

# II.4. Minimizarea funcțiilor booleene prin metoda diagramelor Karnaugh

# Metoda Veitch-Karnaugh

- oferă posibilitatea de a grupa vizual termenii conjuncție din FND pentru care se poate aplica unificarea
- pentru unificarea a doi termeni, aceștia trebuie să difere pe o singură variabilă
  - la un termen apare negată, la celălalt nenegată
- asemenea termeni devin vecini într-o diagramă Karnaugh

# Structura diagramei Karnaugh

- tabel bidimensional
- numele variabilelor
  - pe linii, respectiv coloane
- zona etichetelor
  - etichetă - șir de  $n$  biți
  - fiecare bit corespunde unei variabile (intrări)
  - apar toate combinațiile posibile de valori
- zona valorilor funcției (ieșiri)

# Exemple de diagrame

2 variabile

A \ B	0	1
	0	1
0	1	1
1		

3 variabile

A \ BC	00	01	11	10
	0	1	1	1
0		1		1
1		1	1	

4 variabile

AB \ CD	00	01	11	10
	00	01	11	10
00	1	1		
01				
11		1		1
10	1			1

## Codul Grey

- etichetele nu se scriu în ordinea naturală, ci în ordinea Grey
- oricare două etichete consecutive, inclusiv prima și ultima, diferă printr-un singur bit
  - 2 biți: 00, 01, 11, 10
  - 3 biți: 000, 001, 011, 010, 110, 111, 101, 100
  - 4 biți: 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000

## Adiacențe în diagrame (1)

- două poziții sunt adiacente dacă etichetele corespunzătoare diferă pe un singur bit
  - codul Grey translează adiacența în vecinătate
- pentru o funcție de  $n$  variabile, o locație are  $n$  locații adiacente
  - $n < 5$ : locațiile adiacente locației date se determină vizual (sus, jos, stânga, dreapta)
  - $n \geq 5$ : și alte adiacențe decât cele vizibile direct

## Adiacențe în diagrame (2)

- pot fi mai mult de 2 locații adiacente
  - extinderea unificării la mai mult de 2 variabile
- în diagramele Karnaugh, acesta corespund unor blocuri de  $2^k$  locații
  - putere a lui 2 atât pe linii, cât și pe coloane
    - inclusiv puterea 0
    - formă dreptunghiulară
  - pentru fiecare locație, blocul trebuie să conțină exact  $k$  locații adiacente cu ea

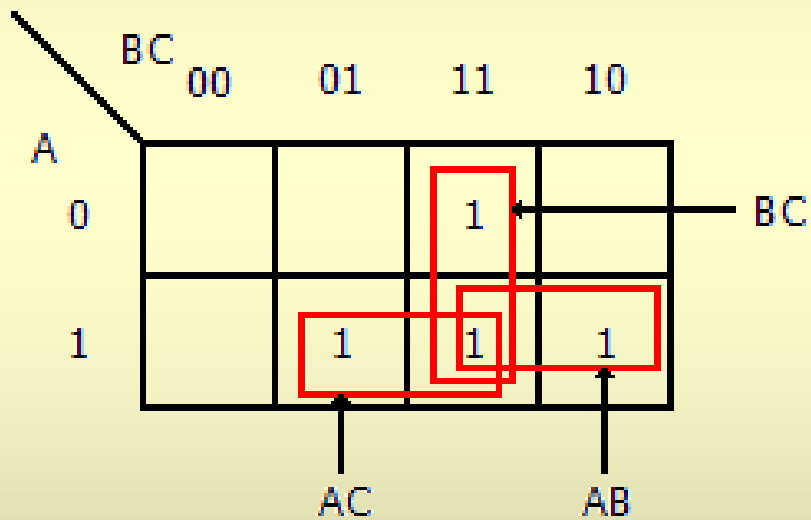
# Minimizare Karnaugh

- se caută blocuri conținând numai valori 1
  - corespunzătoare unor adiacențe (v. anterior)
  - blocuri cât mai mari și mai puține
- pentru fiecare bloc cu  $2^k$  locații 1
  - avem un termen conjuncție cu  $n-k$  variabile
  - conține variabilele cu valori constante pentru toate locațiile din bloc
    - constant 0: variabilă negată; constant 1: nenegată
  - toți acești termeni sunt legați prin disjuncție

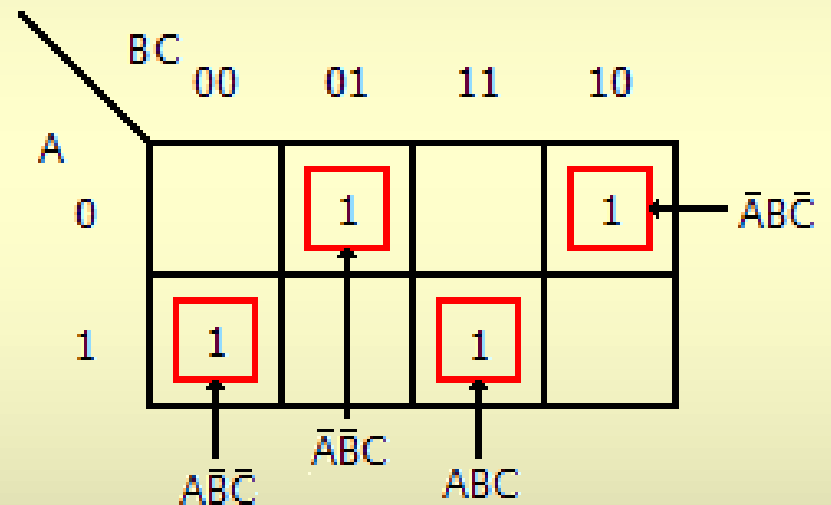


# Example

majoritate din 3

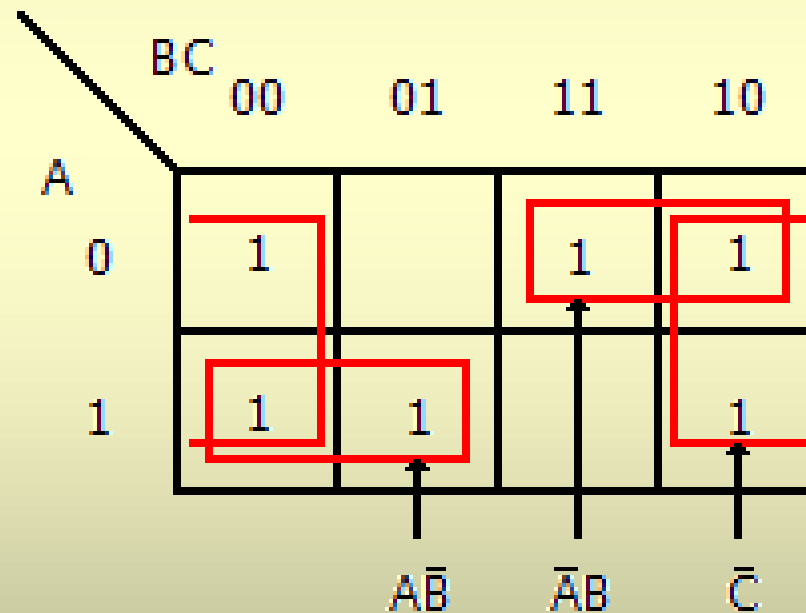


imparitate

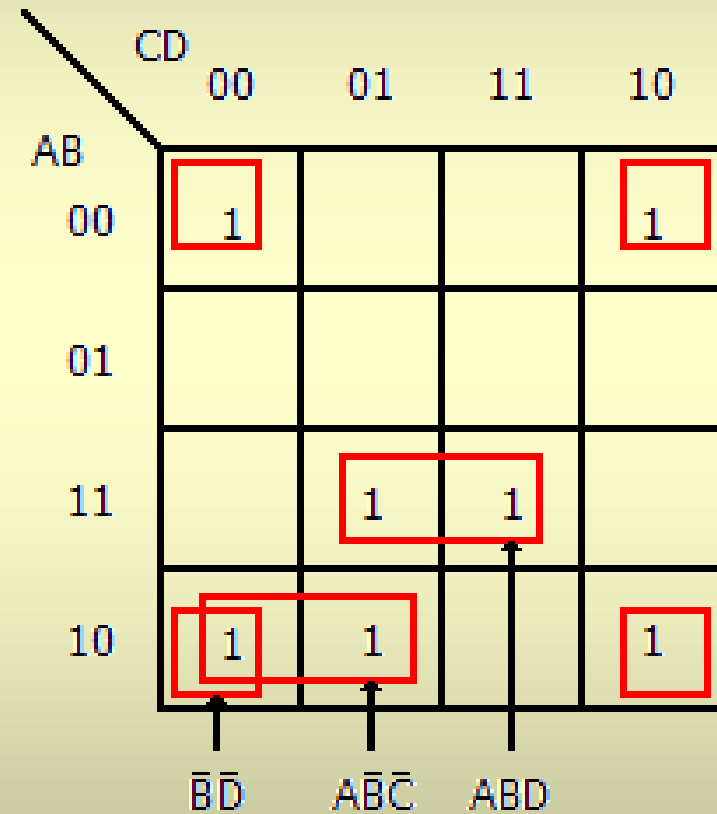
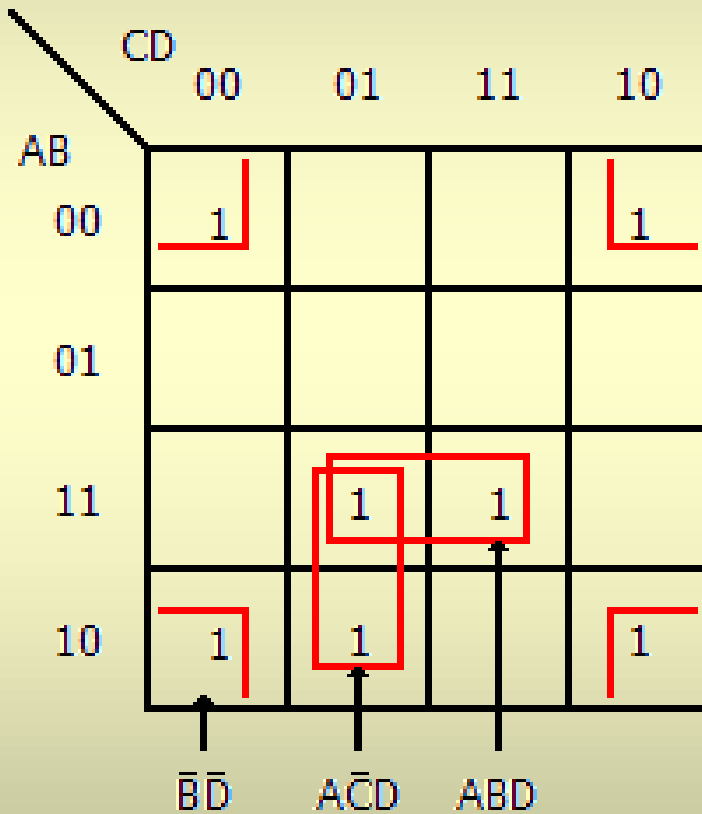


# Adiacența liniilor/coloanelor extreme

$$f = \Sigma(0,2,3,4,5,6)$$



# Expresia depinde de grupare



# Evitarea redundanțelor

simplificare neminimală

		CD			
		00	01	11	10
AB	00			1	
	01	1	1	1	
	11		1	1	1
	10		1		

simplificare minimală

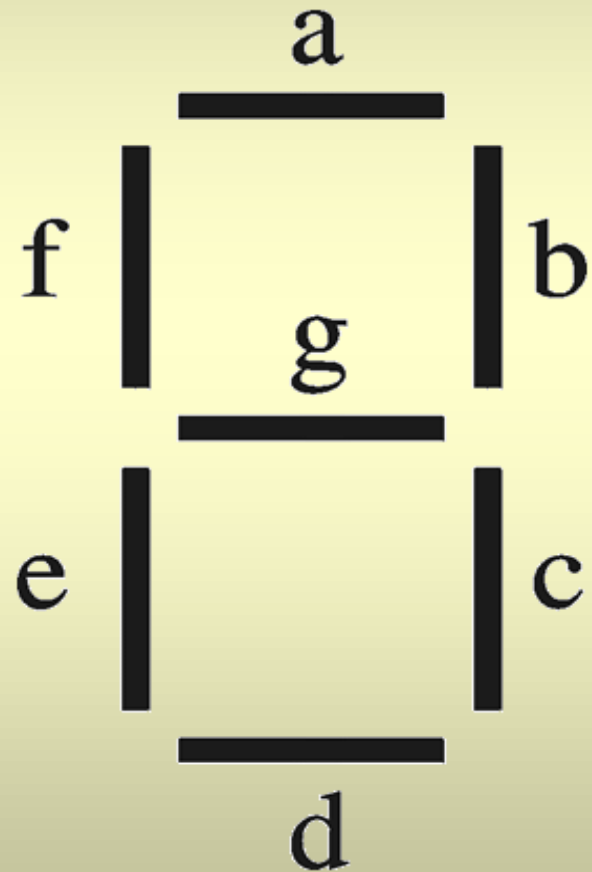
		CD			
		00	01	11	10
AB	00			1	
	01	1	1	1	
	11		1	1	1
	10		1		

# Combinații imposibile de valori

- anumite combinații de valori nu vor apărea niciodată la intrări
  - din definiția comportamentului dorit
  - dar diagrama trebuie completată pentru toate combinațiile de valori ale variabilelor
- în locațiile corespunzătoare acestor combinații se poate trece 0 sau 1
  - astfel încât să obținem o expresie cât mai simplă

## Exemplu - afișaj zecimal

- afișaj cu 7 segmente
- selectarea segmentelor pentru fiecare cifră
  - 0 - stins
  - 1 - aprins
- comanda pe intrare - 4 variabile
  - o cifră zecimală se poate scrie pe 4 biți



# Segmentul $d$ - tabel de adevăr

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	*

# Expresii mai simple

"funcționare de siguranță"

AB \ CD				
	00	01	11	10
00	1		1	1
01		1		1
11				
10	1	1		

combinații imposibile

AB \ CD				
	00	01	11	10
00	1		1	1
01		1		1
11	*	*	*	*
10	1	1	*	*



# Temă: comparator pe 2 biți

- 4 variabile: A, B, C, D
- formează 2 numere
  - $N_1 = AB$
  - $N_2 = CD$
- 3 ieșiri - corespund valorilor de adevăr
  - $LT = (N_1 < N_2)$
  - $EQ = (N_1 = N_2)$
  - $GT = (N_1 > N_2)$

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

# Temă: multiplicator pe 2 biți

- 4 variabile: A, B, C, D
- formează 2 numere
  - $N_1 = AB$
  - $N_2 = CD$
- 4 ieșiri - formează produsul  $N_1 \cdot N_2$

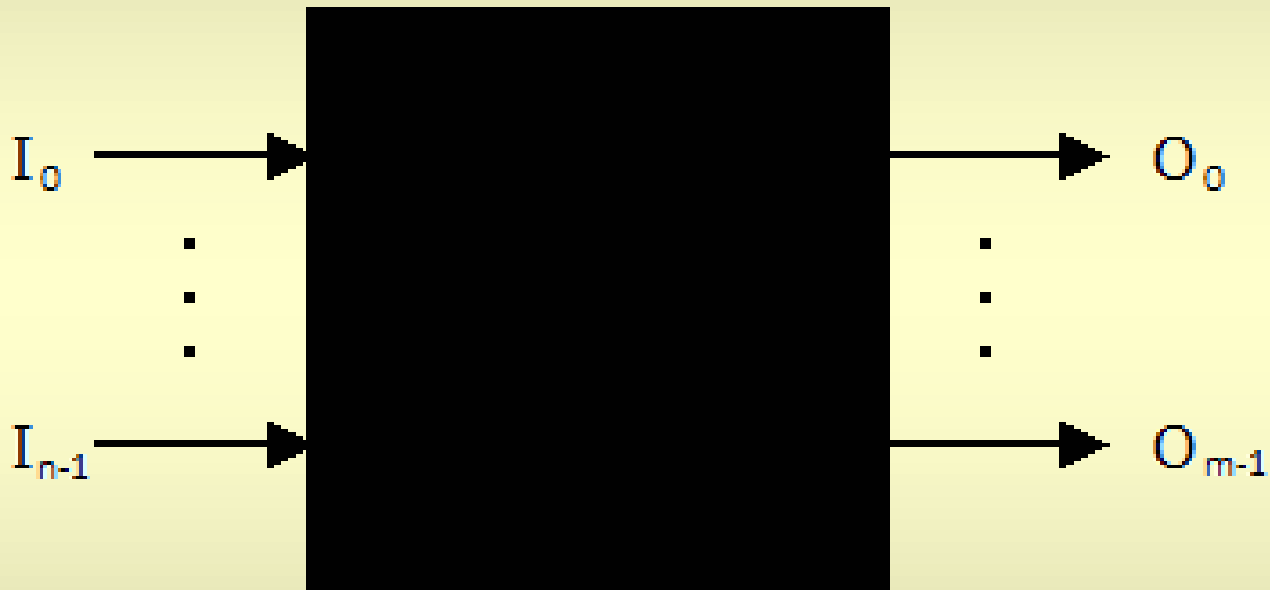
A	B	C	D	P8	P4	P2	P1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

# Temă: incrementare cu 1 BCD

- 4 variabile
  - formează un număr BCD
  - între 0 și 9
- 4 ieșiri - numărul de la intrare incrementat
  - rezultatul este tot un număr 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.5. Circuite combinazionale

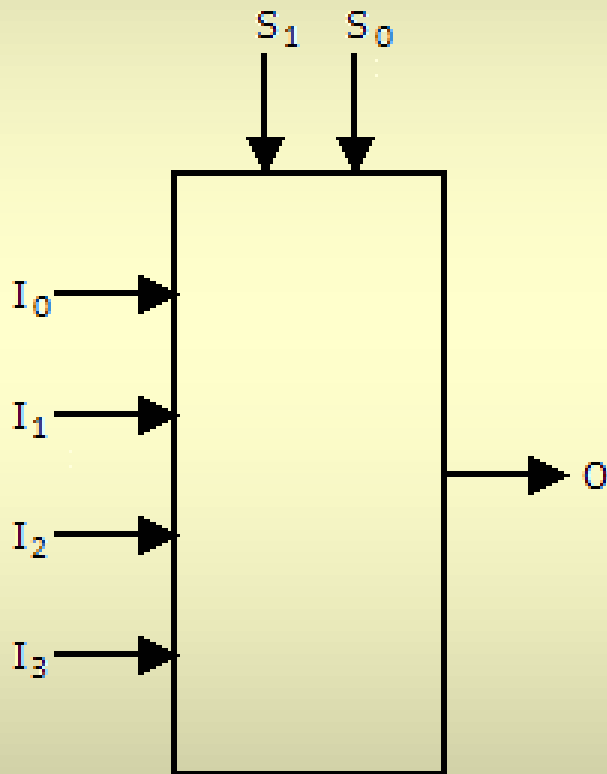


- valorile de la ieșire depind doar de valorile de la intrare din momentul respectiv

# Multiplexorul

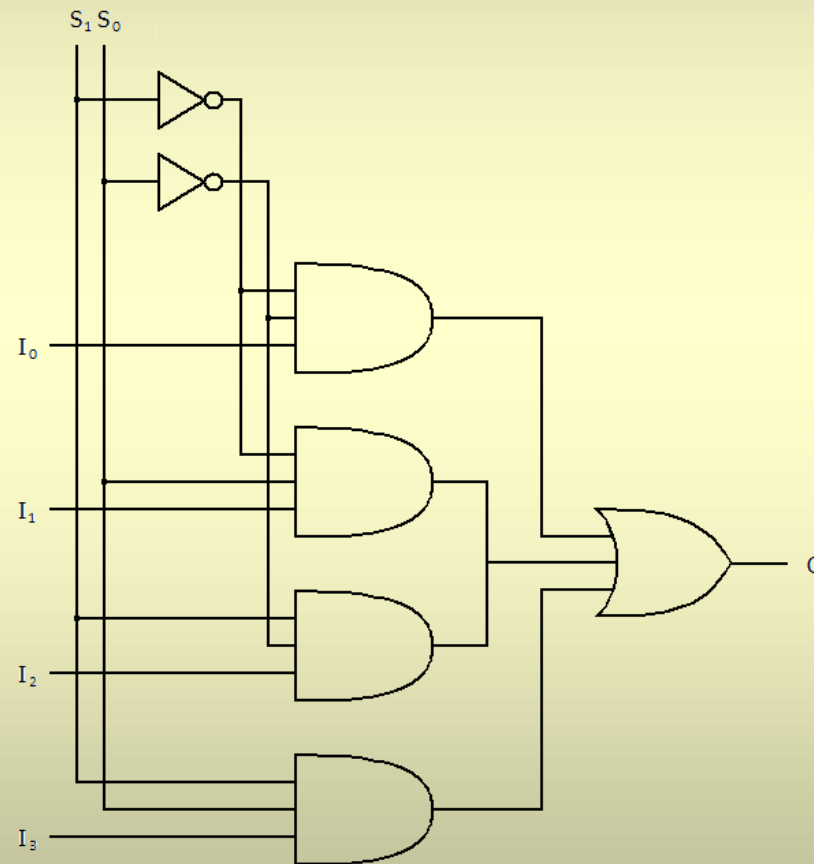
- $2^n$  intrări (de date)
- $n$  intrări de selecție (variabile de control)
  - biți de control (de adresă)
- o singură ieșire
- fiecare intrare corespunde unui termen FND cu variabile de control
- una dintre intrări (bit) este selectată - devine valoare de ieșire

# Multiplexor 4→1 ( $n=2$ )



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

# Diagrama logică (4→1)

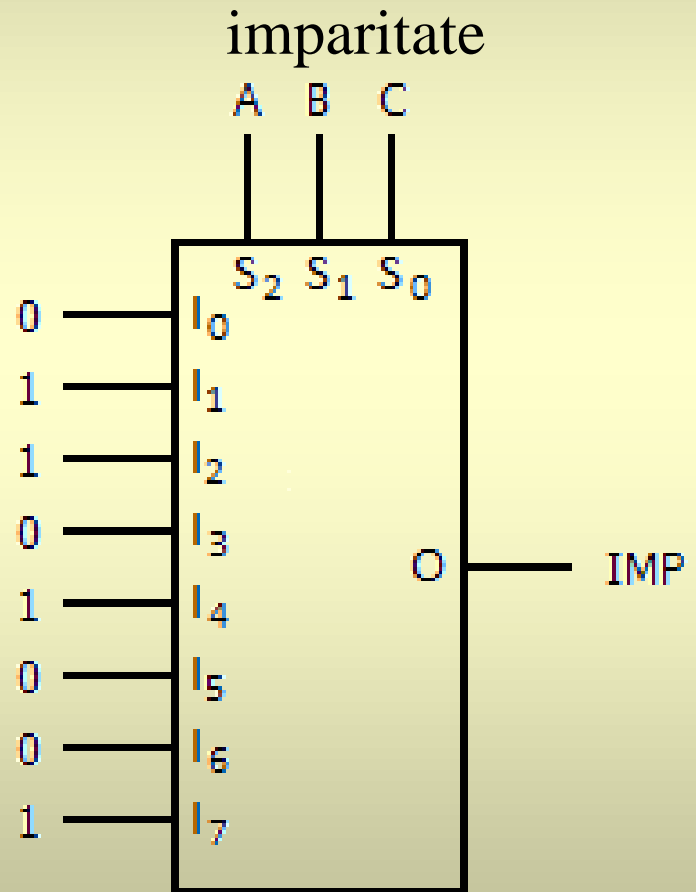
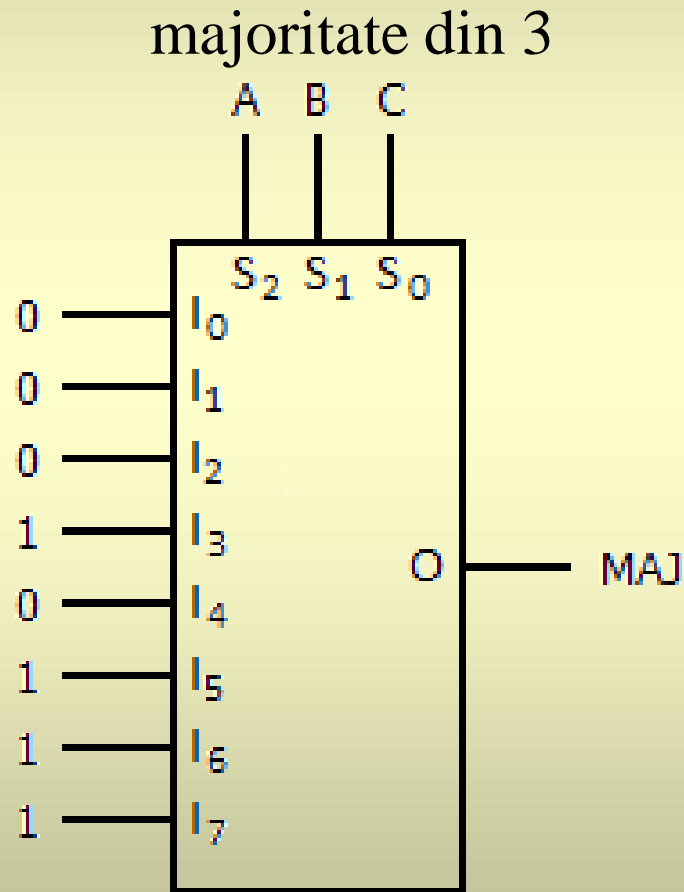




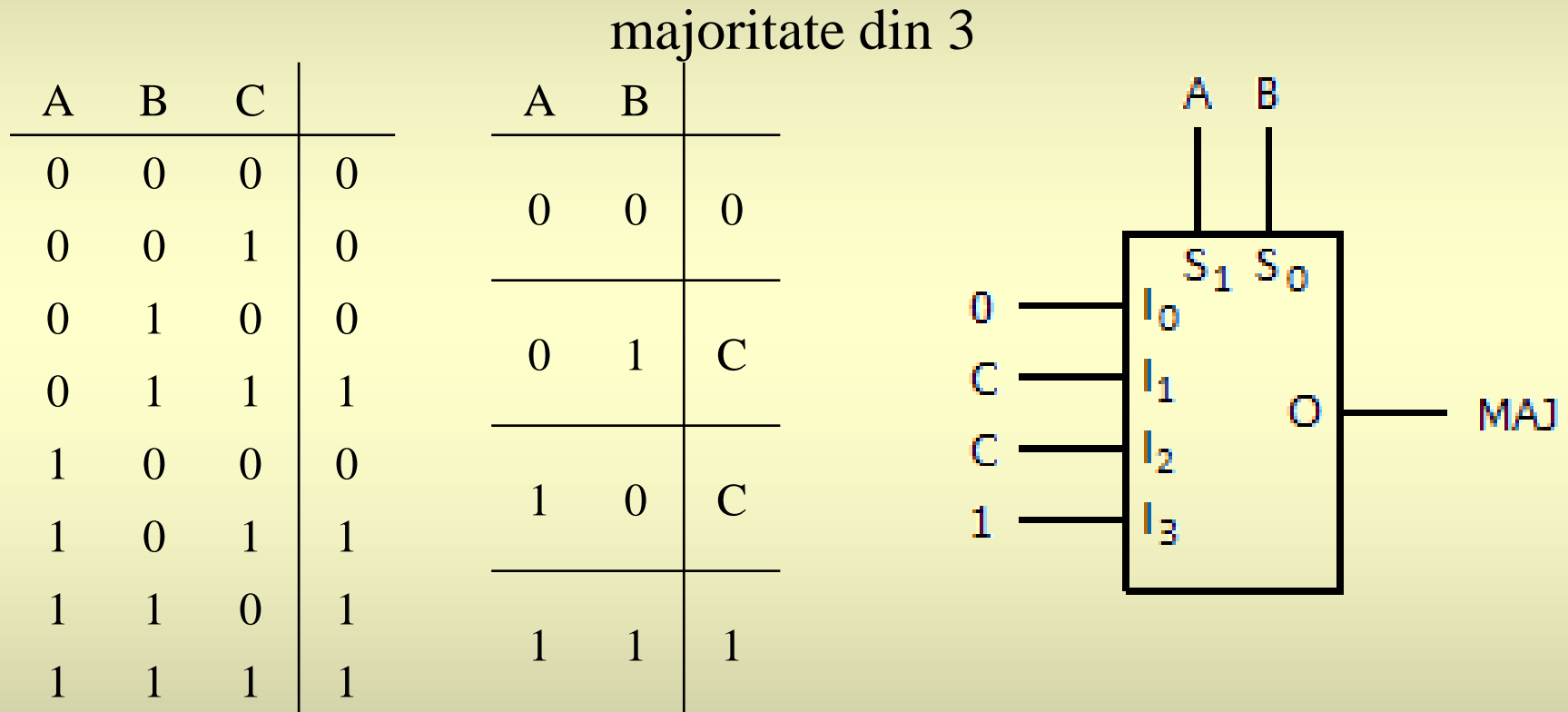
# Implementare funcții booleene

- intrările de selecție formează un număr
- care reprezintă indicele intrării de date care este selectată ca valoare de ieșire
- putem astfel implementa funcții booleene cu ajutorul multiplexorului
  - intrări de date - ieșirile corespunzătoare liniilor din tabelul de adevăr
  - intrări de selecție - intrările funcției booleene

# Example



# Implementare eficientă - *folding*

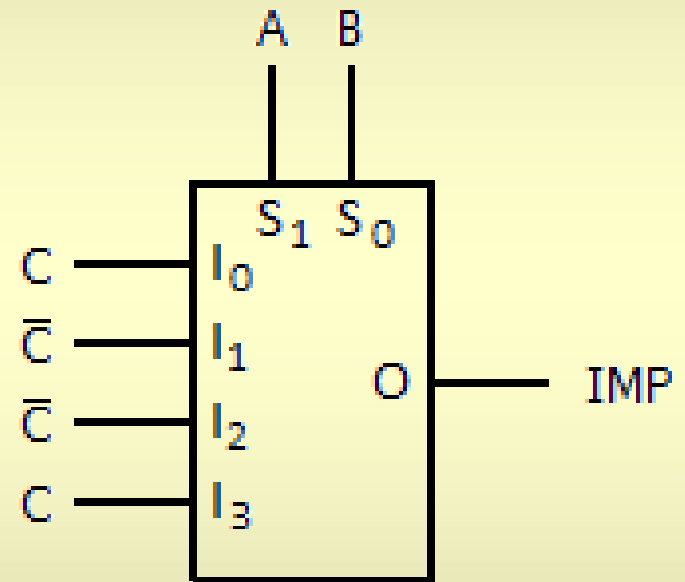


# Implementare eficientă - *folding*

imparitate

A	B	C	
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

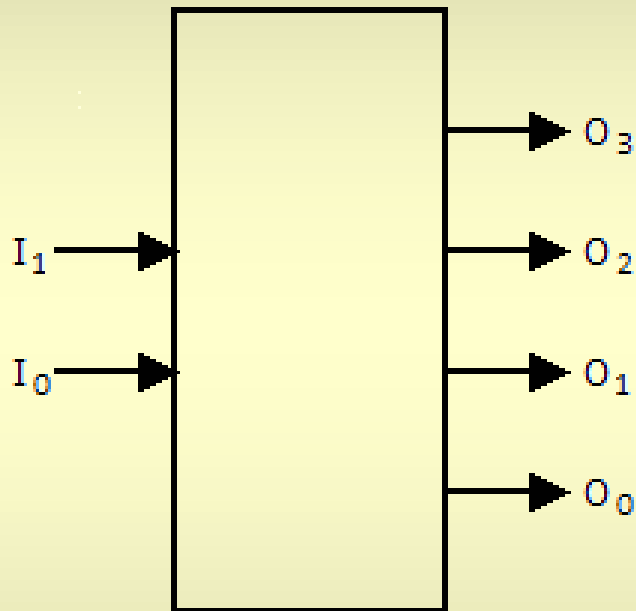
A	B	
0	0	C
0	1	$\bar{C}$
1	0	$\bar{C}$
1	1	C



# Decodorul

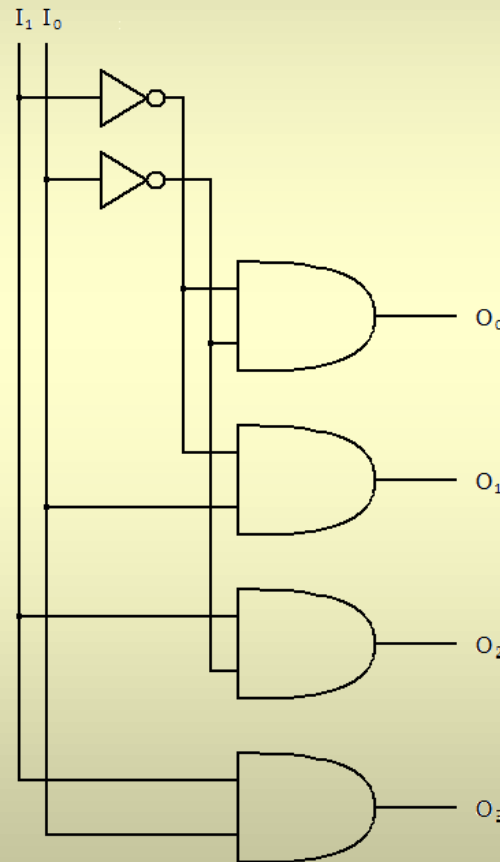
- $n$  intrări
- $2^n$  ieșiri
- în fiecare moment, exact una din ieșiri este activată
  - cea al cărei indice este egal cu numărul format de intrări
  - fiecare ieșire corespunde unui termen FND scris cu variabilele de intrare

## Decodor $n=2$



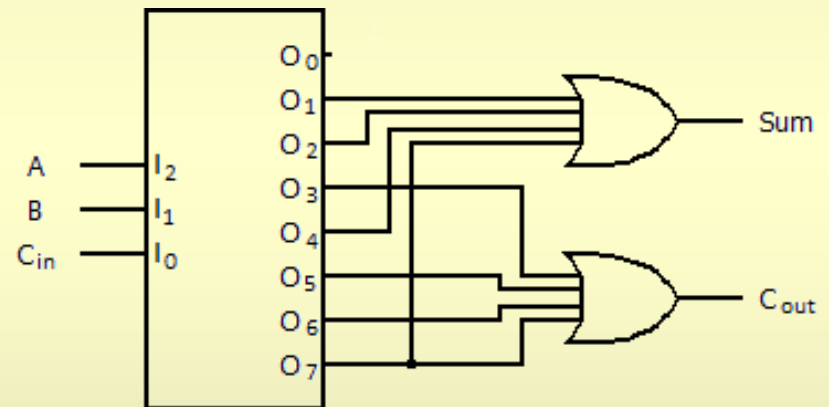
$I_1$	$I_0$	$O_3$	$O_2$	$O_1$	$O_0$
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

# Diagrama logică ( $n=2$ )



# Adunare - implementare cu decodor

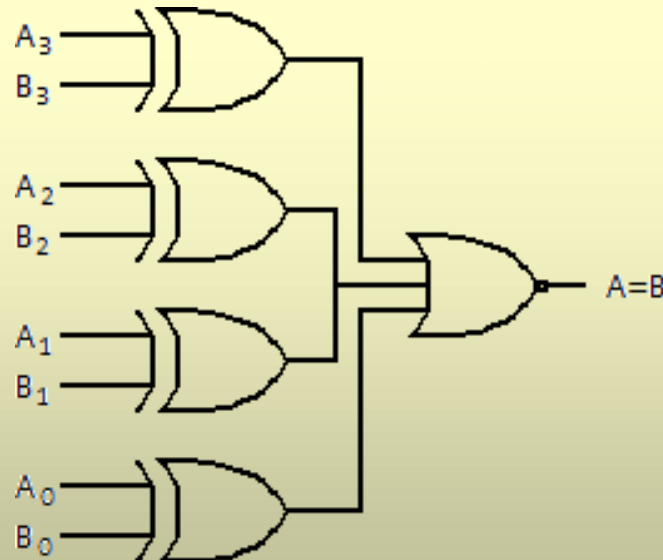
A	B	C <sub>in</sub>	Sum	C <sub>out</sub>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1





# Comparatorul

- operatorii de comparare:  $=$  ,  $>$  ,  $<$  ,  $\geq$  ,  $\leq$ 
  - exemplu de implementare: egalitate pe 4 biți
  - temă: comparator complet ( $<$ ,  $=$ ,  $>$ )

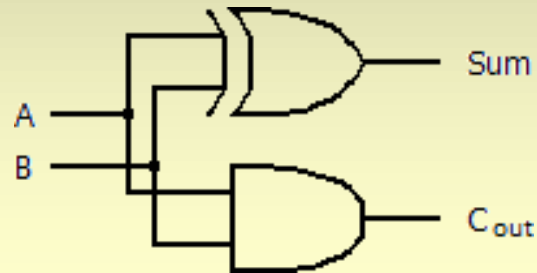


# Sumatorul

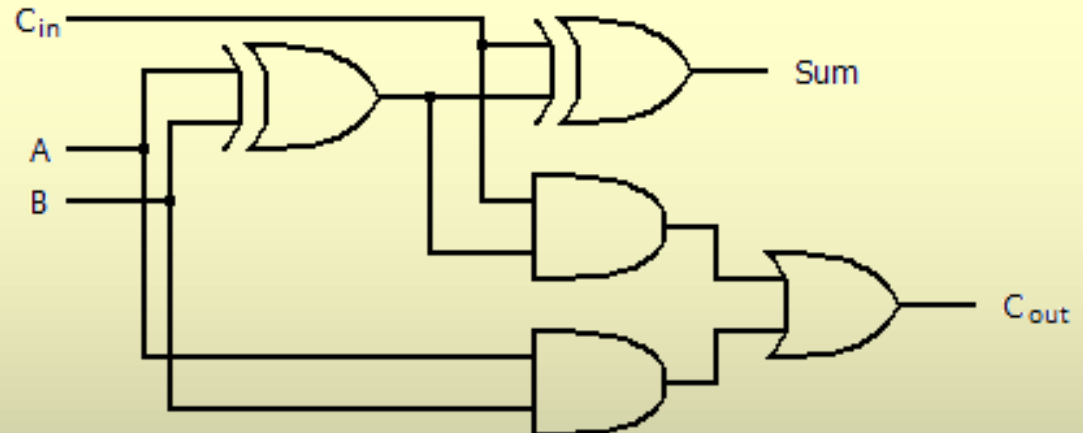
- semi-sumatorul (*half-adder*)
  - adună cei doi biți de intrare
  - ieșire: un bit sumă și un bit transport
  - nu poate fi extins pentru mai multe cifre
- sumatorul complet (*full adder*)
  - adună cei trei biți de intrare (inclusiv transport)
  - aceeași ieșire: un bit sumă și un bit transport
  - poate fi extins pentru mai multe cifre

# Diagrame logice

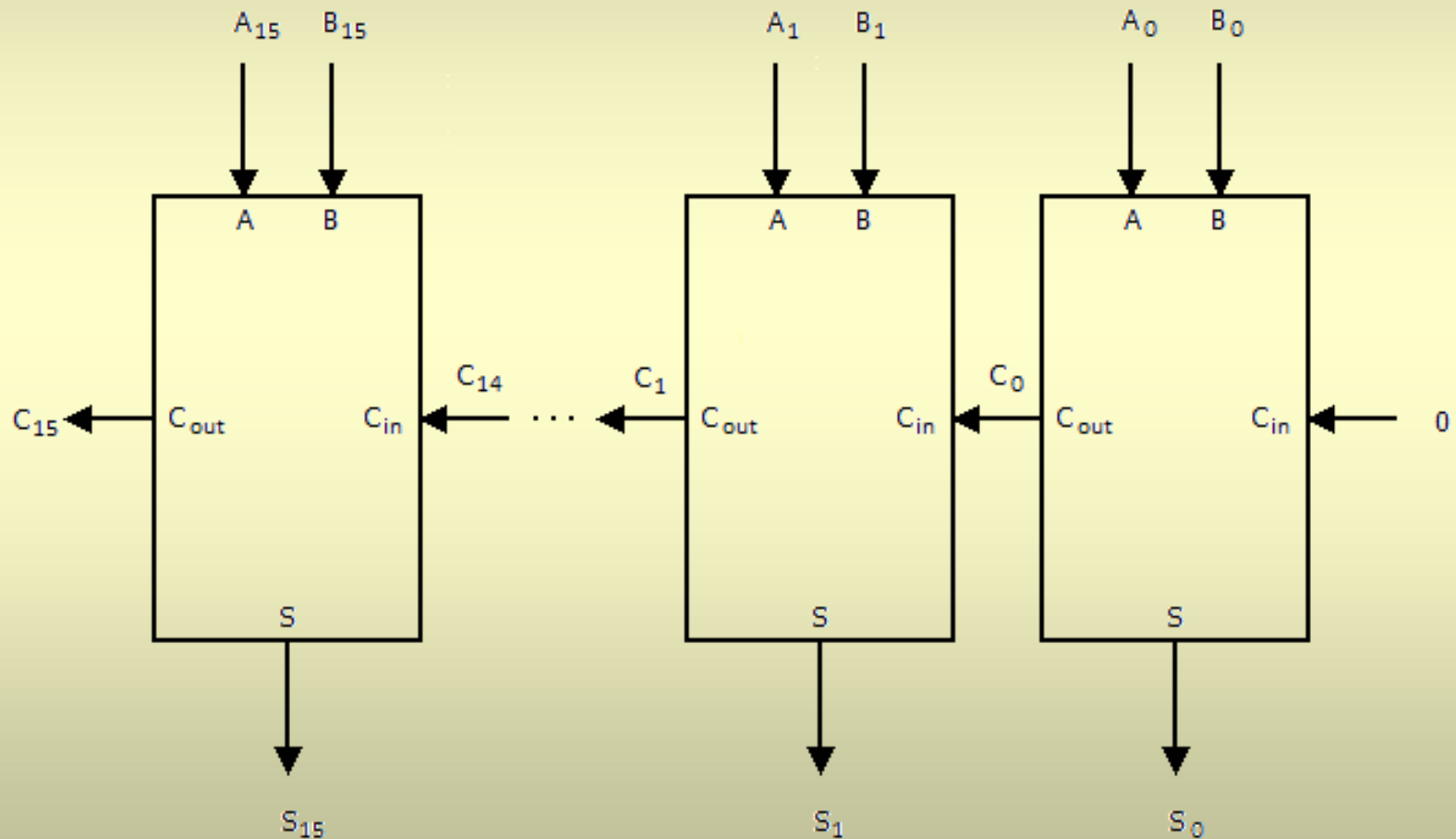
A	B	Sum	C <sub>out</sub>
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



A	B	C <sub>in</sub>	Sum	C <sub>out</sub>
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



# Sumator serial (16 biți)



## Sumatoare seriale

- această variantă este sumatorul cu propagarea transportului
- avantaj: același circuit (simplu), repetat
- dezavantaj: viteza
  - la fiecare rang, trebuie așteptat rezultatul de pe rangul anterior
  - deci întârzierea este proporțională cu numărul de biți

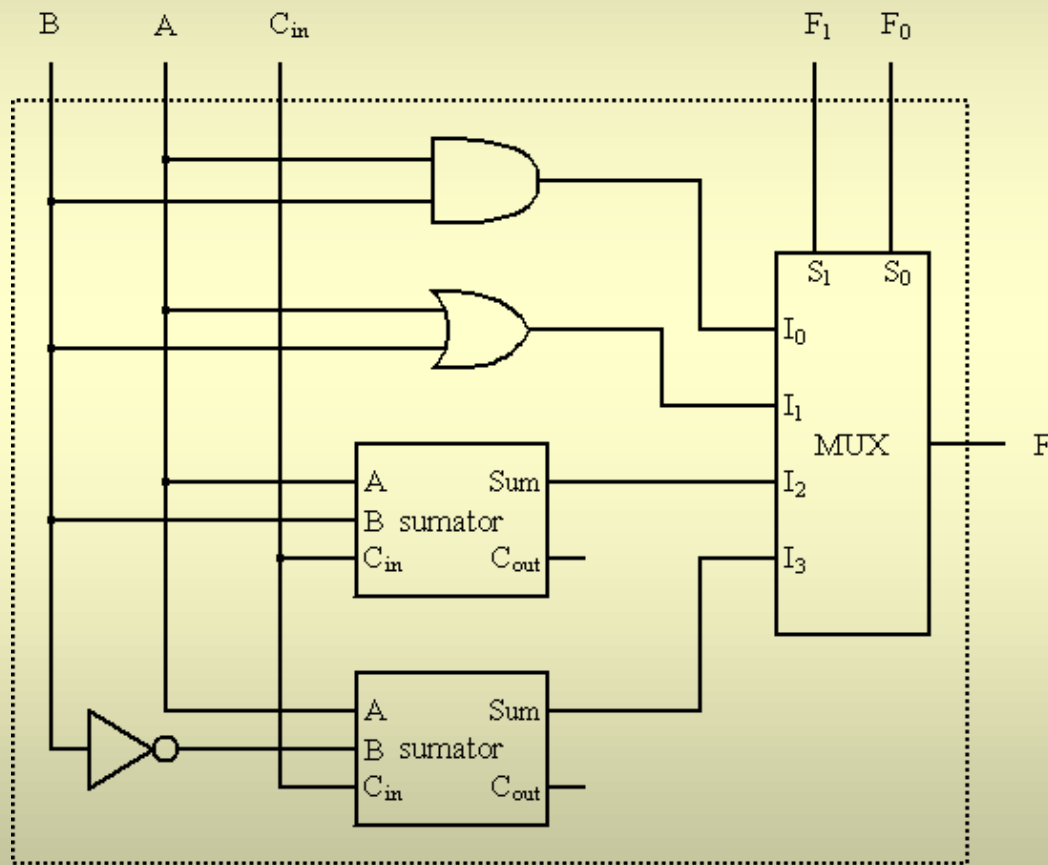
# Accelerarea adunării (1)

- sumator cu anticiparea transportului
  - *carry lookahead adder*
  - transportul de intrare - generat independent pentru fiecare rang
    - $C_0 = A_0 B_0$
    - $C_1 = A_0 B_0 A_1 + A_0 B_0 B_1 + A_1 B_1$
    - ...
    - $C_i = G_i + P_i C_{i-1} = A_i B_i + (A_i + B_i) \cdot C_{i-1}$
    - ...

## Accelerarea adunării (2)

- sumatorul cu anticiparea transportului
  - avantaj - viteza
    - elimină întârzierea datorată propagării transportului
  - dezavantaj - circuite suplimentare, complexe
  - de obicei - combinație anticipare-propagare
- sumator cu selecția transportului
  - la fiecare rang se calculează suma pentru  $C_{in}=0$  și  $C_{in}=1$ , apoi se selectează cea corectă

# Unitate aritmetică și logică elementară (1 bit)

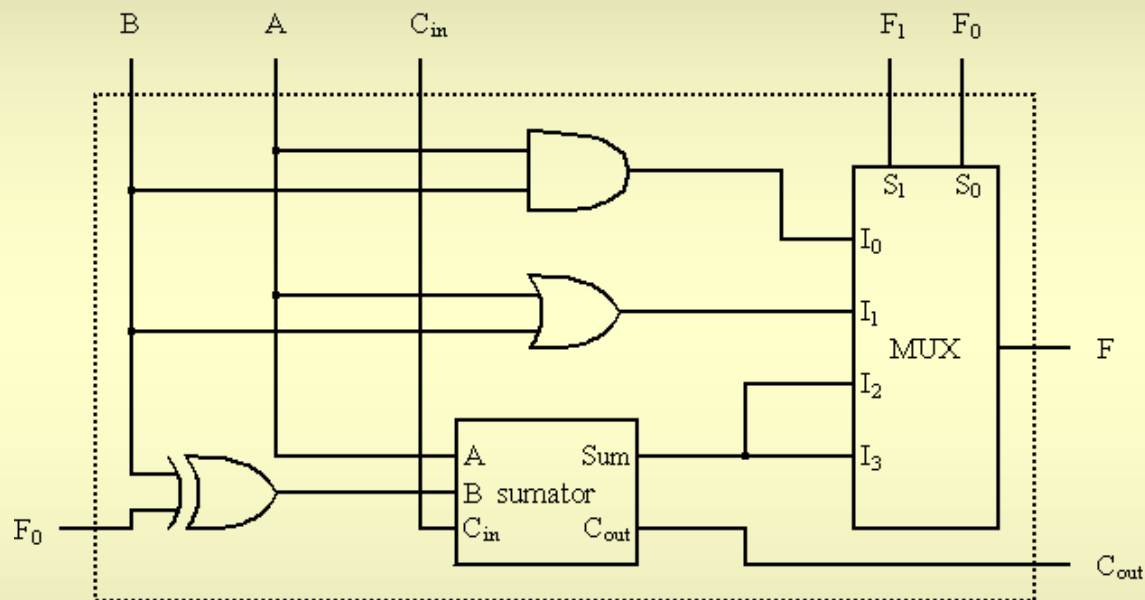


$F_1$	$F_0$	F
0	0	A and B
0	1	A or B
1	0	A+B
1	1	A-B

$F_1, F_0$  - semnale  
de control



# Variantă îmbunătățită



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

$F_1, F_0$  - semnale de control

# Unitate aritmetică și logică pe 16 biți

