = 0$, pe fiecare linie valoarea semnalului rămâne constantă, circuitul își conservă starea.
Dacă aplicăm comanda $C = 1$, circuitul generează un semnal periodic.
Comportarea circuitului este descrisă de diagrama din dreapta.
Constatăm că este nevoie de un anumit timp pentru ca semnalul să se propage prin circuit, timp care depinde de structura circuitului și natura fenomenelor fizice folosite (am marcat cu arce intervalele de timp necesare propagării semnalului la două ciclări succesive).

# Cicluri

Așa cum am văzut și în exemplul precedent, schimbarea stării unui circuit nu este instantanee și depinde de anumite caracteristici fizice și structurale ale circuitului.

Vom nota cu $t_{pLH}$ intervalul de timp în care un circuit comută de la starea 0 la starea 1 și cu $t_{pHL}$ intervalul de timp de trecere de la starea 1 la starea 0.
Ambele valori sunt numere $\geq 0$ și considerate constante pentru un circuit; ele nu sunt neapărat egale între ele.




Situația reală este cea din figura (a) de mai sus.
Uneori se consideră o situație ipotetică, de schimbare instantanee a stărilor, iar evoluția se aproximează ca în figura (b).

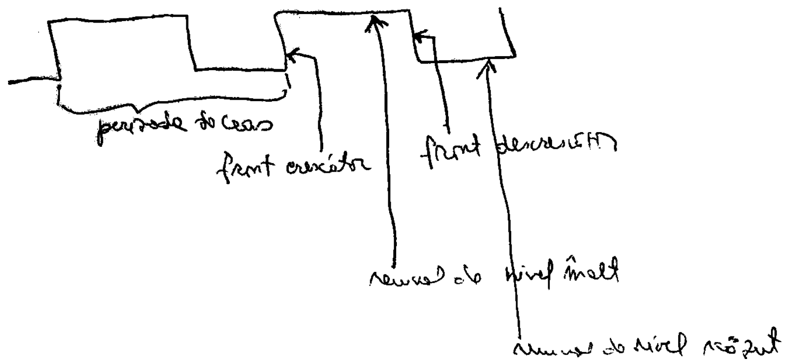
# Cicluri

Ceasul produce un semnal autonom cu perioadă (frecvență) fixă.

El se folosește în logica secvențială pentru a decide momentul în care trebuie actualizat un element ce conține stare.

Un sistem acționat cu ceas se mai numește și **sistem sincronizat**.

Semnalul de ceas are următoarele componente:



## Cicluri

La circuitele controlate de ceas, schimbările de stare se pot produce:

- fie în intervalul când semnalul este de nivel înalt (în acest interval modificarea intrărilor determină modificarea stării ieșirii);
  notăm acest lucru prin:
- fie pe un front de ceas, crescător (ascendent) sau descrescător (descendent);
  aceasta s.n. **acționarea pe frontul ceasului** și se notează: , respectiv
  acționarea pe front este mai bună, deoarece se poate preciza mai exact momentul instalării noii stări.

În metodologia acționării pe front, frontul crescător / descrescător care determină producerea schimbărilor de stare s.n. **front activ**.

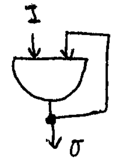
Alegerea lui depinde de tehnologia implementării și nu afectează conceptele implicate în proiectarea logicii.

Constrângerea principală într-un sistem sincronizat este că semnalul ce trebuie scris în elementele de stare trebuie să fie valid (în particular stabil, să nu se mai modifice până nu se modifică intrările) la apariția frontului de ceas activ.
De aceea, perioada ceasului trebuie să fie suficient de lungă a.î. semnalele respective să se stabilizeze (există o limită inferioară a perioadei).

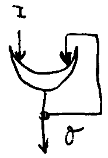
## Zăvor elementar

În continuare, prezentăm principalele circuite cu un ciclu intern:

ZĂVOR ELEMENTAR (LATCH):



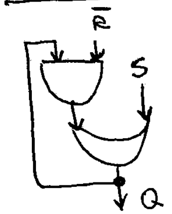

I=1 ⇒ CONSERVĂ STAREA
I=0 ⇒ TRECE ÎN STAREA 0
DECI, POATE FI COMANDAT SĂ COMUTE ÎN STAREA 0 ȘI ZĂVORĂȘTE 0
ESTE ACTIV LA FRECVENȚĂ JOASĂ



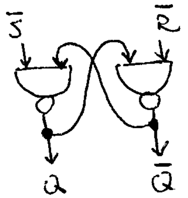

POATE FI COMANDAT SĂ COMUTE ÎN STAREA 1 ȘI ZĂVORĂȘTE 1
ACTIV LA FRECVENȚĂ ÎNALTĂ

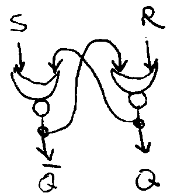
## Zăvor elementar eterogen

ZĂVOR ELEMENTAR ETEROGEN:




de ⟺ MORGAN




OBS: AL DOILEA N-ARE "‾", DAR Q NU ESTE SUB S

$Q = S + \bar{R}Q$

OBS: INTRĂRILE SUNT $\left\{ \begin{array}{l} \bar{R} \text{ ACTIV LA FRECVENȚĂ JOASĂ} \\ S \text{ ——— " ——— ÎNALTĂ} \end{array} \right.$

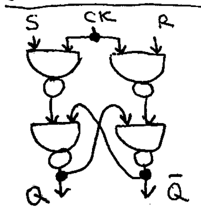
POSIBILITĂȚI DE FUNCȚIONARE:

| S | R | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 1 | 0 | 1 (SET) |
| 0 | 1 | 0 (RESET) |
| 1 | 1 | NEDEFINITĂ |

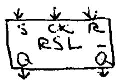
DEZAVANTAJE:
- NU ȘTIE SĂ INVERSEZE STAREA
- NU POATE EVITA HARDWARE INTRAREA 1-1

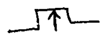
## Zăvor elementar cu ceas

ZĂVOR ELEMENTAR CU CEAS (RSL: RESET-SET LATCH)




SIMBOL:



EX: DACĂ TREBUIE SETAT, FAC S=1, R=0, APOI CK=1 PENTRU UN TIMP tpLH

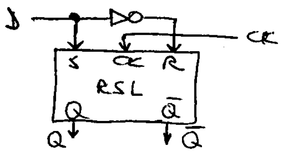
CÂȘTIG: { • NU MAI TREBUIE COMPLEMENTATE INTRĂRILE R, S
• IEȘIRILE NU MAI SUNT INVERSATE (Q ESTE SUB S)

OBS: PE NIVELUL ACTIV AL CK ZĂVORUL ESTE TRANSPARENT, I.E. ÎȘI POATE SCHIMBA STAREA DACĂ MODIFIC S, R.

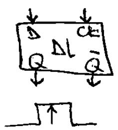
DEZAVANTAJE: { • CÂND CK=1 POATE COMUTA DE MAI MULTE ORI
• NU POATE EVITA INTRAREA 1-1

## Zăvor de date

ZĂVOR DE DATE (DL: DATA LATCH)



SIMBOL:



CÂȘTIG: • ELIMIN INTRAREA 1-1

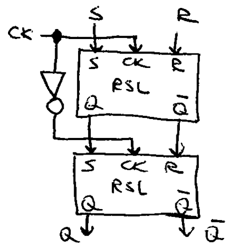
OBS: MEREU Q=D (IEȘIREA URMEAZĂ PERMANENT INTRAREA ⇒ AUTONOMIE REDUSĂ)

DEZAVANTAJ: • ÎNCĂ POATE COMUTA DE MAI MULTE ORI ÎN TIMPUL UNUI TACT

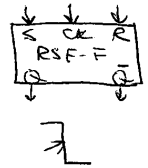
## Structura master-slave

STRUCTURA MASTER-SLAVE (RSF-F: RESET-SET FLIP-FLOP):

SE BAZEAZĂ PE UN CIRCUIT CU 2 STĂRI (NUMIT FLIP-FLOP) CARE COMUTĂ SINCRONIZAT CU ARCUL (ÎN SUS SAU JOS) SEMNALULUI DE CEAS.



SIMBOL:



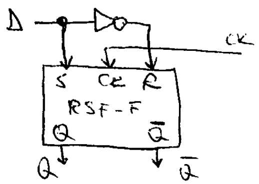
OBS: PRIMUL ZĂVOR ESTE TRANSPARENT LA NIVELUL SUPERIOR (PRIMA 1/2 TACT), AL DOILEA LA NIVELUL INFERIOR (A 2-A 1/2 TACT); PER TOTAL, CIRCUITUL POATE COMUTA DOAR O DATĂ PE TACT, IAR IEȘIREA FINALĂ DATĂ PE FRONTUL DESCENDENT

CÂȘTIG: • NU MAI POATE COMUTA DE MAI MULTE ORI LA UN CICLU

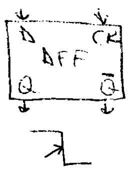
DEZAVANTAJ: • NU EVITĂ INTRAREA 1-1

## Flip-flop cu întârziere

FLIP-FLOP CU ÎNTÂRZIERE (DFF: DELAY FLIP-FLOP)

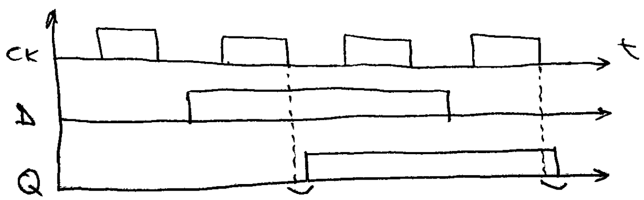


SIMBOL:



CÂȘTIG: • SE ELIMINĂ INTRAREA 1-1 LA S-R

Obs: IEȘIREA Q COPIAZĂ D CU ÎNTÂRZIERE DE 1 TACT (MAI EXACT, IEȘIREA SE CULEGE LA PRIMUL SFÂRȘIT DE TACT DE DUPĂ MODIFICAREA LUI D):




DFF ESTE UN CIRCUIT IMPORTANT (ARE APLICAȚII MULTE).

CELE MAI SIMPLE CIRCUITE DE MEMORIE SUNT:

- ZĂVOARE (LATCH)
  - NEACȚIONATE DE CEAS
    - ZĂVOR ELEMENTAR
    - ZĂVOR ELEMENTAR ETEROGEN
  - ACȚIONATE DE CEAS
    - RSL
    - DL
- BISTABILI (FLIP-FLOP) (SUNT ACȚIONAȚI DE CEAS)
  - RSF-F
  - DFF

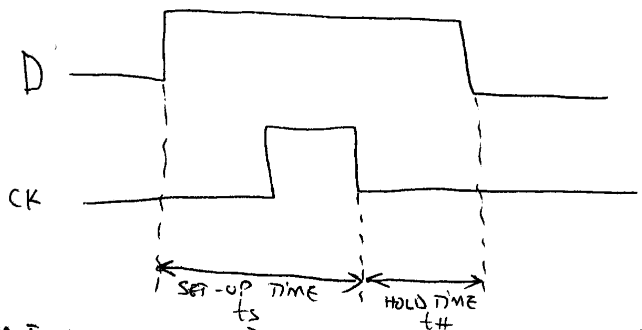
ATÂT LA ZĂVOARELE ACȚIONATE DE CEAS CÂT ȘI LA BISTABILI, IEȘIREA ESTE EGALĂ CU VALOAREA STĂRII MEMORATE ÎN ELEMENT.

DIFERENȚA ÎNTRE ELE ESTE MOMENTUL LA CARE CEASUL PROVOACĂ SCHIMBAREA STĂRII.

- LA ZĂVOARELE ACȚIONATE DE CEAS STAREA SE SCHIMBĂ ORI DE CÂTE ORI INTRĂRILE SE SCHIMBĂ ȘI CEASUL ESTE ACTIVAT (ADICĂ AVEM UN INTERVAL DE TRANSPARENȚĂ)
- LA BISTABILI STAREA SE SCHIMBĂ DOAR PE FRONTUL DE CEAS (LA NOI )

INTRAREA TREBUIE SĂ FIE TOTUŞI VALIDĂ (STABILĂ) PENTRU UN INTERVAL DE TIMP ÎNAINTE (TIMPUL DE STABILIRE - SET-UP TIME) ŞI DUPĂ (TIMPUL DE MENŢINERE - HOLD TIME) FRONTULUI

ILUSTRARE ÎN CAZUL DFF:

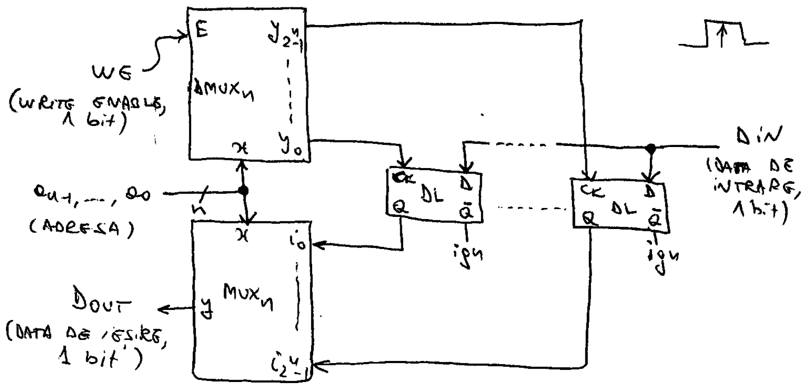



DACĂ NU SE RESPECTĂ ACESTE INTERVALE DE STABILITATE, IEŞIREA BISTABILULUI POATE SĂ NU FIE PREVIZIBILĂ

HOLD TIME DE OBICEI ESTE 0 SAU FOARTE MIC, DECI NU REPREZINTĂ UN MOTIV DE ÎNGRIJORARE.

# RAM

MEMORIA RAM: ESTE O EXTENSIE PARALELĂ DE 1-DS




EXEMPLIFICAREA S-A FĂCUT PENTRU UN RAM CU LOCAŢII DE 1 BIT; PENTRU UN RAM CU LOCAŢII PE K BIŢI, FACEM O EXTENSIE PARALELĂ DE K RAM-URI CU LOCAŢII DE 1 BIT; WE ŞI A VOR FI COMUNE, DIN ŞI DOUT VOR FORMA DATELE DE I/O DE K BIŢI.

# RAM

FUNCŢIONARE:

- LA CITIRE: $a_{n-1}, \ldots, a_0$ SELECTEAZĂ PRIN MUX O CELULĂ ŞI REDĂ PRIN DOUT INFORMAŢIA STOCATĂ ACOLO
- LA SCRIERE: $a_{n-1}, \ldots, a_0$ SELECTEAZĂ PRIN DMUX O CELULĂ ŞI TRIMITE ACOLO WE CA SEMNAL DE CEAS; DIN SE DISTRIBUIE LA TOATE CELULELE, DAR VA INTRA DOAR ÎN ACEEA.

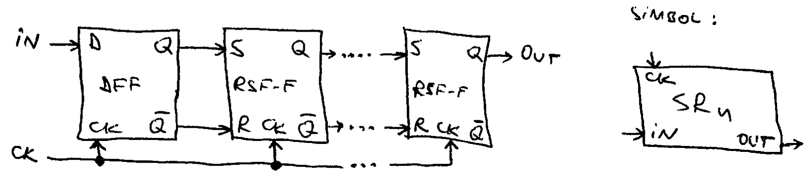
OBS: - RAM ESTE UN CIRCUIT SIMPLU ŞI SE POATE CONSTRUI UŞOR LA DIMENSIUNI MARI; EXISTĂ VARIANTE ŞI OPTIMIZĂRI - A SE VEDEA CARTEA P&H: SDRAM, DRAM, etc.
- RAM ADMITE ŞI O DEFINIŢIE RECURSIVĂ, PLECÂND DE LA DEFINIŢIA RECURSIVĂ A COMPONENTELOR SALE (EXERCIŢIU!)
- OBSERVĂM CĂ ÎN MOD NATURAL RAM ARE UN NUMĂR DE LOCAŢII PUTERE A LUI 2.

# Registru serial

REGISTRU: SISTEM DE F-F (RSF-F SAU DFF) CAPABIL SĂ STOCHEZE CUVINTE DE n BIŢI; TOATE F-F AU LINII DE CONTROL (EX. CEAS) COMUNE.

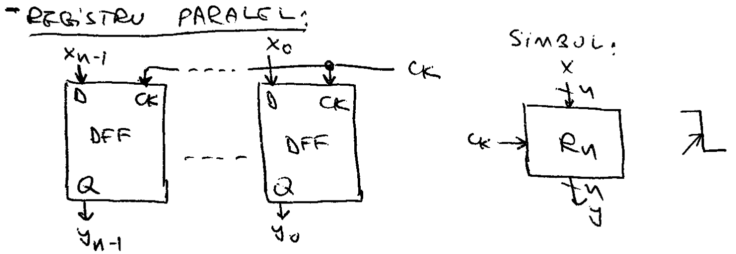
DUPĂ MODUL DE EXTENSIE, REGISTRUL POATE FI:

- REGISTRU SERIAL:




Obs: • REGISTRUL INTRODUCE O ÎNTÂRZIERE DE [redacted] TACŢI ÎNTRE INTRARE ŞI IEŞIRE
• SE POATE DEFINI ŞI RECURSIV (EXTENSIE SERIALĂ) (EXERCIŢIU!)

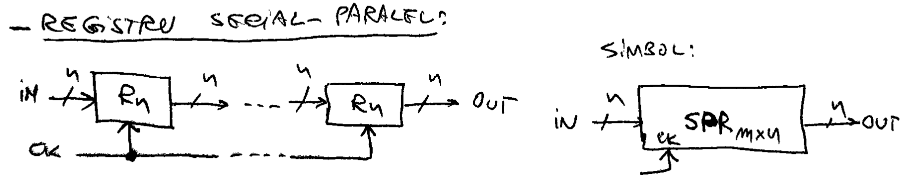
# Registru paralel




Obs: • Principalul avantaj este netransparența, cu excepția unei "transparențe neaccesibile" în primele $t_s + t_H$ momente; deci, poate fi închis cu un nou ciclu și (pentru că este netransparent) se poate încărca cu orice valoare, inclusiv una ce depinde de propriul conținut.

• $R_n$ admite și o definiție recursivă (extensie paralelă). (exercițiu!)

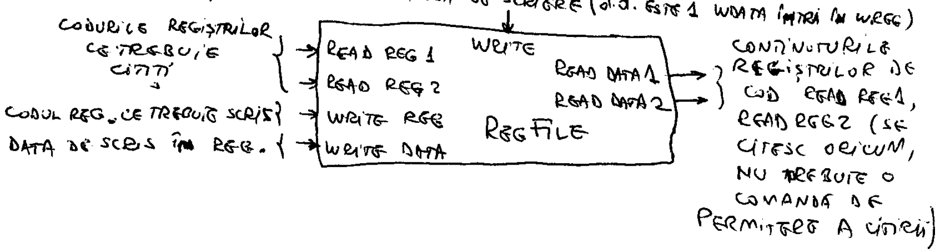
# Registru serial-paralel




# Fișier de regiștri

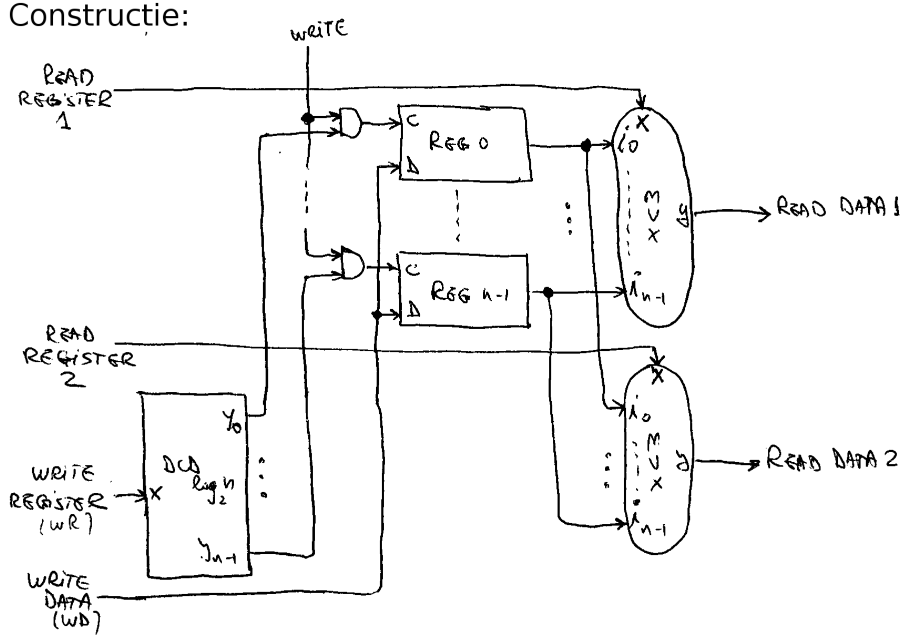
Fișier de regiștri: set de regiștri care pot fi citiți sau scriși prin furnizarea numărului registrului accesat

În diverse variante de procesor MIPS se folosește următorul fișier de regiștri:




# Fișier de regiștri

Construcție:



# Fişier de regiştri

OBS: • WR, WD, W AU CONSTRÂNGERI PRIVIND $t_S, t_H$, PENTRU A ASIGURA CĂ ÎN FIŞIERUL DE REGIŞTRI ESTE CORECT SCRISĂ DATA

• CE SE ÎNTÂMPLĂ DACĂ UN ACELAŞI REGISTRU ESTE CITIT ŞI SCRIS ÎN ACELAŞI CICLU DE CEAS? DEOARECE SCRIEREA SE PRODUCE PE FRONTUL CEASULUI, REGISTRUL ESTE VALID PE PERIOADA DE TIMP CÂND ESTE CITIT; VALOAREA ÎNTOARSĂ LA CITIRE ESTE VALOAREA SCRISĂ ACOLO ÎNTR-UN CICLU PRECEDENT; DACĂ DORIM CA CITIREA SĂ ÎNTOARCĂ VALOAREA SCRISĂ ÎN ACELAŞI CICLU, ESTE NEVOIE DE LOGICĂ SUPLIMENTARĂ ÎN FIŞIERUL DE REGIŞTRI SAU ÎN AFARA LUI.

# Aplicaţie la 1-DS

**Exerciţiu (aplicaţie la 1-DS):**

FIE $a(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{Z}_2[x]$ FIXAT. CONSTRUIŢI UN CIRCUIT CARE PRIMEŞTE SUCCESIV COEFICIENŢII UNUI POLINOM $b(x) \in \mathbb{Z}_2[x]$ ŞI PRODUCE SUCCESIV COEFICIENŢII PRODUSULUI $a(x) \cdot b(x)$.

REZOLVARE:

ÎN $\mathbb{Z}_2$ AVEM $\left\{ \begin{array}{l} \cdot \text{ (ÎNMULŢIREA)} \equiv \wedge \text{ (AND)} \\ + \text{ (ADUNAREA)} \equiv \oplus \text{ (XOR)} \end{array} \right\}$ ŞI SUNT ASOCIATIVE ŞI COMUTATIVE

(SE POATE VERIFICA PE TABELELE OPERAŢIILOR)

DACĂ $b(x) = \sum_{i=0}^{m} b_i x^i$, ATUNCI $a(x) b(x) = \sum_{p=0}^{n+m} c_p x^p$,

ŞI $\forall p \in \overline{0, n+m}$ $c_p = \sum_{\substack{i+j=p \\ i \in \overline{0,n} \\ j \in \overline{0,m}}} a_i b_j = \bigoplus_{\substack{i+j=p \\ i \in \overline{0,n} \\ j \in \overline{0,m}}} a_i \wedge b_j$.

EXPLICITĂM CALCULELE PENTRU CÂTIVA $c_p$:

$c_{n+m} = a_n \wedge b_m$

$c_{n+m-1} = a_n \wedge b_{m-1} \oplus a_{n-1} \wedge b_m$

$c_{n+m-2} = a_n \wedge b_{m-2} \oplus a_{n-1} \wedge b_{m-1} \oplus a_{n-2} \wedge b_m$

- - - -

$c_2 = a_2 \wedge b_0 \oplus a_1 \wedge b_1 \oplus a_0 \wedge b_2$

$c_1 = a_1 \wedge b_0 \oplus a_0 \wedge b_1$

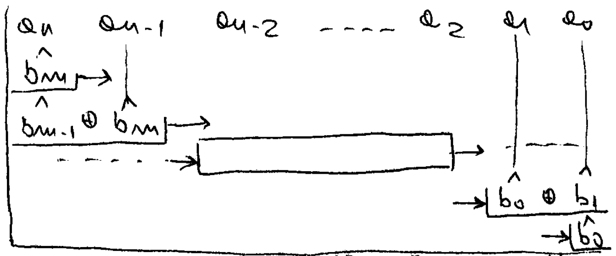
$c_0 = a_0 \wedge b_0$

Obs. CĂ O VREME SUMELE CRESC CA NR. DE TERMENI, APOI (CÂND $p < n, m$) SCAD

# Aplicaţie la 1-DS

PUTEM SĂ PRESUPUNEM CĂ TOATE SUMELE AU TOATE $a_i$-URILE ($n+1$ TERMENI) IAR ACOLO UNDE TERMENUL LIPSEŞTE AVEM $b_j = 0$. ILUSTRĂM MAI JOS MODUL DE CALCUL AL CELOR $n+m+1$ SUME DE CÂTE $n+1$ TERMENI:

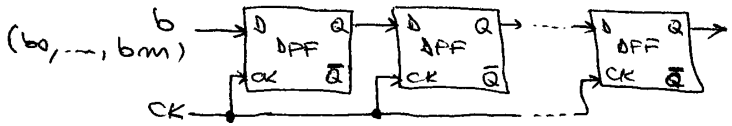
Obs. CĂ "TRENUL" $b_0, \ldots, b_m$ AVANSEAZĂ PAS CU PAS SPRE DREAPTA PE SUB $a_n, \ldots, a_0$ ŞI LA FIECARE PAS SE




ÎNMULŢESC ($\wedge$) $b_j$-URILE CU $a_i$-URILE SUB CARE SE AFLĂ ŞI SE EMITE SUMA ($\oplus$) PRODUSELOR CA $c_p$.

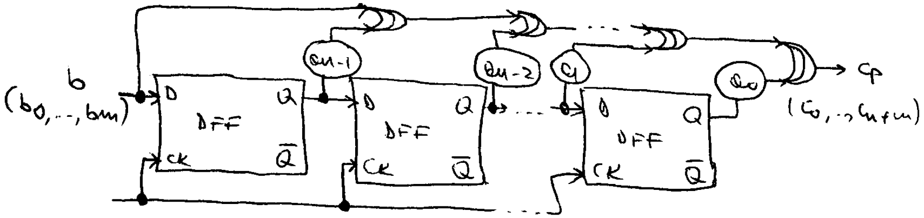
# Aplicaţie la 1-DS

PENTRU IMPLEMENTARE, AVEM NEVOIE ÎN PRIMUL RÂND DE UN CIRCUIT CARE SĂ STOCHEZE SEGMENTUL CURENT DE $n+1$ $b_j$-URI CARE TREBUIE ÎNMULŢITE ($\wedge$) CU $a_i$-URI ŞI SĂ PERMITĂ AVANSUL PAS CU PAS SPRE DREAPTA. UN ASEMENEA CIRCUIT POATE FI:

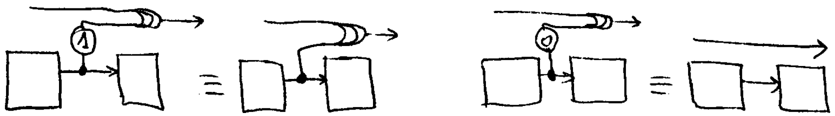


# Aplicație la 1-DS

ACEST CIRCUIT TREBUIE ÎMBOGĂȚIT CU UN CLC CARE LA FIECARE PAS SĂ CALCULEZE DIN $b_j$-URILE CURENT REȚINUTE (PRELUATE DIN D/Q DFF-URILOR) ȘI $a_i$-URILE DATE (FIXATE PRIN CIRCUIT) $c_p$-UL CURENT. AVÂND ÎN VEDERE CĂ $a_i \in \{0,1\}$ ȘI $a_n = 1$ (DĂ GRADUL LUI $a(x)$), CIRCUITUL COMPLET POATE FI:



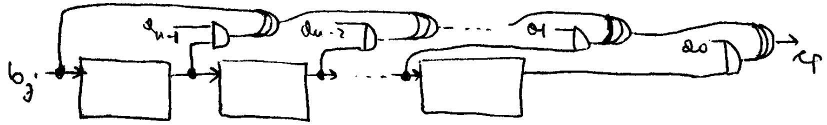

UNDE:






# Aplicație la 1-DS

DACĂ DOREAM CA ȘI $a_i$ SĂ FIE DATE DE INTRARE (DOAR NR. LOR VA FI FIXAT PRIN CONSTRUCȚIE), PUTEM AVEA URMĂTORUL CLC:




Coeficienții $a_i$ trebuie însă introduși toți o dată și menținuți la intrare la fiecare tact, în timp ce coeficienții $b_j$ se introduc pe rând, la tacți succesivi.



# Aplicație la 1-DS

Obs: CONSTATĂM CĂ EXTENSIILE SERIALE DE BISTABILI (EX. DFF) SE PRETEAZĂ LA REZOLVAREA UNOR PROBLEME UNDE SE CERE PRELUCRAREA UNOR ȘIRURI ALE CĂROR ELEMENTE SUNT CITITE PE RÂND, LA TACȚI SUCCESIVI; EX: RECUNOAȘTEREA UNOR PATTERN ÎN ȘIRUL CITIT.

STRUCTURA GENERALĂ A CIRCUITULUI ESTE O EXTENSIE SERIALĂ DE DFF + UN CLC CE EFECTUEAZĂ CALCULUL DE LA FIECARE PAS.



# Automate finite

AUTOMATELE SUNT MAȘINI ABSTRACTE (OBIECTE MATEMATICE), NU TEHNICE. EXISTĂ MAI MULTE FELURI DE ASEMENEA MAȘINI: AUTOMATE FINITE (DETERMINISTE, NEDETERMINISTE, NEDETERMINISTE CU $\lambda$-TRANZIȚII), AUTOMATE STIVĂ, TRANSLATORI (DE DIVERSE FELURI), MAȘINI TURING, ETC.

ELE SUNT FOLOSITE ÎN DIVERSE DOMENII ALE MATEMATICII ȘI INFORMATICII TEORETICE PENTRU A DESCRIE O TRANSFORMARE PRIN PRELUCRAREA EFECTUATĂ PE O ASEMENEA MAȘINĂ: LIMBAJE FORMALE, TEORIA COMPILĂRII, CALCULABILITATE, ETC.

AICI DORIM SĂ IMPLEMENTĂM FUNCȚIONALITATEA UNUI AUTOMAT PRINTR-UN CIRCUIT. VOM VEDEA CĂ ORICE AUTOMAT FINIT SE POATE IMPLEMENTA ÎNTR-UN MOD STANDARD CA 2-DS. DE LA CAZ LA CAZ, ÎN FUNCȚIE DE UTILITATEA DE MOMENT, SE POATE CONSTRUI ȘI UN 1-DS, 3-DS, ETC, DAR ÎN TOATE CAZURILE VA EXISTA ȘI UN 2-DS ECHIVALENT.

ÎN ORICE CAZ, NU TREBUIE CONFUNDAT AUTOMATUL (OBIECT MATEMATIC) CU 2-DS-UL CARE ÎL IMPLEMENTEAZĂ (AȘA CUM NU TREBUIE CONFUNDATĂ O FUNCȚIE BOOLEANĂ CU PLA-UL/PROM-UL CARE O IMPLEMENTEAZĂ).

## Automate finite

DEF: AUTOMAT: SISTEM $A=(Q, X, Y, \delta, \lambda)$, UNDE:

- $Q$ MULȚIME NEVIDĂ (STĂRI)
- $X$ MULȚIME FINITĂ NEVIDĂ (VALORI DE INTRARE)
- $Y$ MULȚIME FINITĂ NEVIDĂ (VALORI DE IEȘIRE)
- $\delta : Q \times X \to Q$ (FUNCȚIA DE TRANZIȚIE A STĂRILOR)
- $\lambda$ ~~ESTE~~ FUNCȚIA DE IEȘIRE, DEFINITĂ $\begin{cases} \lambda : Q \times X \to Y & \text{(PT. AUT. MEALY)} \\ \lambda : Q \to Y & \text{(PT. AUT. MOORE)} \end{cases}$

OBS: DE FAPT AM DEFINIT NOȚIUNEA DE TRANSLATOR, DIN TEORIA LIMBAJELOR FORMALE.

DEF. AUTOMATUL ESTE FINIT, DACĂ $\begin{cases} Q \text{ FINITĂ} \\ \delta \text{ ACCEPTĂ O DEFINIȚIE NERECURSIVĂ} \end{cases}$

OBS: SE POATE DEMONSTRA CĂ AUTOMATELE MEALY/MOORE SUNT ECHIVALENTE (I.E. SE POT DESCRIE UNUL PRIN CELĂLALT) FĂCÂND EVENTUAL ABSTRACȚIE DE PRIMA VALOARE DE IEȘIRE.

ÎN CELE CE URMEAZĂ, DACĂ NU SPECIFICĂM ALTCEVA, VOM CONSIDERA AUTOMATE FINITE MEALY (ÎN CARTEA P&H SUNT FOLOSITE MULT AUTOMATE MOORE).

## Automate finite

UN AUTOMAT POATE FI DESCRIS:

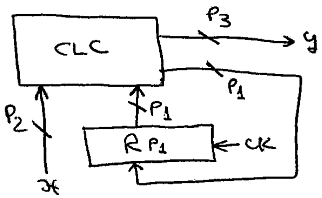
- SPECIFICÂND $\begin{cases} Q, X, Y & \text{PRIN ENUMERARE} \\ \delta, \lambda & \text{PRIN TABEL} \end{cases}$
- PRINTR-UN GRAF DE TRANZIȚIE, AVÂND:
  - NODURI = STĂRI: $(q)$
  - ARCE:
    - LA AUT. MEALY: $(q) \xrightarrow{x/y} (t)$, ÎNSEMNÂND $\begin{cases} \delta(q,x)=t \\ \lambda(q,x)=y \end{cases}$
    - LA AUT. MOORE: $(q)^{y} \xrightarrow[x]{} (t)$, ÎNSEMNÂND $\begin{cases} \delta(q,x)=t \\ \lambda(q)=y \end{cases}$

## Automate finite

IMPLEMENTARE:

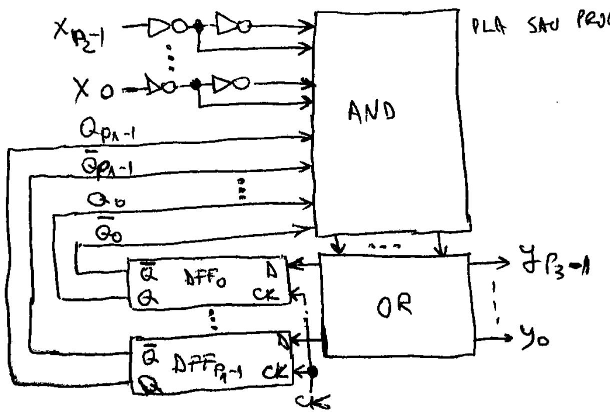
- NUMEROTĂM ELEMENTELE LUI $Q, X, Y$ (EX: $q_0, \ldots, q_{n-1}$) ȘI ÎNLOCUIM FIECARE ELEMENT CU CODUL SĂU, REPREZENTAT PE UN NUMĂR DE BIȚI (EX: $q_6 \to 6 \to 110$). DECI $Q=\{0,1\}^{p_1}$, $X=\{0,1\}^{p_2}$, $Y=\{0,1\}^{p_3}$.
- FOLOSIM UN REGISTRU PENTRU A REȚINE STAREA CURENTĂ ȘI ÎL ÎNCHIDEM PRINTR-UN CICLU CE CONȚINE UN CLC CE IMPLEMENTEAZĂ $\delta$ ȘI $\lambda$.

VARIANTĂ: ÎN LOC DE UN REGISTRU FOLOSIM UN SISTEM DE DFF



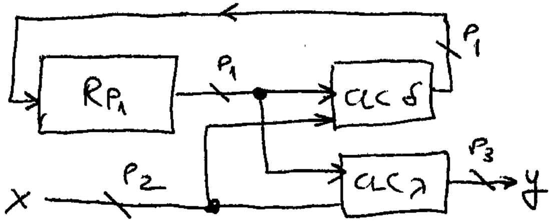

## Automate finite

VARIANTA ÎN CARE FOLOSIM DFF + PLA/PROM:

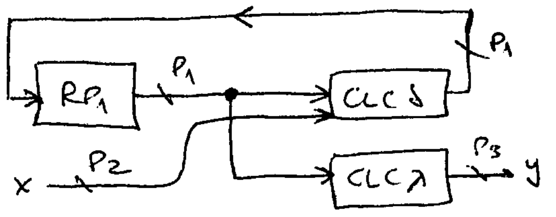


## Automate finite

SE POATE OPTA PENTRU CLC SEPARATE PENTRU $\delta$, $\lambda$;
AVANTAJ: CLC MAI MICI




AUTOMAT MEALY




AUTOMAT MOORE

## Automate finite

ÎN CARTEA P&H SE ZICE: AUT. MOORE AU AVANTAJUL CĂ POT FI MAI RAPIDE, AUT. MEALY AU AVANTAJUL CĂ POT FI MAI MICI (I.E. CU MAI PUȚINE STĂRI).

CUM SE REZOLVĂ O PROBLEMĂ DE IMPLEMENTARE:

- DIN ENUNȚ EXTRAGEM O DESCRIERE (SPECIFICARE SAU GRAF), OBȚINÂND TABELE PENTRU $\delta$, $\lambda$
- DIN TABELE CONSTRUIM CLC CA PLA/PROM ÎN MANIERA OBIȘNUITĂ; RESTUL CIRCUITULUI E STANDARD

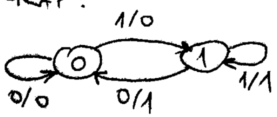
OBS! UNEORI AUTOMATUL POATE FI SIMPLIFICAT ÎNAINTE DE IMPLEMENTARE, DE EXEMPLU PUTEM ELIMINA ANUMITE STĂRI CARE NU SUNT UTILIZATE (DE EX. SUNT INACCESIBILE); EXISTĂ ALGORITMI ÎN ACEST SENS (VEZI LIMBAJE FORMALE).

## Automatul DFF

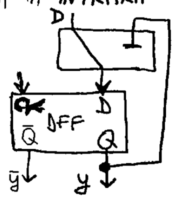
Prezentăm câteva automate importante, ce pot fi folosite ca bistabili (flip-flop) de eficiență sporită:

ORICE DFF SE POATE EXPRIMA CA UN AUT. CU 2 STĂRI, UNDE CLC SE REDUCE LA FUNCȚIA IDENTICĂ A INTRĂRII

GRAF:




IMPLEMENTARE:

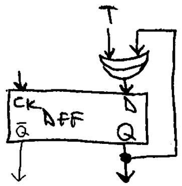



În acest caz, $Q^+ = D$, iar $y = Q$ (sau $y = D$ cu întârziere de un tact).

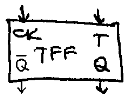
Unui automat DFF trebuie să-i dăm comanda $D = r$, pentru a-l determina să treacă din starea curentă $Q = s$ în starea nouă $Q^+ = r$ (pe scurt: $D = r$, pentru $s \to r$); la ieșire va furniza starea curentă $s$.

## Automatul TFF

AUTOMATUL T FLIP-FLOP (TFF):




SIMBOL:

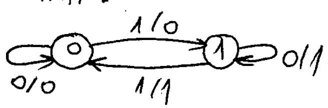



FUNCȚIONARE:

| T | $Q^+$ |
|---|---|
| 0 | $Q$ |
| 1 | $\bar{Q}$ |

DECI $T = \begin{cases} r, & \text{PENTRU } 0 \to r \\ \bar{r}, & \text{PENTRU } 1 \to r \end{cases}$

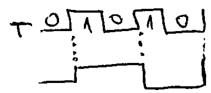
GRAF:




$Q = X = Y = \{0, 1\}$

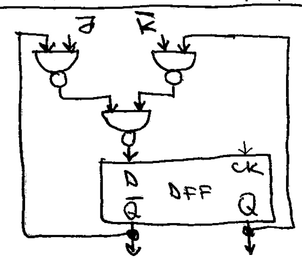
UTILIZĂRI:

- COUNTER MODULO 2: DACĂ MENȚIN T=1 TIMP DE MAI MULȚI TACȚI, IEȘIREA VA FI: 0, 1, 0, 1, ...
- DIVIZOR DE FRECVENȚĂ:

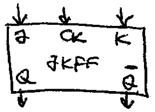


# Automatul JKFF

AUTOMATUL JK FLIP-FLOP (JKFF):




SIMBOL:




FUNCȚIONARE:

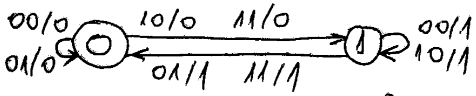
| J | K | $Q^+$ | |
|---|---|---|---|
| 0 | 0 | $Q$ | (no op) |
| 0 | 1 | 0 | (reset) |
| 1 | 0 | 1 | (set) |
| 1 | 1 | $\overline{Q}$ | (switch) |

DECI

$$JK = \begin{cases} r\,-, & \text{PENTRU } 0 \to r \\ -\,\overline{r}, & \text{PENTRU } 1 \to r \end{cases}$$

(ASTFEL, POT ALEGE ÎN MAI MULTE FELURI COMANDA PT. ACELAȘI EFECT, ȘI VOI ALEGE A.Î. CLC SĂ IASĂ CÂT MAI SIMPLU)

GRAF:




$Q = Y = \{0,1\}$, $X = \{0,1\}^2$

COMENTARII:

- AUTOMATELE TFF, JKFF, SPRE DEOSEBIRE DE BISTABILII RSFF, DFF ȘTIU SĂ COMUTE ÎN CONTRARA STĂRII - LA ASTA ESTE NECESAR CICLUL SUPLIMENTAR; ÎN PLUS, LA JKFF CAPĂTĂ SENS INTRAREA 1-1.
- JKFF ESTE CEL MAI BUN FLIP-FLOP DE PÂNĂ ACUM; EL LE GENERALIZEAZĂ PE CELELALTE:
  PT. $J = \overline{K} = D \Longrightarrow$ DFF
  PT. $J = K = T \Longrightarrow$ TFF
- CU JKFF SE POT CONSTRUI DIVIZORI IMPARI DE FRECVENȚĂ
- IMPLEMENTAREA GENERALĂ A UNUI AUTOMAT SE POATE FACE ȘI CU SISTEME DE TFF SAU JKFF + CLC (ÎN LOC DE DFF + CLC).

Observație: Dacă la implementarea generală a unui automat, în locul bistabililor DFF (care sunt 1-DS) folosim automate DFF, TFF sau JKFF (care sunt 2-DS), circuitul rezultat nu va mai fi un 2-DS, ci un 3-DS.

# Aplicație la 2-DS

**Exercițiu rezolvat (sinteza unui automat finit):**

Construiți un automat finit care recunoaște apariția secvenței 1011 în cadrul unei secvențe binare citite succesiv (i.e. furnizează la ieșire 1 d.d. ultimii 4 biți citiți formează secvența 1011).

Exemplu: IN: 0 1 0 1 0 1 1 0 1 1 1 0 0 1 1
OUT: 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0

**Rezolvare:**

Problema este una de limbaje formale - acolo se dau metode cu care putem construi automatul din enunț în mod algoritmic; aici îl construim intuitiv, plecând de la următoarele observații:

- Automatul trebuie să aibă 4 stări, corespunzătoare etapelor în care este recunoscut 1011: 1, 10, 101, 1011; aceste stări se pot numerota $q_0$, $q_1$, $q_2$, $q_3$ și se pot asimila cu numerele 0, 1, 2, 3, pe care le putem scrie pe doi biți: 00, 01, 10, 11:


În fiecare stare se poate primi la intrare 1 sau 0, deci avem câte două arce; în fiecare caz, se poate emite la ieșire 1 = recunoscut, 0 = nerecunoscut (încă). Astfel, $Q = \{0,1\}^2$, $X = Y = \{0,1\}$.

# Aplicație la 2-DS

- Drumul pe care este recunoscută o apariție a lui 1011 este:

$$q_0 \xrightarrow{1/0} q_1 \xrightarrow{0/0} q_2 \xrightarrow{1/0} q_3 \xrightarrow{1/1}$$

Restul arcelor care vor fi adăugate corespund celorlalte cazuri.

Notăm că tranziția din starea $q_3$ pentru intrarea $x = 1$ este singura tranziție a automatului în care se emite la ieșire $y = 1$ (deoarece doar acum se recunoaște o apariție a lui 1011); deci, arcul corespunzător este singurul din desen care va avea notat "/1" (restul vor avea notat "/0").

## Aplicație la 2-DS

- Dorim ca automatul să recunoască aparițiile lui 1011 chiar dacă se suprapun parțial, de exemplu:

$$1\,0\,1\,1\,0\,1\,1$$

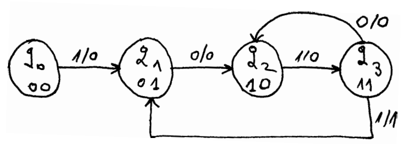
De asemenea, chiar dacă ultimii 4 biți introduși nu formează 1011, este posibil ca o parte din ei, aflați la sfârșit, să formeze 1011 împreună cu biții care vor fi citiți în continuare.

De aceea, la procesarea unei intrări $x = 1$ sau 0, indiferent dacă a recunoscut sau nu o apariție a lui 1011, automatul trebuie să treacă într-o stare care să permită păstrarea celui mai lung sufix al secvenței introduse care ar putea fi prefix al unei (noi) apariții a lui 1011.

## Aplicație la 2-DS

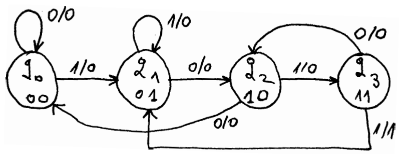
Astfel, tranziția din starea $q_3$ $(= 11)$ pentru intrarea $x = 1$ trebuie să ducă în starea $q_1$ $(= 01)$, deoarece în acest moment ultimii 4 biți introduși sunt 1011 și dintre ei putem păstra doar ultimul 1 (el ar putea fi primul 1 într-o nouă apariție a lui 1011 și ar corespunde drumului $q_0 \xrightarrow{1/0} q_1$).

De asemenea, tranziția din starea $q_3$ $(= 11)$ pentru intrarea $x = 0$ trebuie să ducă în starea $q_2$ $(= 10)$, deoarece în acest moment ultimii 4 biți introduși sunt 1010 și dintre ei putem păstra doar sufixul 10 (el ar putea prefixul 10 într-o viitoare apariție a lui 1011 și ar corespunde drumului $q_0 \xrightarrow{1/0} q_1 \xrightarrow{0/0} q_2$):




## Aplicație la 2-DS

În final, graful de tranziție al automatului este:




Specificația automatului folosind tabele de valori compacte (a se vedea sfârșitul secțiunii referitoare la algebra booleană $B_2$) este:

$Q = \{0,1\}^2$, $X = Y = \{0,1\}$.

$\delta$:

| $q_1$ | $q_0$ | $x=0$ | $x=1$ | $q_1^+$ $(D_1)$ | $q_0^+$ $(D_0)$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 0 | 0 1 | 0 | $x$ |
| 0 | 1 | 1 0 | 0 1 | $\overline{x}$ | $x$ |
| 1 | 0 | 0 0 | 1 1 | $x$ | $x$ |
| 1 | 1 | 1 0 | 0 1 | $\overline{x}$ | $x$ |

$\lambda$:

| $q_1$ | $q_0$ | $x=0$ | $x=1$ | $y$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | $x$ |

În cazul implementării prin DFF + PLA, biții de stare rezultați $q_1^+$, $q_0^+$, sunt chiar comenzile $D_1$, $D_0$ ce trebuie date DFF-urilor pentru a trece în stările respective (conform regulii: "$D = r$, pentru $s \to r$"); în tabel, am notat acest fapt între paranteze.

## Aplicație la 2-DS

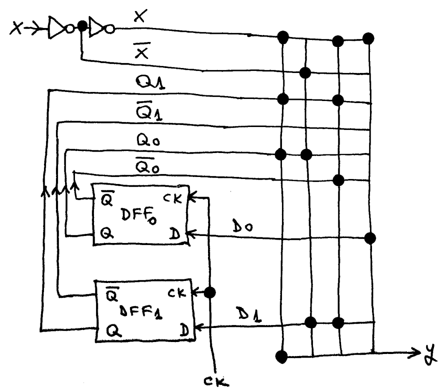
Din tabelele precedente, rezultă următoarele sume de produse minimale:

$D_1 = q_0\overline{x} + q_1\overline{q_0}x$

$D_0 = x$

$y = q_1 q_0 x$

Implementare:




Evaluarea complexității CLC: 13 puncte de contact.

Obs: $\overline{Q_1}$ nu are contacte și putea fi omis din circuit.

## Aplicație la 2-DS

În cazul implementării prin TFF + PLA, trebuie să completăm tabelul lui $\delta$ cu coloane în care să calculăm comenzile $T_1$, $T_0$ ce trebuie date TFF-urilor pentru a trece din stările $q_1$, $q_0$ în respectiv stările $q_1^+$, $q_0^+$.
Pentru fiecare $i = 1, 2$, $T_i$ se calculează din $q_i$ și $q_i^+$, pe baza regulii:

$$T = \begin{cases} r, & \text{pentru } 0 \to r \\ \overline{r}, & \text{pentru } 1 \to r \end{cases}$$

Obținem tabelul:

$\delta$:

| $q_1$ | $q_0$ | $x$: 0 | 1 | $q_1^+$ | $q_0^+$ | $T_1$ | $T_0$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 0 | 0 1 | 0 | $x$ | 0 | $x$ |
| 0 | 1 | 1 0 | 0 1 | $\overline{x}$ | $x$ | $\overline{x}$ | $\overline{x}$ |
| 1 | 0 | 0 0 | 1 1 | $x$ | $x$ | $\overline{x}$ | $x$ |
| 1 | 1 | 1 0 | 0 1 | $\overline{x}$ | $x$ | $x$ | $\overline{x}$ |

Rezultă următoarele sume de produse minimale:

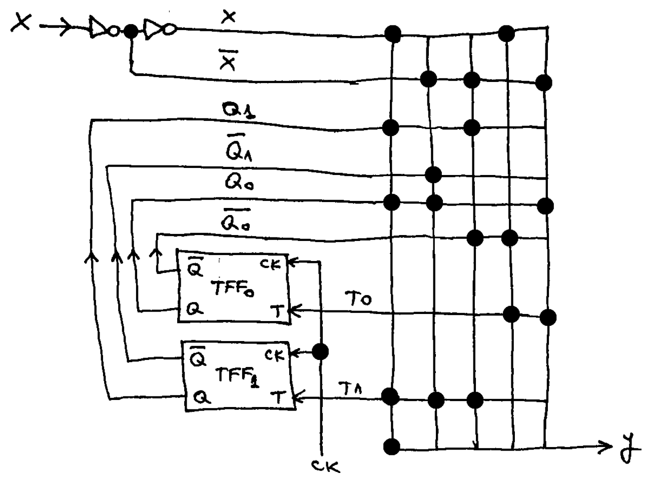
$T_1 = \overline{q_1} q_0 \overline{x} + q_1 \overline{q_0}\, \overline{x} + q_1 q_0 x$

$T_0 = \overline{q_0} x + q_0 \overline{x}$

$y = q_1 q_0 x$ (a rămas la fel ca mai înainte).

## Aplicație la 2-DS

Implementare:




Evaluarea complexității CLC: 19 puncte de contact.

## Aplicație la 2-DS

În cazul implementării prin JKFF + PLA, trebuie să completăm tabelul lui $\delta$ cu coloane în care să calculăm comenzile $J_1$, $K_1$, $J_0$, $K_0$ ce trebuie date JKFF-urilor pentru a trece din stările $q_1$, $q_0$ în respectiv stările $q_1^+$, $q_0^+$.
Pentru fiecare $i = 1, 2$, $J_i$, $K_i$, se calculează din $q_i$ și $q_i^+$, pe baza regulii:

$$JK = \begin{cases} r\,\_, & \text{pentru } 0 \to r \\ \_\,\overline{r}, & \text{pentru } 1 \to r \end{cases}$$

Obținem tabelul (am păstrat doar coloanele relevante):

$\delta$:

| $q_1$ | $q_0$ | $x$: 0 | 1 | $q_1^+$ | $q_0^+$ | $J_1$ | $K_1$ | $J_0$ | $K_0$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 0 | 0 1 | 0 | $x$ | 0 | _ | $x$ | _ |
| 0 | 1 | 1 0 | 0 1 | $\overline{x}$ | $x$ | $\overline{x}$ | _ | _ | $\overline{x}$ |
| 1 | 0 | 0 0 | 1 1 | $x$ | $x$ | _ | $\overline{x}$ | $x$ | _ |
| 1 | 1 | 1 0 | 0 1 | $\overline{x}$ | $x$ | _ | $x$ | _ | $\overline{x}$ |

Au rezultat niște coloane ce conțin valori indiferente "_", pe care trebuie să le înlocuim cu ceva concret pentru a putea obține din aceste coloane o scriere a lui $J_i$, $K_i$, $i = 1, 2$, ca sumă de produse.

## Aplicație la 2-DS

Înlocuirea se va face a.î. sumele de produse rezultate să fie cât mai simple (cât mai puțini termeni, cât mai puține variabile), conform criteriilor prezentate la sfârșitul secțiunii referitoare la algebra booleană $B_2$.
Am adăugat coloanele obținute la sfârșitul tabelului:

$\delta$:

| $q_1$ | $q_0$ | $x$: 0 | 1 | $q_1^+$ | $q_0^+$ | $J_1$ | $K_1$ | $J_0$ | $K_0$ | $J_1$ | $K_1$ | $J_0$ | $K_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 0 | 0 1 | 0 | $x$ | 0 | _ | $x$ | _ | 0 | $\overline{x}$ | $x$ | $\overline{x}$ |
| 0 | 1 | 1 0 | 0 1 | $\overline{x}$ | $x$ | $\overline{x}$ | _ | _ | $\overline{x}$ | $\overline{x}$ | $x$ | $x$ | $\overline{x}$ |
| 1 | 0 | 0 0 | 1 1 | $x$ | $x$ | _ | $\overline{x}$ | $x$ | _ | 0 | $\overline{x}$ | $x$ | $\overline{x}$ |
| 1 | 1 | 1 0 | 0 1 | $\overline{x}$ | $x$ | _ | $x$ | _ | $\overline{x}$ | $\overline{x}$ | $x$ | $x$ | $\overline{x}$ |

Rezultă următoarele sume de produse minimale:

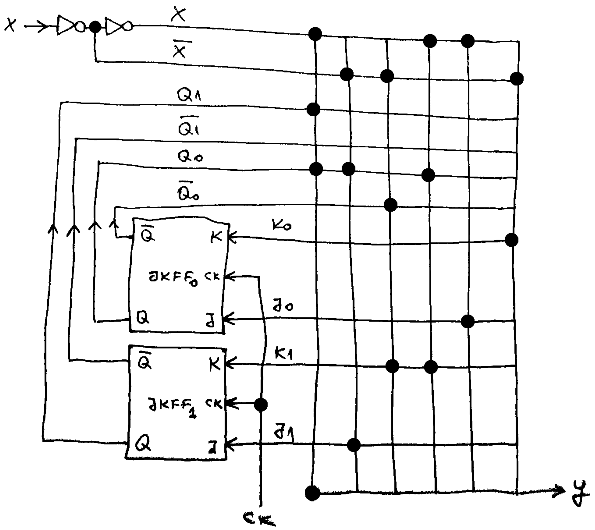
$J_1 = q_0 \overline{x}$, $J_0 = x$, $y = q_1 q_0 x$ (a rămas la fel ca mai înainte).

$K_1 = \overline{q_0}\, \overline{x} + q_0 x$, $K_0 = \overline{x}$,

Obs: Dacă în coloana $J_1$ am fi înlocuit cele două "_" cu 0, ar fi rezultat $J_1 = \overline{q_1} q_0 \overline{x}$ (un termen cu trei variabile, în loc de două); de asemenea, dacă în coloana $J_0$ am fi înlocuit cele două "_" cu 0, ar fi rezultat $J_0 = \overline{q_0} x$ (un termen cu două variabile, în loc de una).

## Aplicație la 2-DS

Implementare:




Evaluarea complexității CLC: 17 puncte de contact.

## Aplicație la 2-DS

Observații:

- Mai sus, am încercat să minimizăm CLC minimizând fiecare sumă de produse în parte.

Un alt criteriu de minimizare ar fi să înlocuim valorile indiferente "_" a.î. sumele de produse rezultate să aibă cât mai mulți termeni comuni.

- În mod normal, implementarea cu JKFF ar fi trebuit să conducă la circuitul cel mai simplu (cel mai mic număr de puncte de contact), deoarece avem mai multe variante de valori J, K, dintre care putem alege una cât mai bună.

O explicație de ce nu s-a întâmplat așa poate fi complexitatea mare a instrumentului folosit - JKFF are două comenzi, spre deosebire de DFF și TFF, care au doar una. Aceasta adaugă în mod artificial complexitate circuitului.

În general, instrumentele eficiente dar complexe își evidențiază avantajele în cazul sarcinilor complexe (circuite complexe, cu multe puncte de contact). În cazul sarcinilor simple (cum este circuitul cerut în această problemă) sunt mai bune instrumentele simple (în cazul de față DFF - pentru el am obținut numărul minim de puncte de contact).

## Aplicație la 2-DS

Și în dezvoltarea de software, dacă avem de scris un program complex, cu mii de linii de cod, o ierarhie complexă de clase, multe fișiere sursă, este de preferat să folosim un IDE complex, care să ne ofere multiple instrumente de a gestiona proiecte de mare anvergură.

Dacă însă avem de scris un program de 10 - 20 linii, un IDE complex se poate dovedi incomod - trebie să creăm un proiect, să facem mai multe setări, pot apărea probleme de incompatibilitate dacă dorim să recompilăm proiectul cu o altă versiune a IDE-ului, etc.

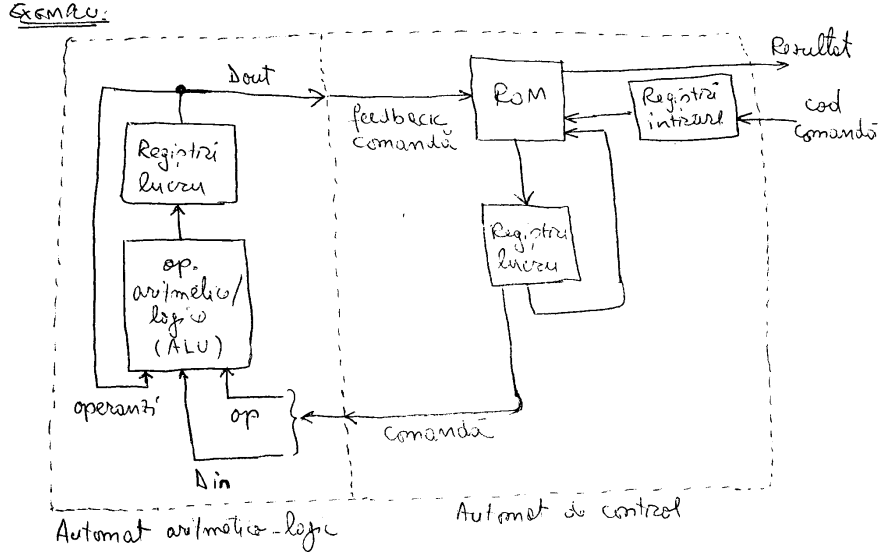
În acest caz este mai eficient să folosim instrumente simple: un editor de fișiere text (ex. "gedit") pentru codul sursă și un compilator în mod linie de comandă (ex. "gcc").

## 3-DS (Procesoare)

Procesor:

UN PROCESOR ESTE UN CIRCUIT CE LEAGĂ PRINTR-UN CICLU UN AUTOMAT ARITMETICO-LOGIC CU UN AUTOMAT DE CONTROL

EXEMPLU:

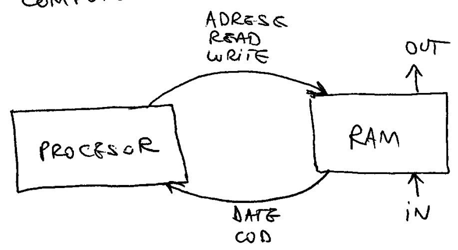


OBS: DETALIILE DE ORGANIZARE A PROCESORULUI POT DIFERI MULT DE LA O ARHITECTURĂ LA ALTA; ÎN CONTINUARE VOM STUDIA ASEMENEA DETALII ÎN CAZUL ARHITECTURII MIPS.



Calculator:

EXEMPLU: COMPUTER-UL:




RAM-UL CONȚINE DATE ȘI PROGRAME

OBS: PROCESORUL SE POATE ÎNCHIDE ȘI PRIN:

- CLC (0-DS)
- STIVĂ (2-DS)
- CO-PROCESOR (3-DS)

OBS: ȘI ÎN CAZUL CALCULATOARELOR DETALIILE DE ORGANIZARE POT DIFERI MULT DE LA O ARHITECTURĂ LA ALTA.