# II.5. MINIMIZAREA FUNCȚIILOR BOOLEENE PRIN METODA DIAGRAMELOR KARNAUGH

# Structura unei diagrame Karnaugh

## pentru **n** variabile

- Pe structura unui **tabel bidimensional**
- Zona variabilelor
  - 2 clase : etichete de linii / coloane (**n** par → clase egale)
  - se scriu **numele** variabilelor
- Zona etichetelor
  - o etichetă este un **şir de n biţi**, dacă funcţia are **n** variabile
  - Pentru **n** par, **n**/2 biţi într-o etichetă de linii, **n**/2 pentru o etichetă de coloană
  - fiecare bit dintr-o etichetă corespunde unei variabile
- Zona celor $2^n$ locaţii din diagramă
  - în care se vor trece doar valorile de 1
  - unei locaţii îi corespunde o unică etichetă

# Metoda Karnaugh

Ordinea codului Grey

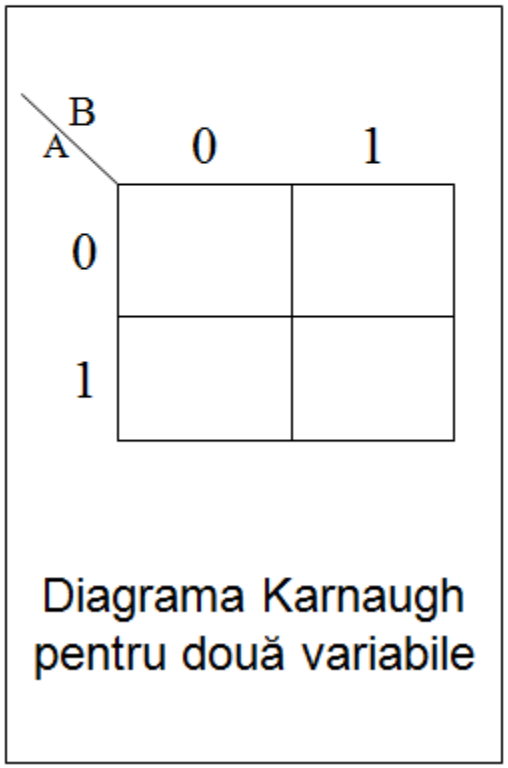



Diagrama Karnaugh pentru două variabile

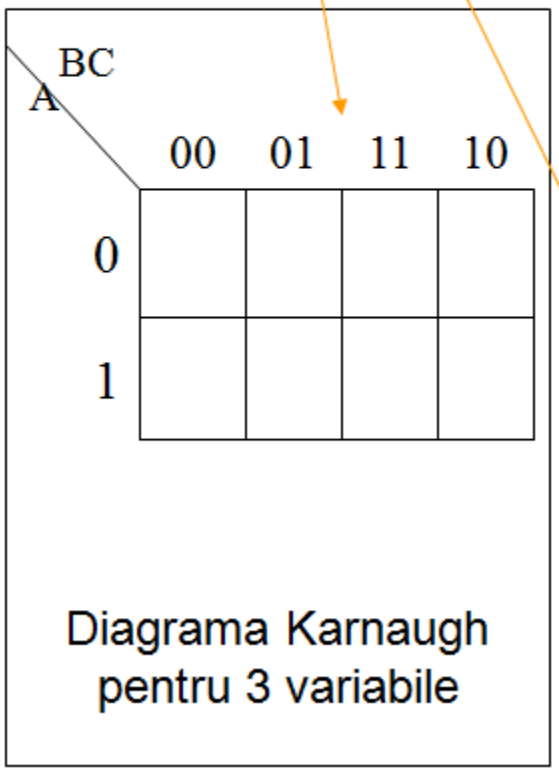



Diagrama Karnaugh pentru 3 variabile

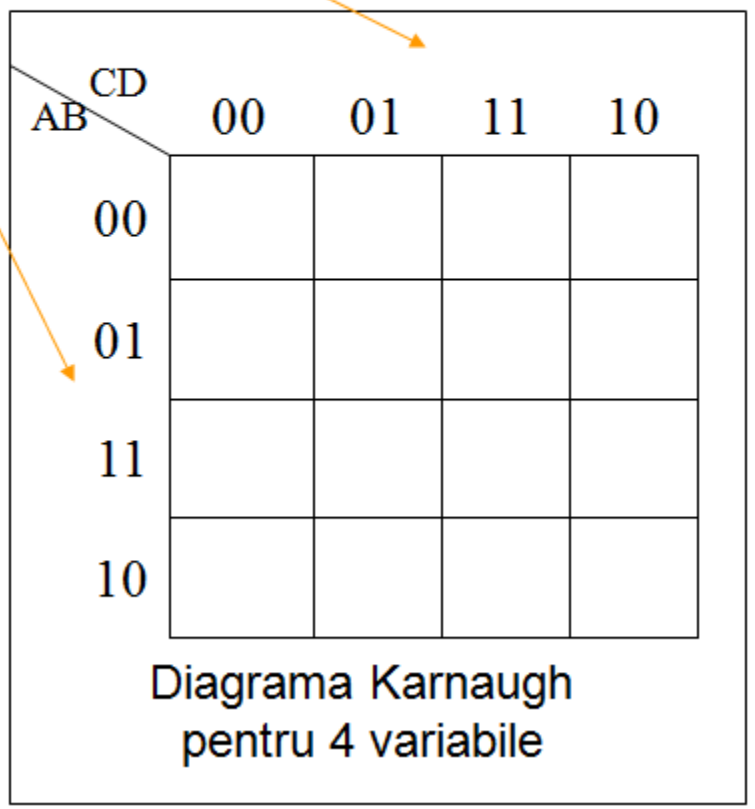



Diagrama Karnaugh pentru 4 variabile

# Etichete: codul Grey

•etichetele nu se scriu în ordinea naturală, ci în *ordinea Grey*

•pe 2 poziţii binare: 00, 01, 11, 10

•pe 3 poziţii binare:

**0**00, **0**01, **0**11, **0**10, **1**10, **1**11, **1**01, **1**00

•pe 4 poziţii binare:

**0**000, **0**001, **0**011, **0**010, **0**110, **0**111, **0**101,**0**100,
**1**100, **1**101, **1**111, **1**110, **1**010, **1**011, **1**001,**1**000

•oricare două etichete consecutive – inclusiv prima şi ultima! - diferă printr-un singur bit

# Adiacenţe în diagrame Karnaugh

- Două poziţii sunt adiacente dacă etichetele corespunzătoare diferă pe un singur bit
- Generalizează "vecinătatea" intuitivă
- 4 variabile: cele patru colţuri sunt adiacente!
- Pentru o funcţie de **n** variabile, o locaţie are **n** locaţii adiacente
  - < 5 variabile – vizual
  - 5 sau mai multe: şi alte adiacenţe decât cele vizibile

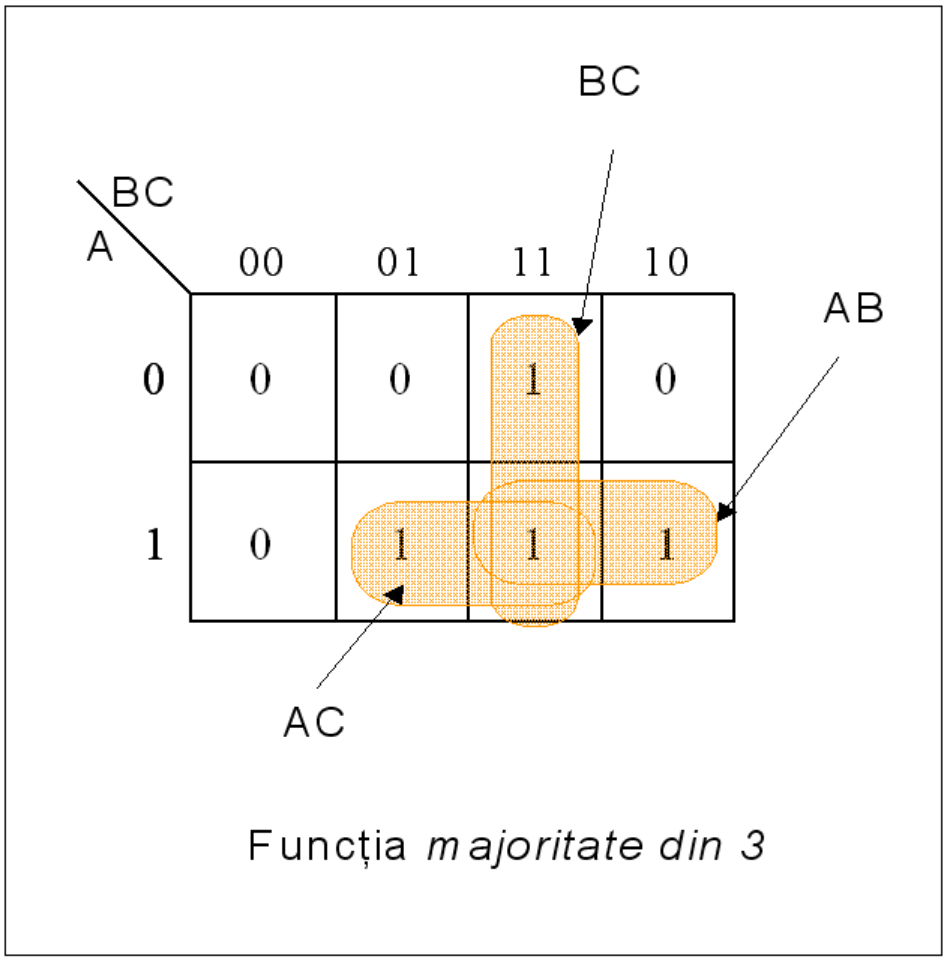
# Paşii minimizării Karnaugh

1. Se trec în locaţiile corespunzătoare (conform etichetelor) valorile de 1 ale funcţiei
2. Se caută blocuri conţinând numai valori 1, astfel încât:
   - fiecare valoare 1 să fie inclusă în cel puţin un bloc
   - blocurile să fie cât mai mari şi mai puţine
   - un bloc să conţină un număr de locaţii egal cu o putere a lui 2
     - eventual puterea 0
   - dacă blocul conţine $2^{\mathbf{k}}$ locaţii, atunci pentru fiecare locaţie blocul să conţină exact **k** locaţii adiacente cu ea

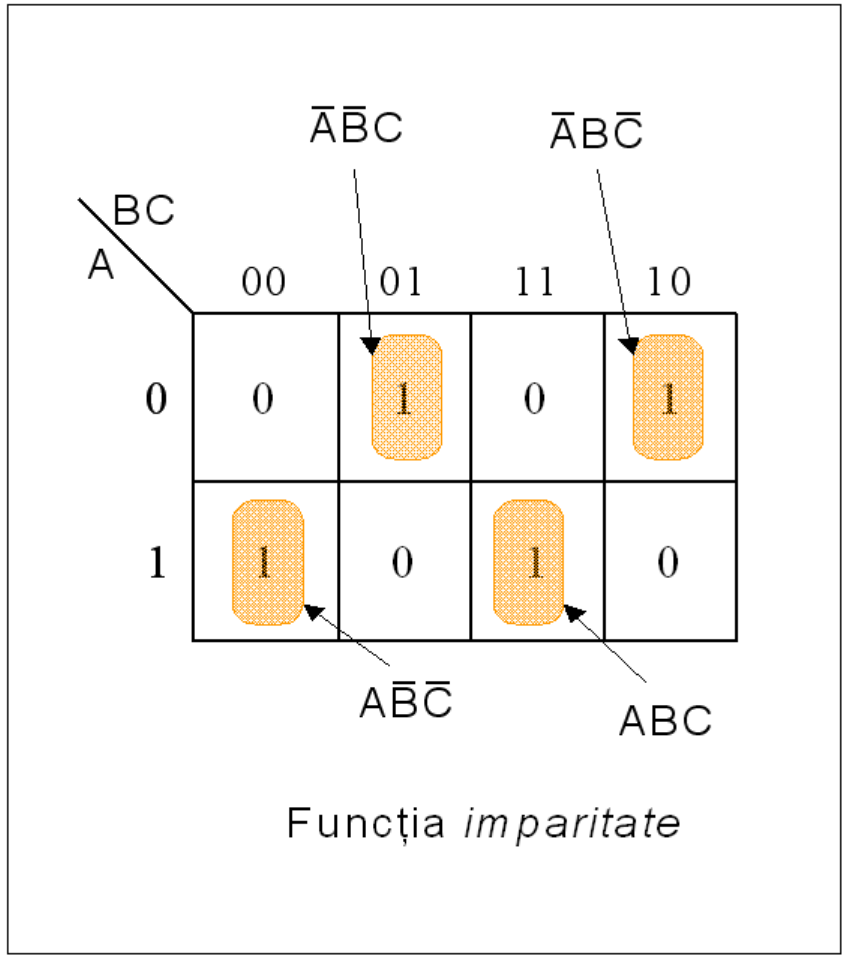
# Paşii minimizării Karnaugh

3. Se scrie expresia minimizată a funcţiei astfel:
   - fiecărui bloc cu $2^k$ locaţii 1 îi corespunde un termen conţinând **n-k** variabile legate prin conjuncţie
   - în termen apar acele variabile ale căror etichete sunt constante pentru toate locaţiile din bloc
   - o variabilă apare negată dacă eticheta sa constantă este 0 şi nenegată altfel
   - termenii astfel obţinuţi (după considerarea tuturor blocurilor) sunt legaţi prin disjuncţie

# Exemple: "majoritatea din 3"; imparitate

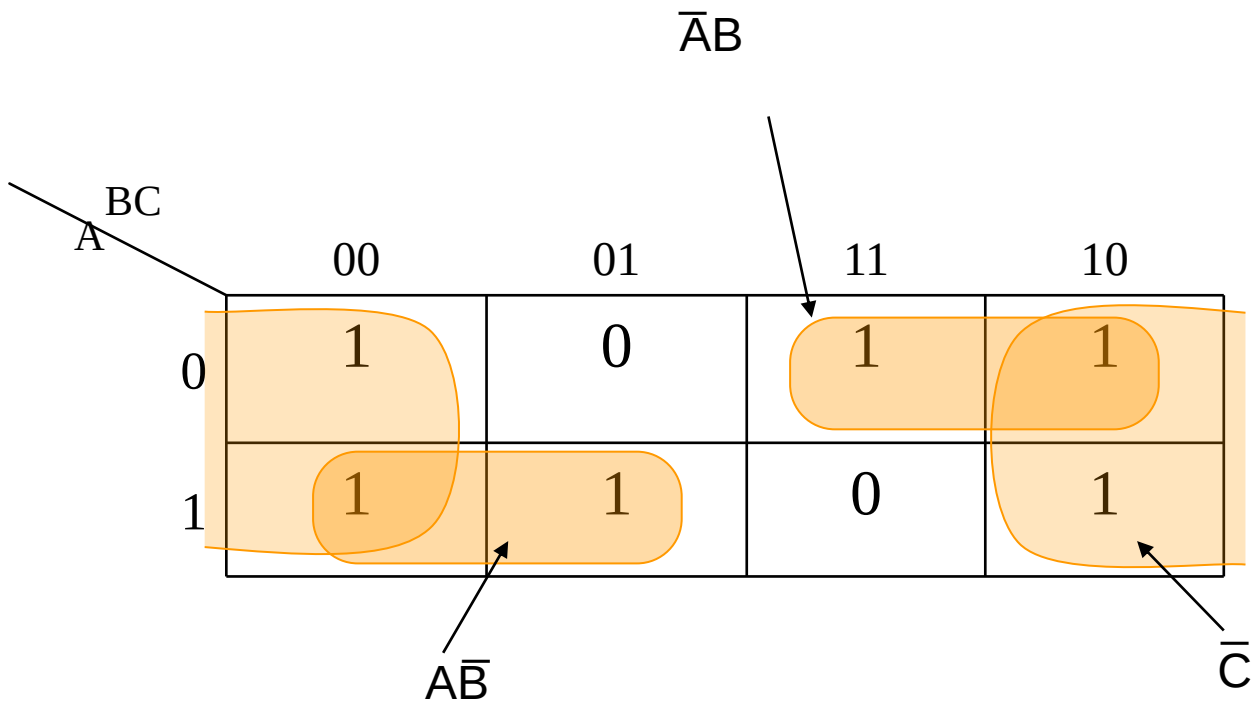



Funcția *majoritate din 3*

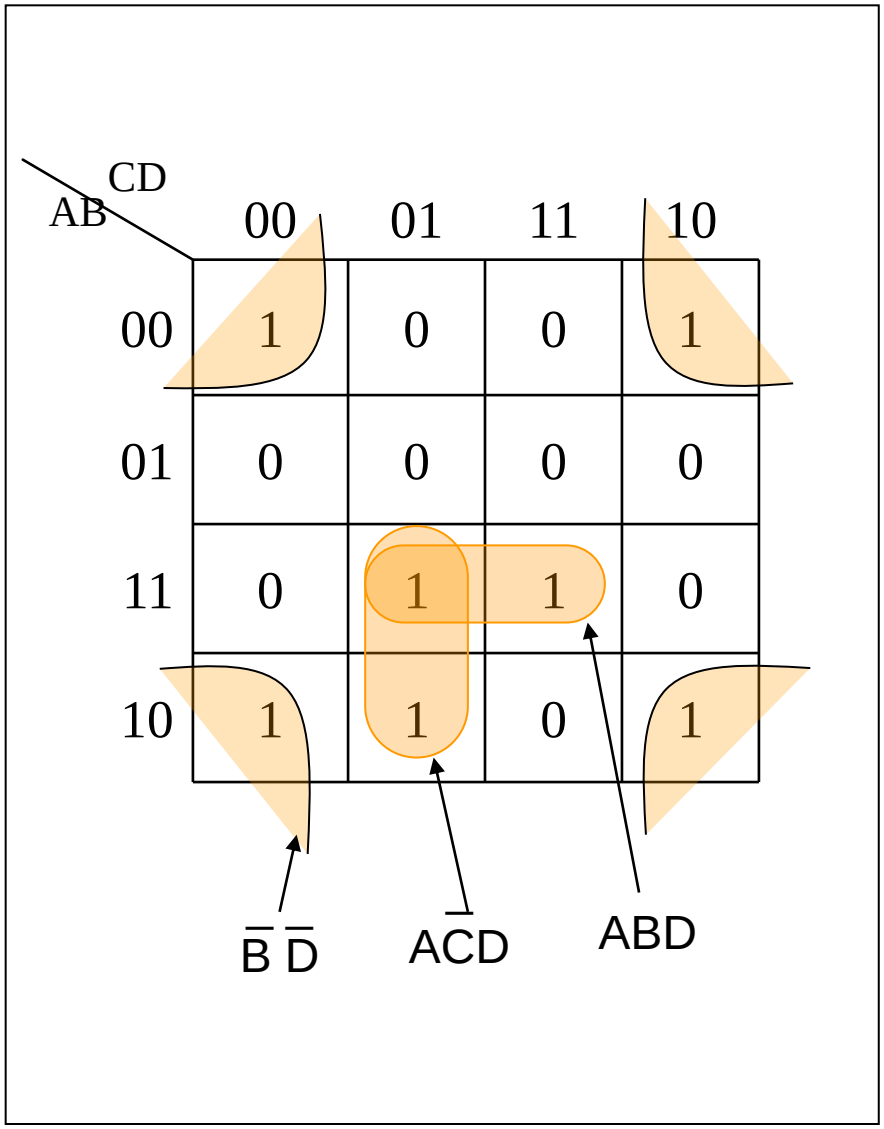


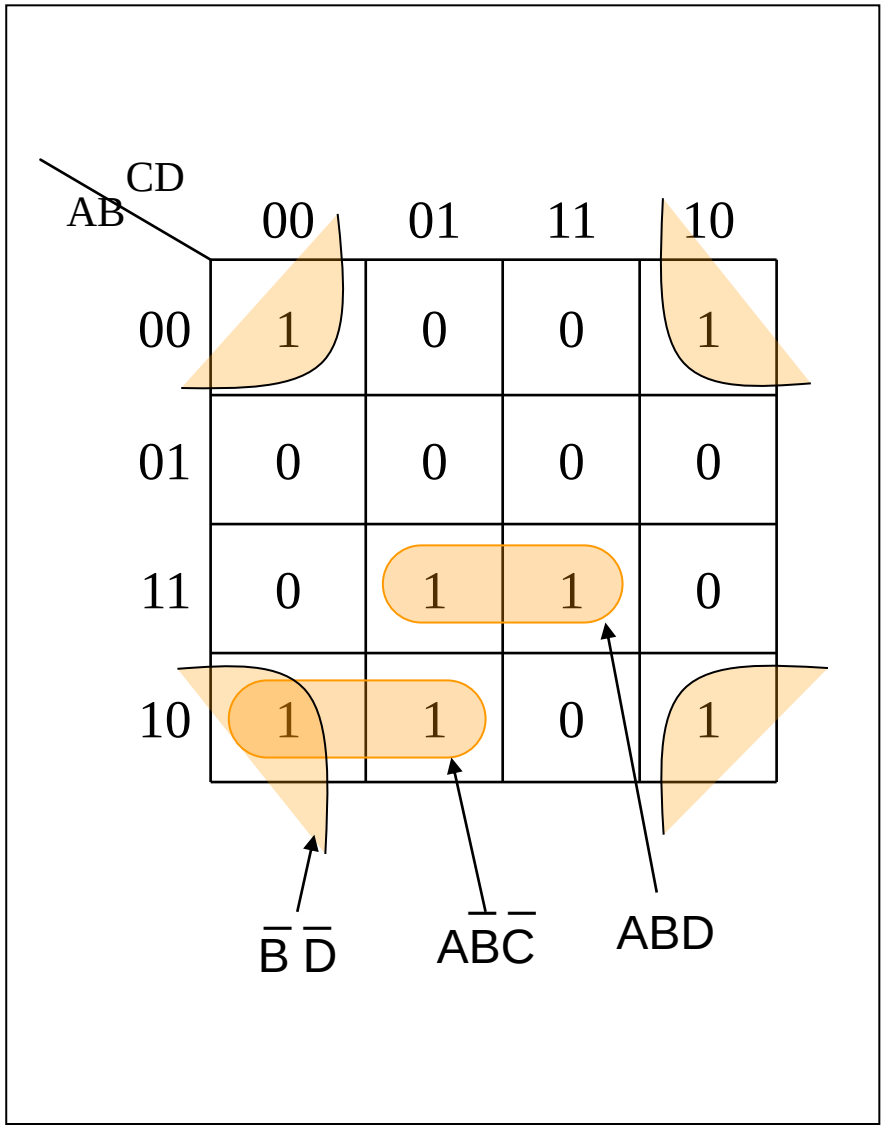

Funcția *imparitate*

# Adiacenţa liniilor/**coloanelor** extreme
# f(A,B,C)=Σ(0,2,3,4,5,6)

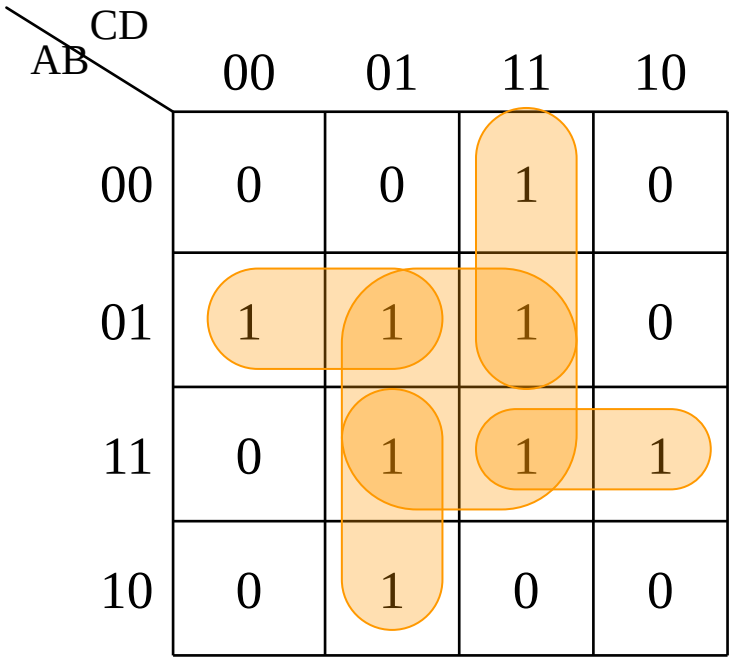


# Expresia depinde de grupare

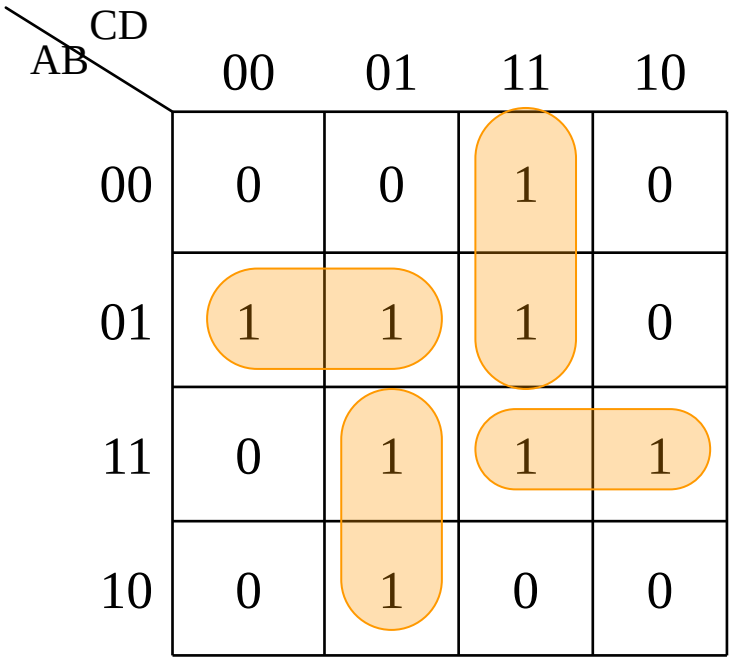


# Evitarea redundanţelor




Simplificare Karnaugh
neminimală



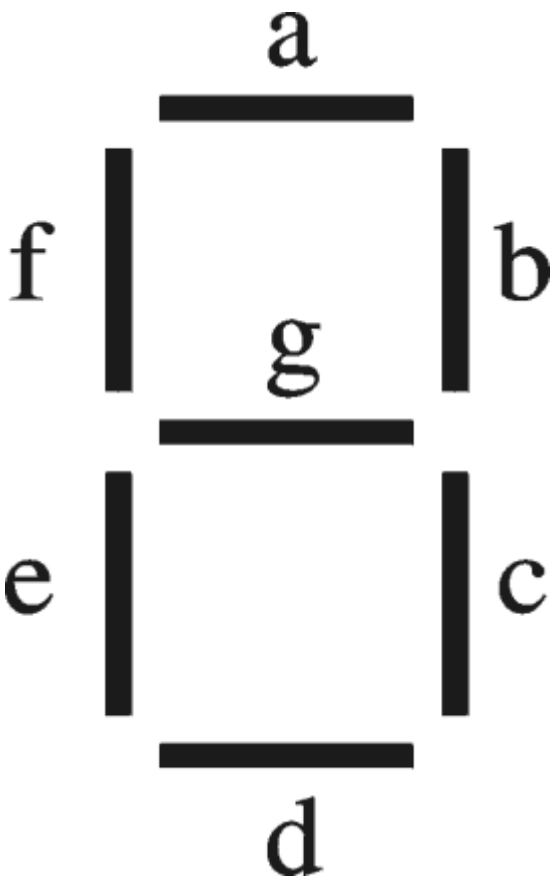

Simplificare Karnaugh
minimală

# Combinaţii imposibile de valori

- variabilele nu vor avea niciodată acele combinaţii de valori
- se poate deci considera **restricţia** funcţiei booleene doar la subdomeniul de definiţie al combinaţiilor permise
  - doar aceasta va fi "vizibilă" în funcţionarea circuitului
- se consideră cea mai convenabilă – din punctul de vedere al minimizării – extensie la combinaţiile imposibile
  - se consideră valoarea 0 sau 1, după cum convine

# Exemplu – afişarea cifrelor zecimale

- Afişaj cu 7 segmente
- Selectarea segmentelor pentru fiecare cifră
- 0 – stins
- 1 – aprins

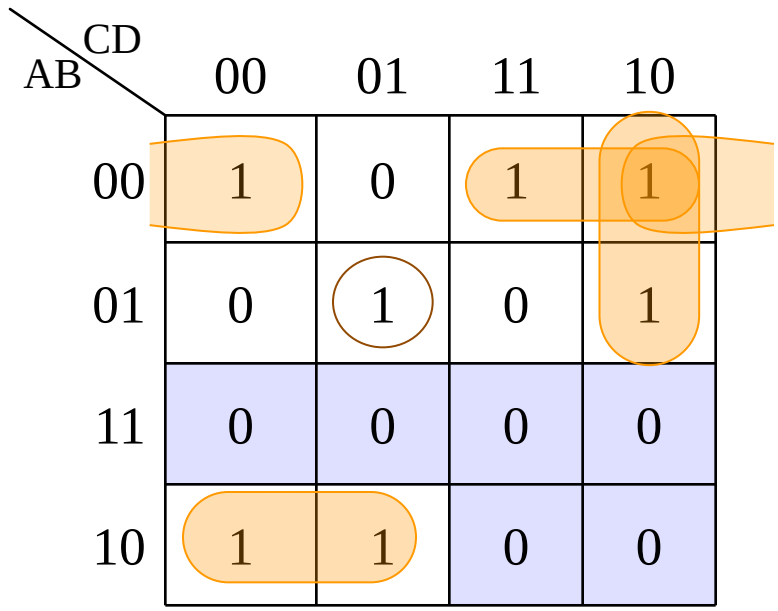


# Funcţia booleană ataşată segmentului **d**

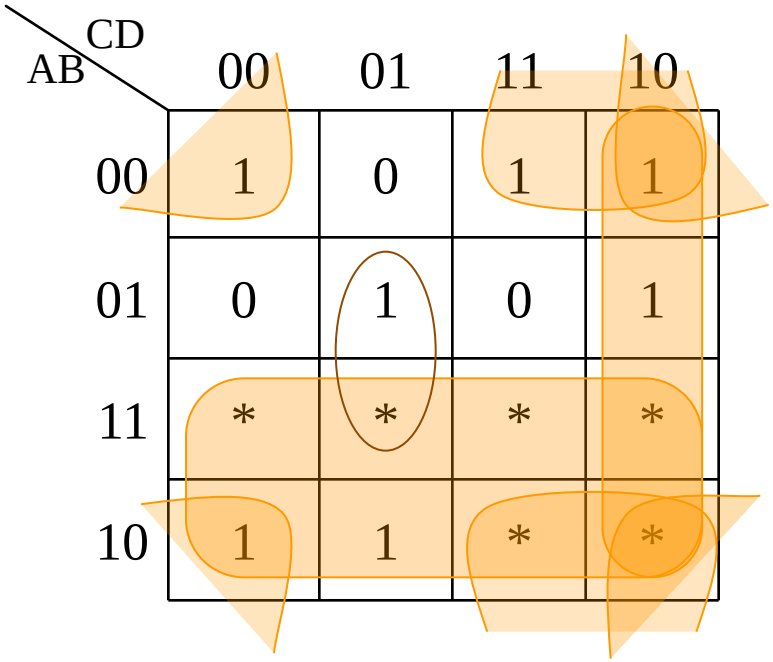
| Nr | A | B | C | D | d |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 0 |

| Nr | A | B | C | D | d |
|---|---|---|---|---|---|
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | * |
| 11 | 1 | 0 | 1 | 1 | * |
| 12 | 1 | 1 | 0 | 0 | * |
| 13 | 1 | 1 | 0 | 1 | * |
| 14 | 1 | 1 | 1 | 0 | * |
| 15 | 1 | 1 | 1 | 1 | * |

# Combinaţiile imposibile pot simplifica expresia




Simplificare “funcţionare de siguranţă”




Simplificare exploatând combinaţiile imposibile

# Temă: comparator pe 2 biţi

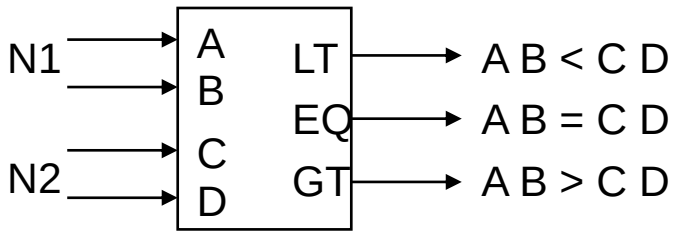


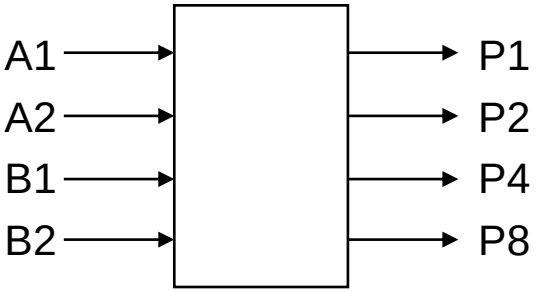

Diagrama bloc

şi tabelul de adevăr

| A | B | C | D | LT | EQ | GT |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

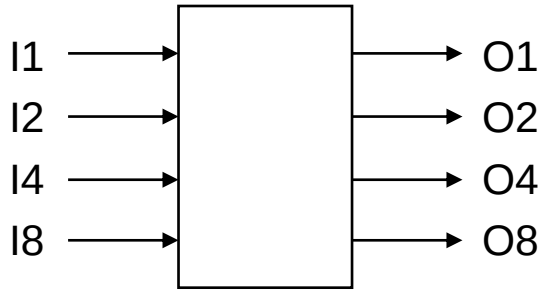
E nevoie de câte o reducere Karnaugh pentru fiecare dintre cele 3 funcţii

# Temă: multiplicator pe 2 biţi




| A2 | A1 | B2 | B1 | P8 | P4 | P2 | P1 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

# Temă: "incrementare cu 1 BCD"




| I8 | I4 | I2 | I1 | O8 | O4 | O2 | O1 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | * | * | * | * |
| 1 | 0 | 1 | 1 | * | * | * | * |
| 1 | 1 | 0 | 0 | * | * | * | * |
| 1 | 1 | 0 | 1 | * | * | * | * |
| 1 | 1 | 1 | 0 | * | * | * | * |
| 1 | 1 | 1 | 1 | * | * | * | * |

# II.6. CIRCUITE COMBINAŢIONALE

# Circuite combinaţionale

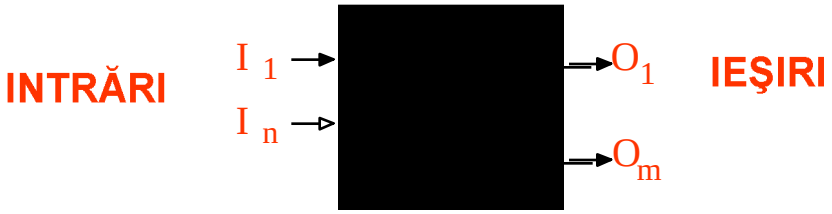



**Diagrama bloc a unui circuit combinaţional**

- Valorile de la ieşire depind doar de valorile de la intrare din momentul respectiv

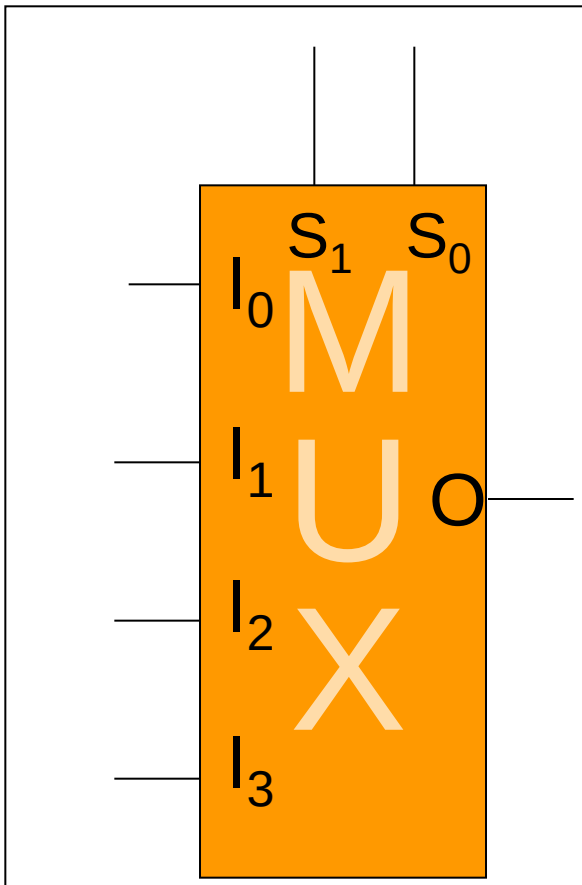
# II.6.1. MULTIPLEXORUL

# Multiplexorul

- $2^n$ intrări
- n intrări de selecţie (variabile de **control**)
  - biţi de control (de adresă)
- o singură ieşire
- fiecare intrare corespunde unui termen FND cu variabile de control
- controlul selectează o valoare de la intrare (bit) care devine valoare de ieşire

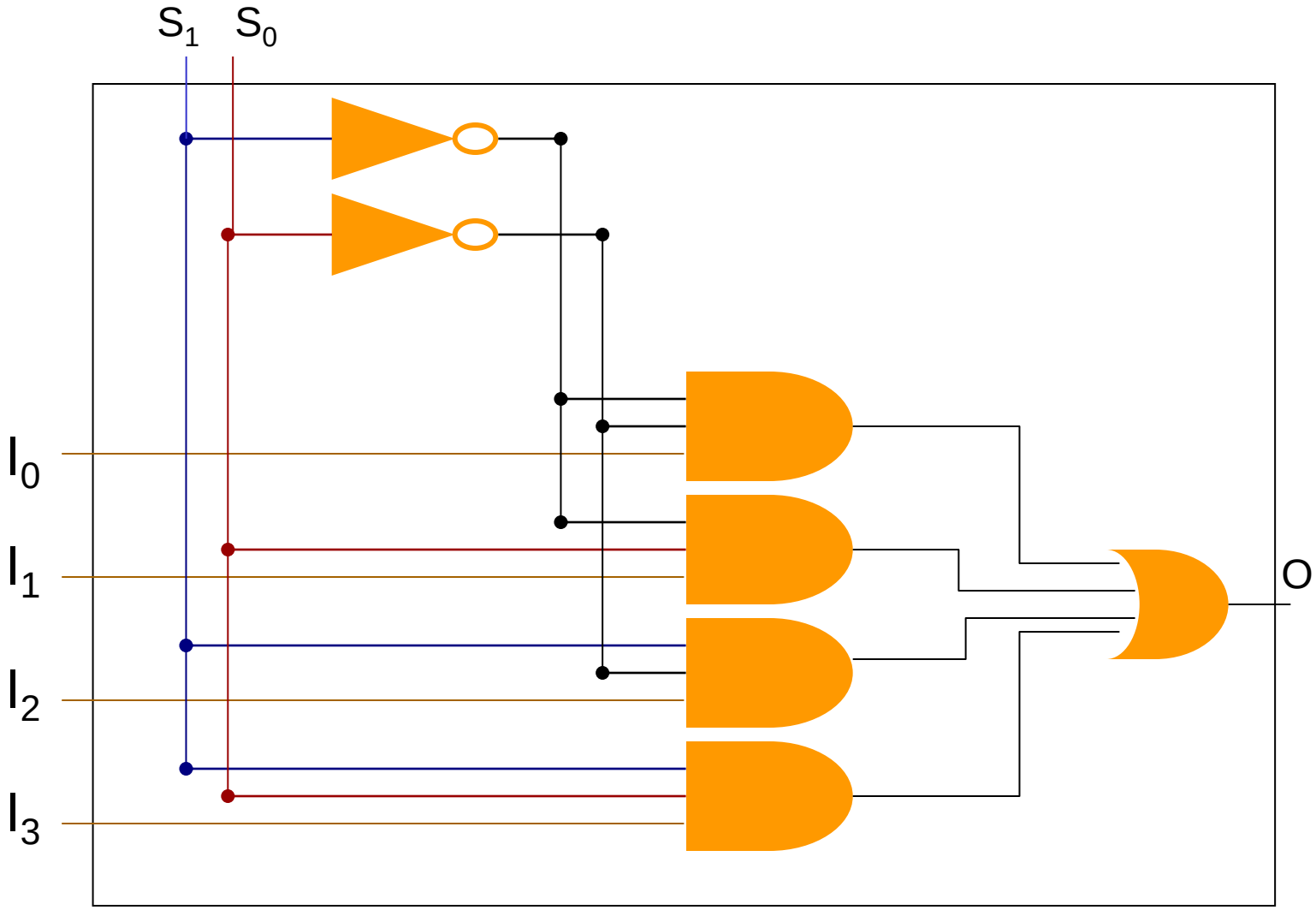
**MUX 4 → 1**

diagrama bloc şi tabelul de adevăr



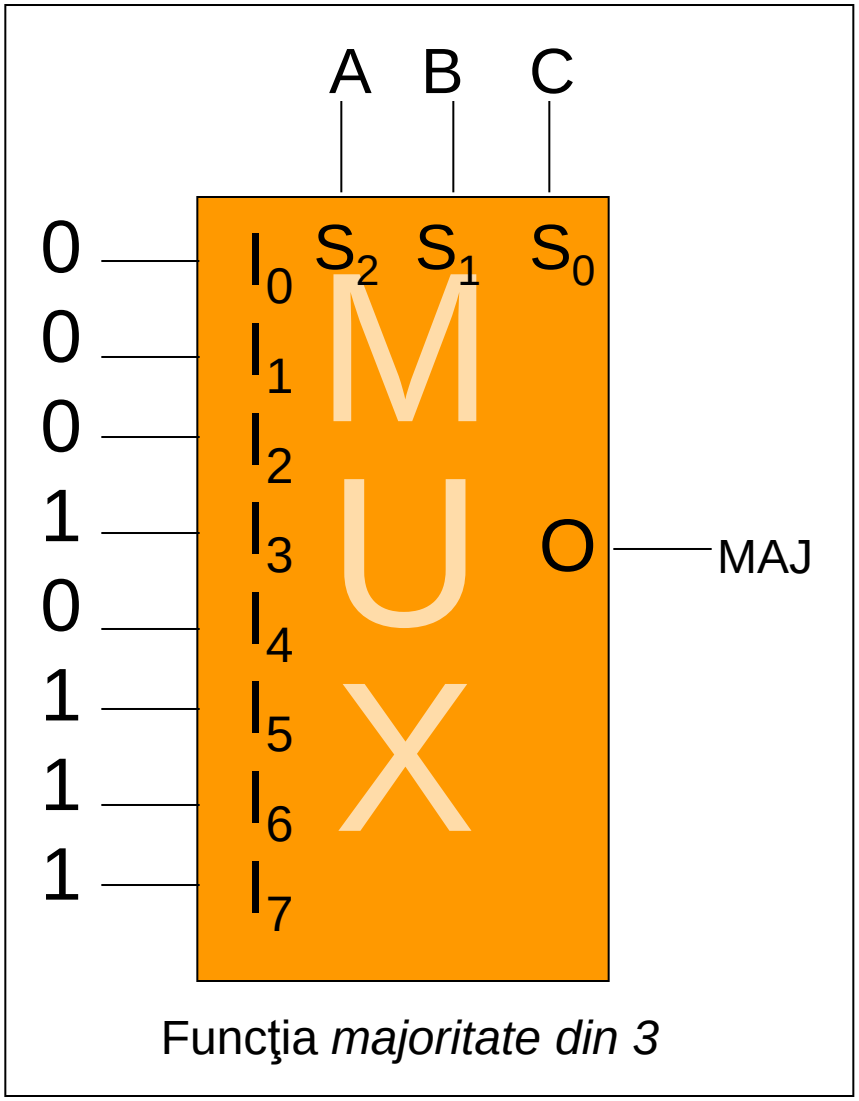

| $S_1$ | $S_0$ | O |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

# Multiplexorul 4→1: diagrama logică

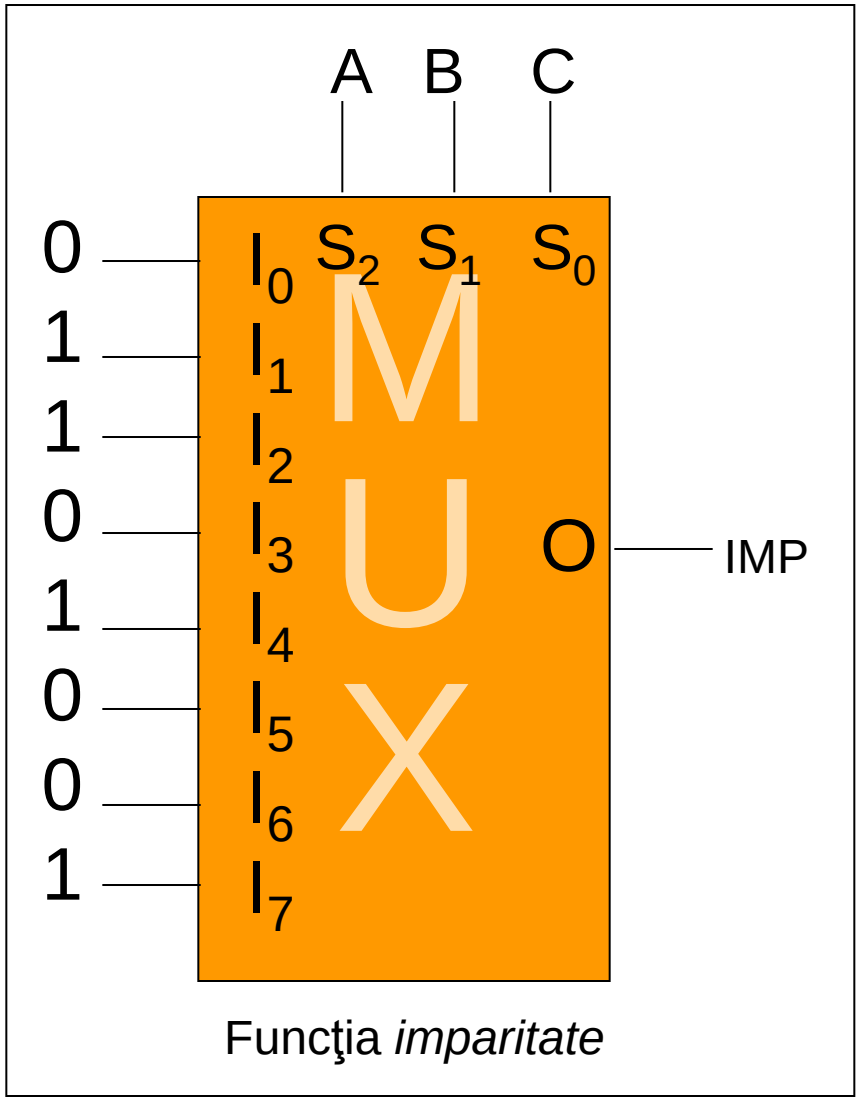


- Prima poartă AND poate avea la ieşire valorile 0 sau $I_0$, a doua 0 sau $I_1$ etc.
- Poarta OR poate avea la ieşire valorile $I_0$, $I_1$, $I_2$, $I_3$
  - De unde ideea de a folosi variabile de intrare ca valori de ieşire nu conectându-le direct la ieşirea circuitului, ci lăsându-le ca intrări şi selectându-le prin multiplexor

# Funcţii booleene pot fi implementate prin multiplexoare




Funcţia *majoritate din 3*




Funcţia *imparitate*

# "Majoritatea din 3": implementare eficientă prin multiplexor

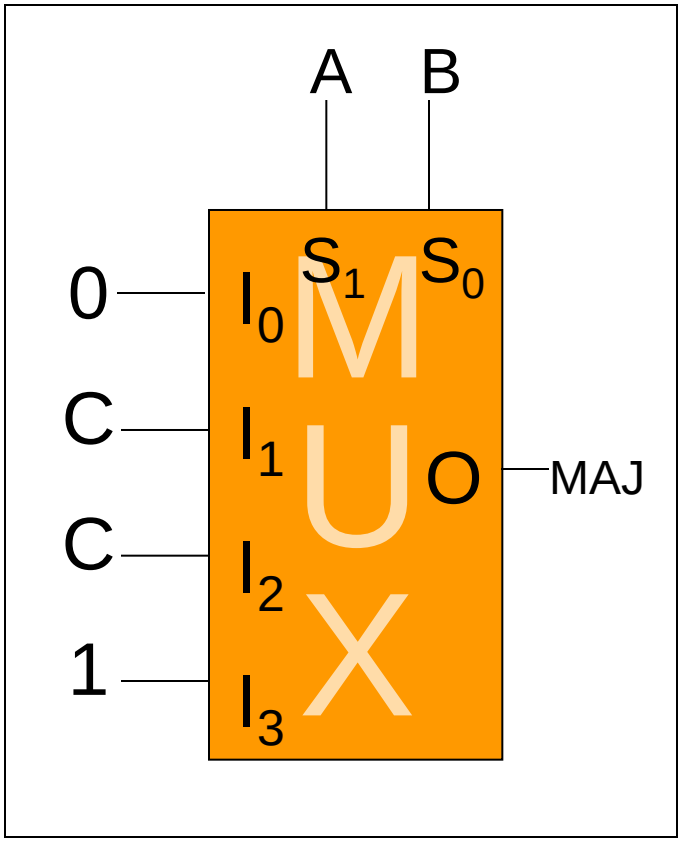
Eficient: *folding*

| A | B | C | MAJ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Tabel de adevăr originar

| A | B | MAJ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | C |
| 1 | 0 | C |
| 1 | 1 | 1 |

Tabel de adevăr pentru multiplexor



# "Imparitate": implementare eficientă prin multiplexor

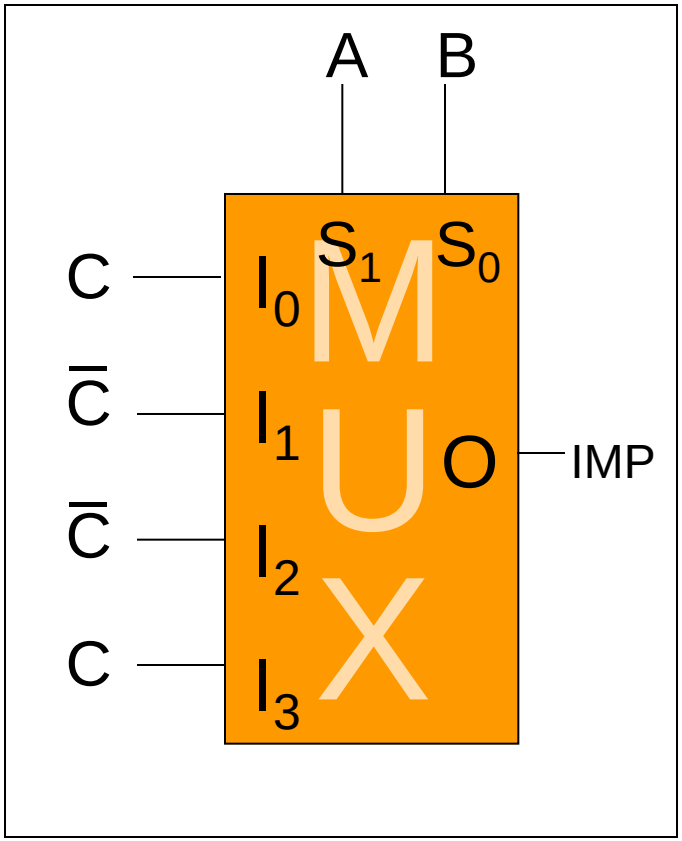
*folding*

| A | B | C | IMP |
|---|---|---|---|
| 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 | 0 |
| 1 | 1 | 1 | 1 |

Tabel de adevăr originar

| A | B | IMP |
|---|---|---|
| 0 | 0 | C |
| 0 | 1 | $\bar{C}$ |
| 1 | 0 | $\bar{C}$ |
| 1 | 1 | C |

Tabel de adevăr pentru multiplexor



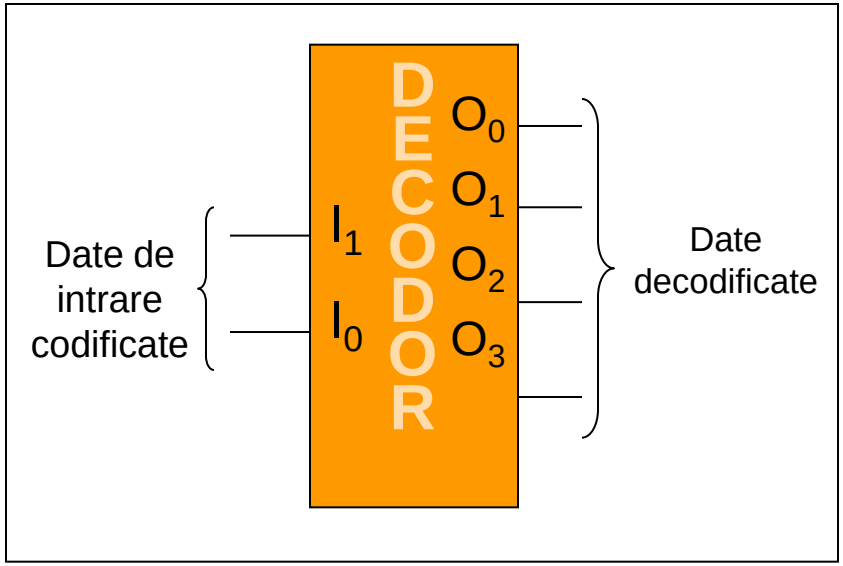
# II.6.2. DECODORUL, COMPARATORUL

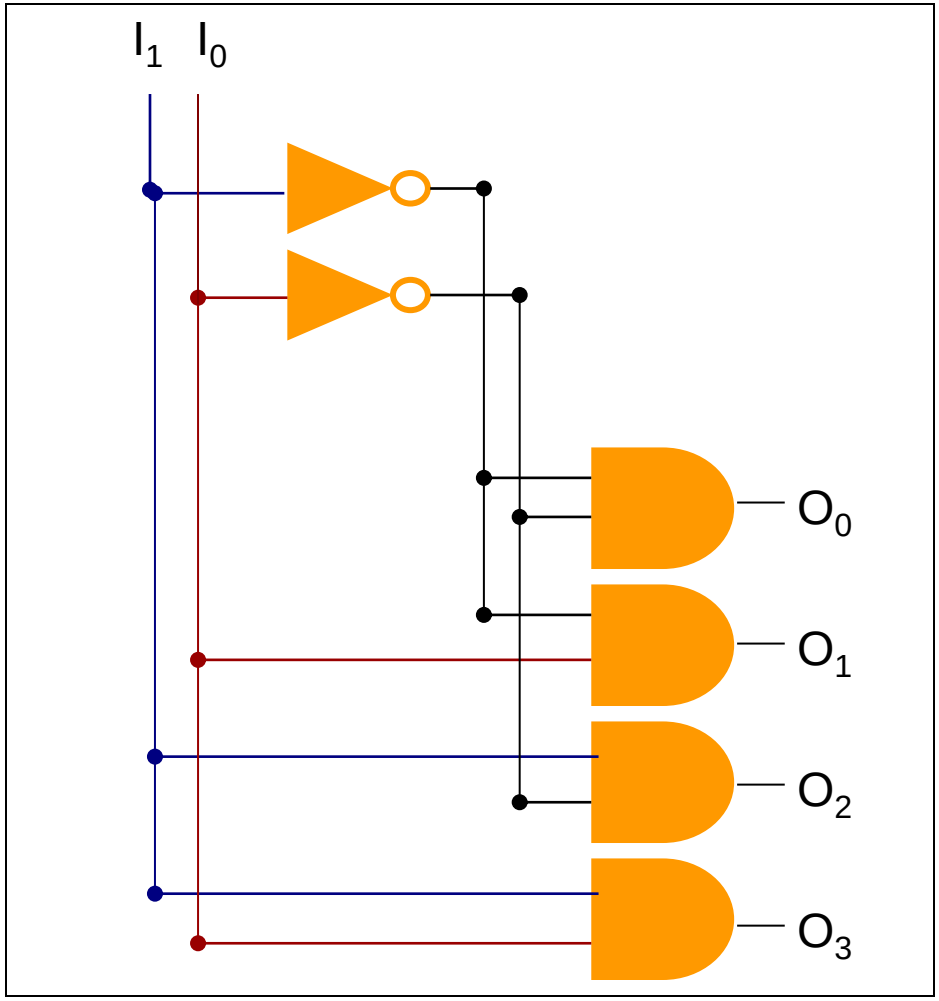
# Decodorul

- Decodorul are k intrări şi $2^k$ ieşiri
  - identificarea unei locaţii de memorie după adresă
- Circuitul activează în fiecare moment **una** din $2^k$ ieşiri
  - intrările au rolul controalelor de la multiplexor (selectează adrese)
  - fiecare ieşire corespunde unui termen FND scris cu variabilele de intrare

# Decodorul: k=2

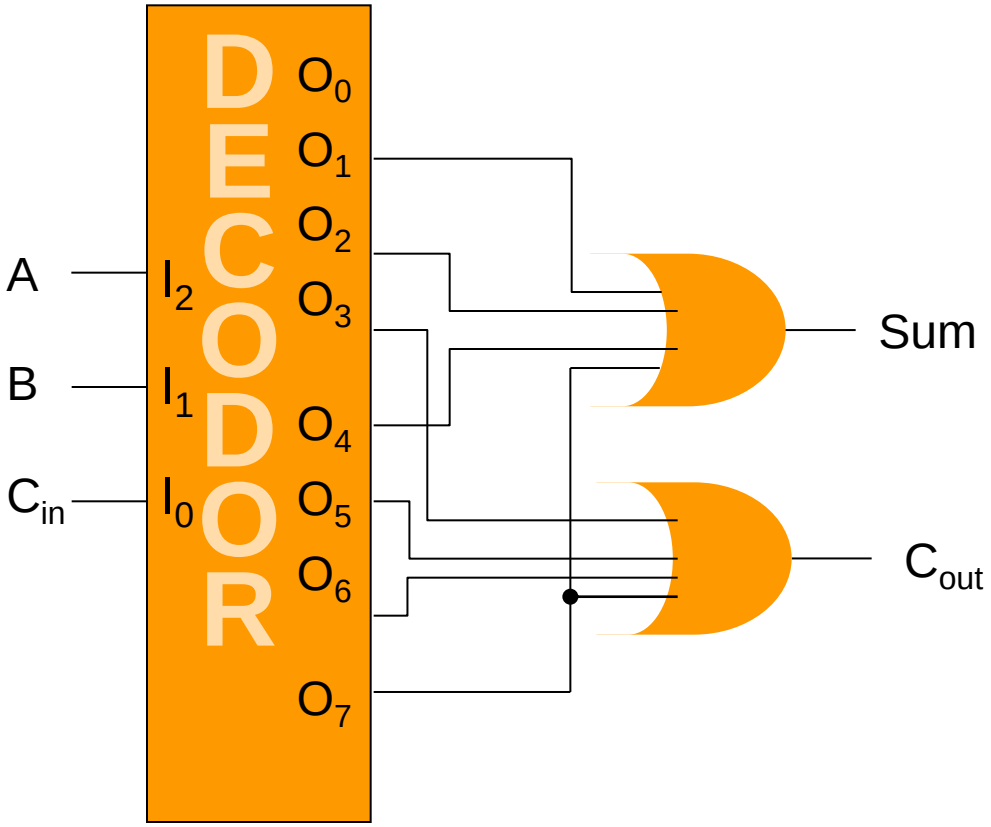
| $I_1$ | $I_0$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |



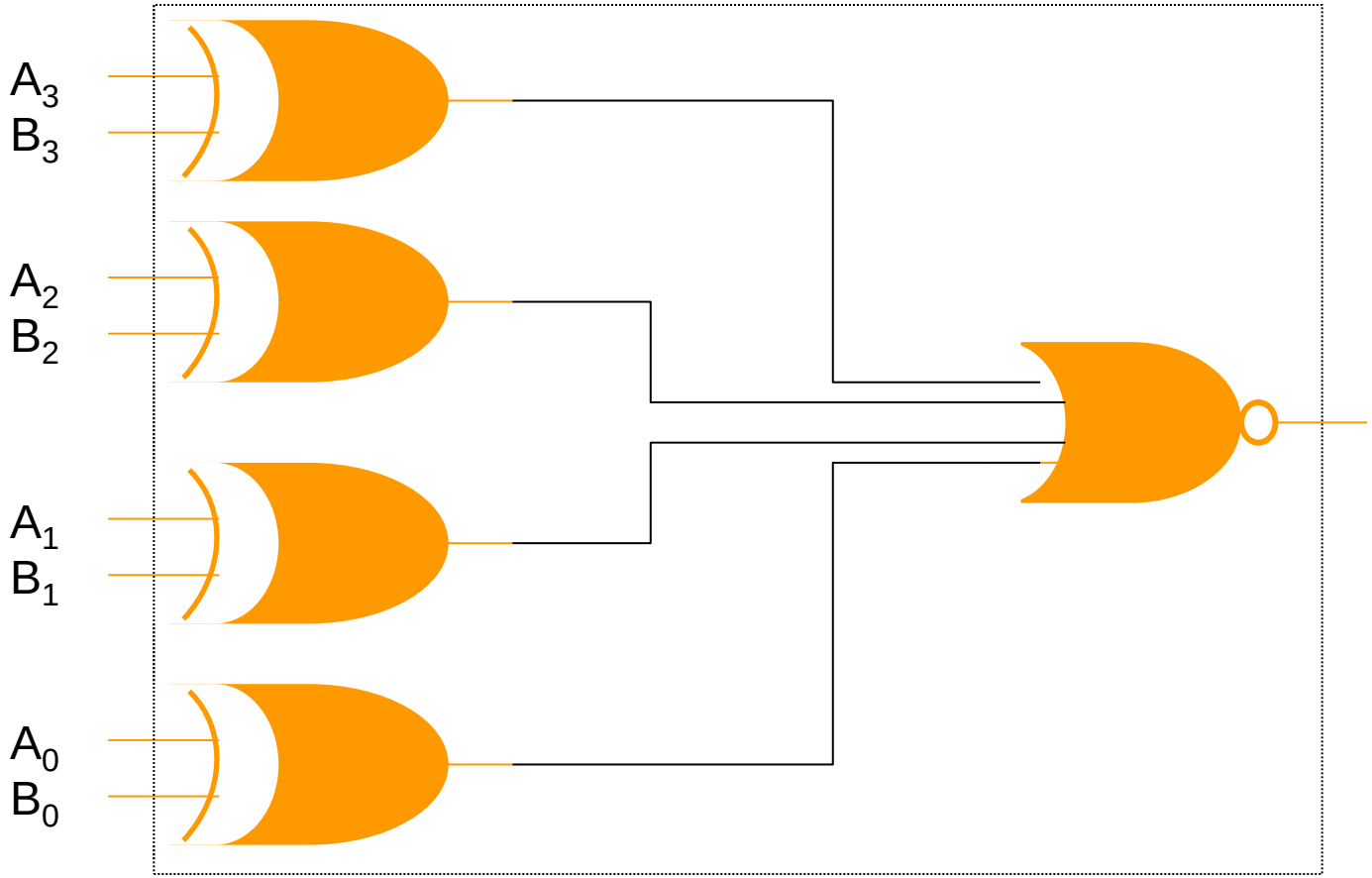
# Implementarea adunării prin decodoare

| A | B | $C_{in}$ | Sum | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |



# Circuit de comparare (Comparatorul)

- Implementează operatorii de comparare (= , > , < , ≥ , ≤)
- Exemplu: egalitate pe 4 biţi
  - Temă: comparator complet (<, =, >)



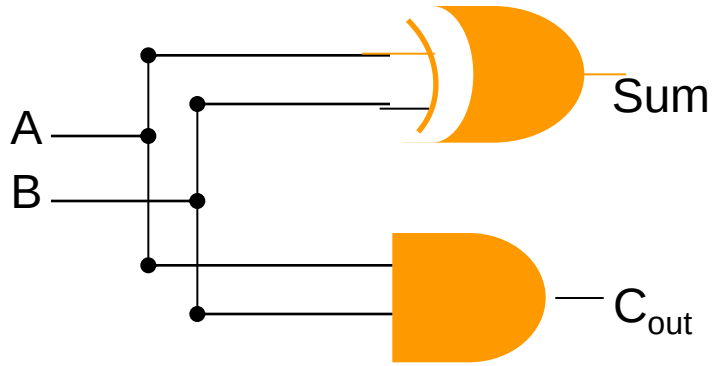
# II.6.3. SUMATOARE

# Semi-sumatorul şi sumatorul complet

- Semi-sumatorul (*half-adder*)
  - Adună cei doi biţi de intrare
    - Furnizează la ieşire un bit *sumă* şi un bit *transport*
  - Neajuns: nu poate fi extins pentru adunarea de numere mai lungi
- Sumatorul complet (*full adder*)
  - Adună cei trei biţi de intrare
    - Furnizează la ieşire tot un bit *sumă* şi un bit *transport*
  - Poate fi folosit pentru a construi sumatoare pe N biţi
    - conectând $C_{out}$ de la un sumator la $C_{in}$ al următorului.
    - Acesta va fi *sumatorul serial* (*ripple-carry adder*)

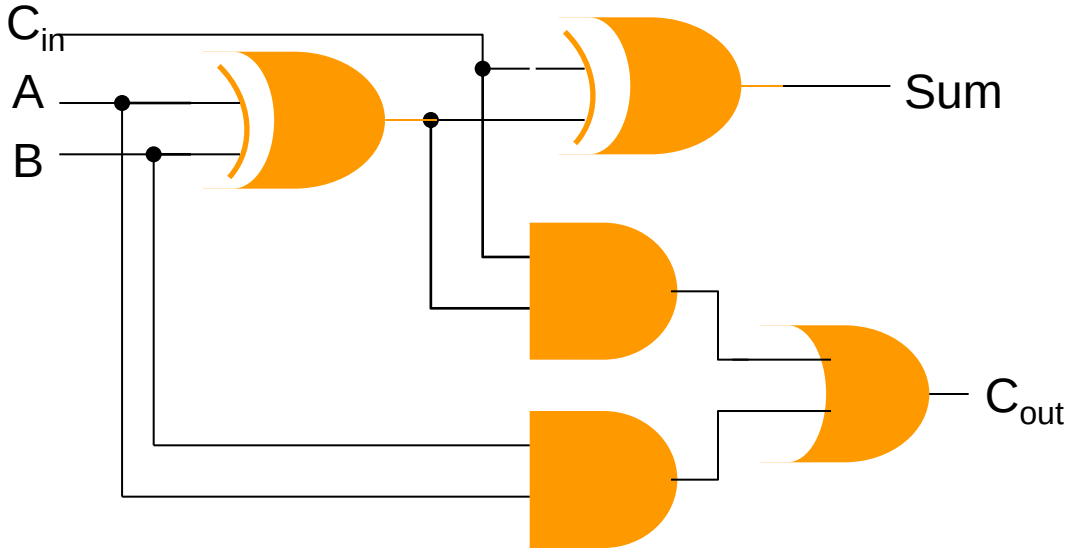
# Semi-sumator şi sumator: diagrame logice

| A | B | Sum | $C_{out}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



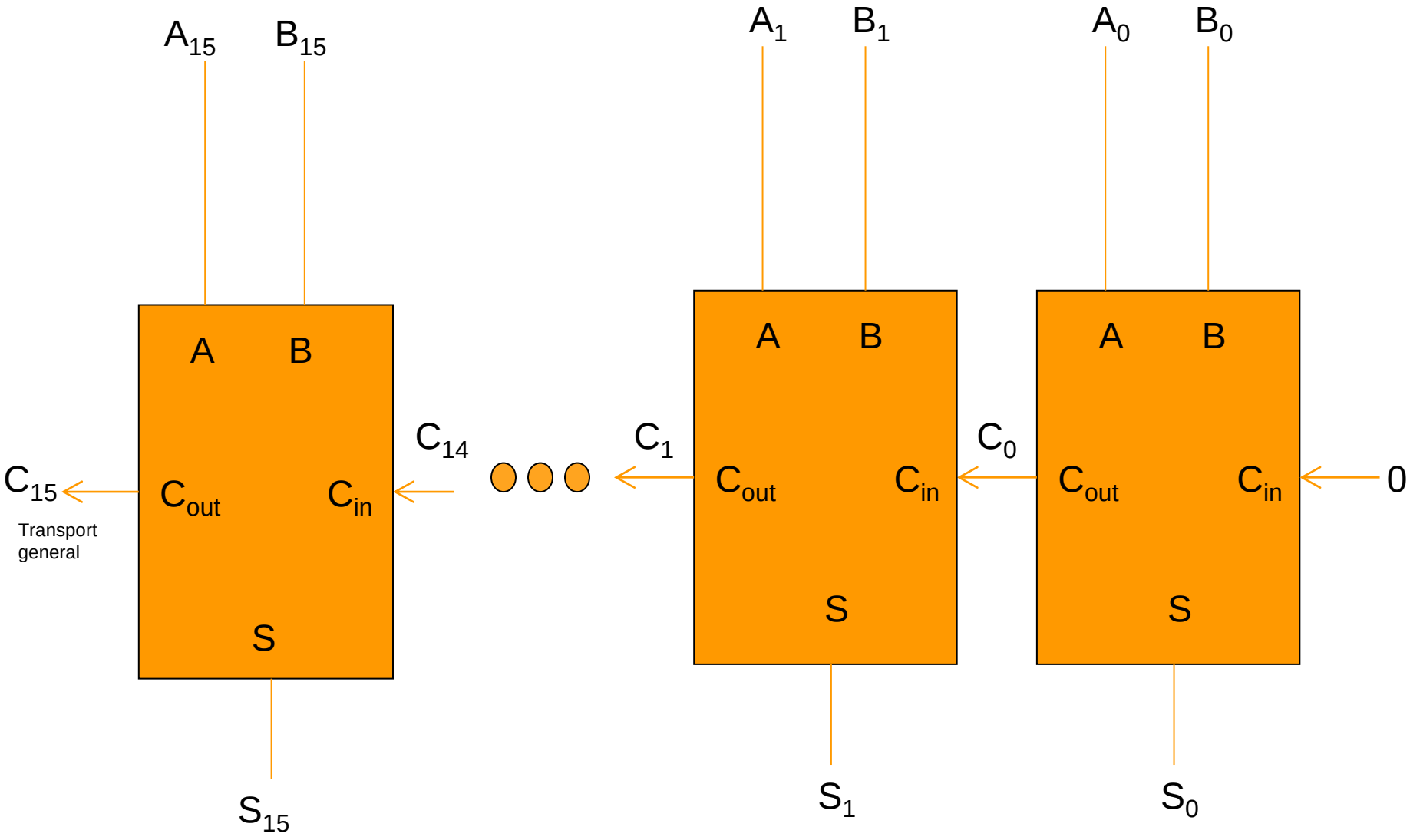

Tabelul de adevăr şi diagrama logică ale semi-sumatorului

| A | B | $C_{in}$ | Sum | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |




Tabelul de adevăr şi diagrama logică ale sumatorului complet

# Sumatorul serial pe 16 biţi (+)



# Sumatoare seriale

- Cu propagarea transportului
  - de la un rang la următoarele
- Şi primul sumator este complet
  - $C_0 = 0$ pentru adunare
  - $C_0 = 1$ pentru scădere
- Avantaj: circuite relativ simple, repetate identic la fiecare rang
- Dezavantaj: sumatoarele seriale pot fi lente
  - întârzierea - proporţională cu numărul de biţi
- Cazul cel mai relevant:
  **11111111 + 00000001**

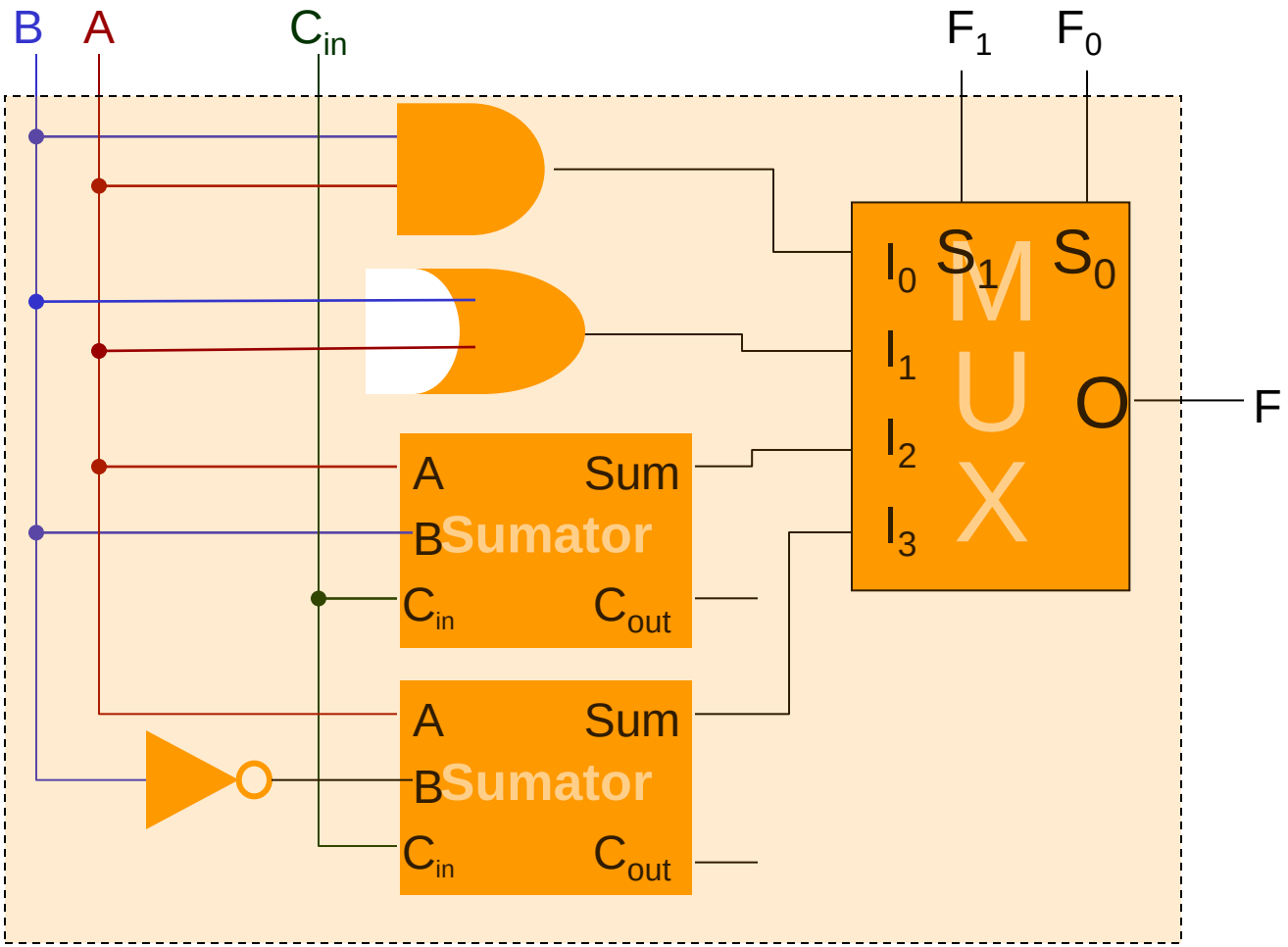
# Accelerarea adunării

- Sumatoare cu anticiparea transportului
  - *Carry lookahead adders*
  - Elimină întârzierea datorată propagării transportului
  - Transportul-intrare (carry-in) se generează independent pentru fiecare rang
    - $C_0 = A_0 B_0$
    - $C_1 = A_0 B_0 A_1 + A_0 B_0 B_1 + A_1 B_1$
    - . . .
    - $C_i = G_i + P_i \cdot C_{i-1} = A_i \cdot B_i + (A_i + B_i) \cdot C_{i-1} = \ldots$
    - …
  - Necesită circuite complexe
  - De obicei, se utilizează o combinaţie de tehnici de anticipare şi propagare
- Sumatoare cu selecţia transportului
  - Exemplu: pentru 32 de biţi, fiecare octet e "adunat" de două sumatoare ($C_0 = 0$ şi respectiv $C_0 = 1$), apoi se selectează $S_i$ corect

# II.6.4. O UNITATE ARITMETICĂ ŞI LOGICĂ ELEMENTARĂ

# Unitate Aritmetică şi Logică (1 bit): AND, OR, +, -

## Proiectare iniţială

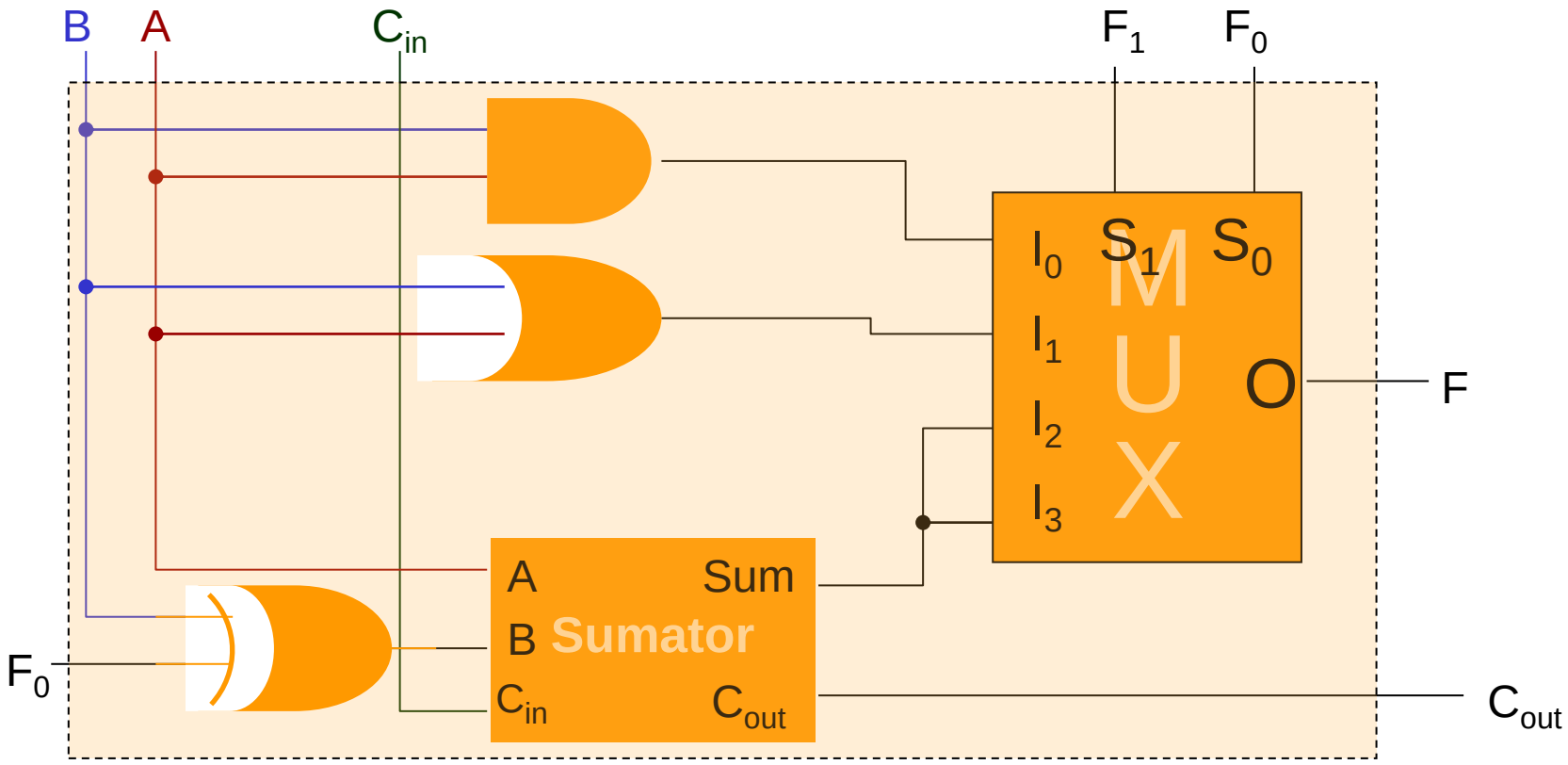



| $F_1$ | $F_0$ | F |
|---|---|---|
| 0 | 0 | A and B |
| 0 | 1 | A or B |
| 1 | 0 | A+B |
| 1 | 1 | A-B |

Semnale de control: $F_0$ $F_1$

# Unitate Aritmetică şi Logică (1 bit): AND, OR, +, -

Proiectare îmbunătăţită



# Unitate Aritmetică şi Logică (16 biţi): AND, OR, +, -