

II.5. MINIMIZAREA
FUNCTIILOR BOOLEENE
PRIN METODA
DIAGRAMELOR KARNAUGH

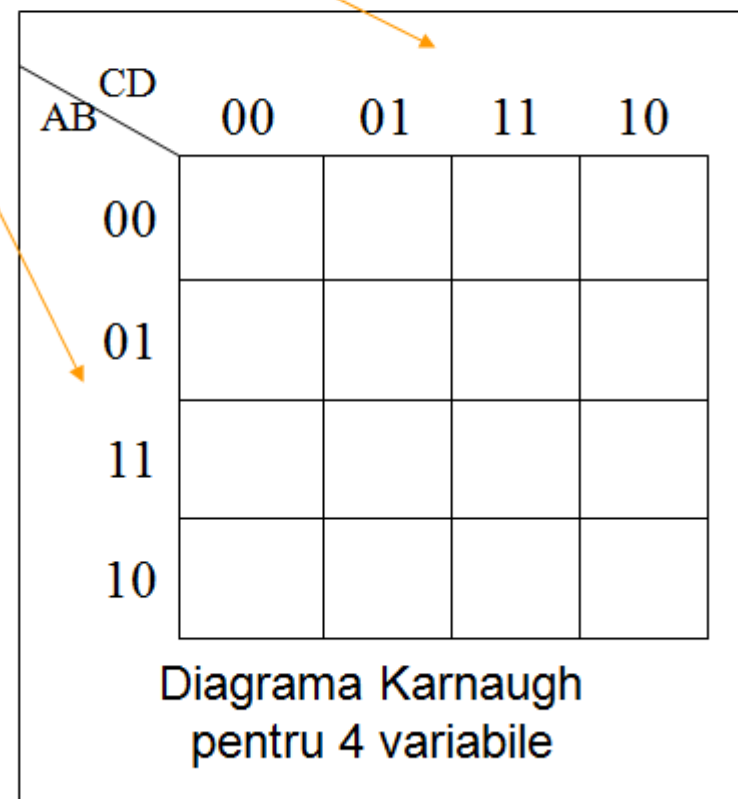
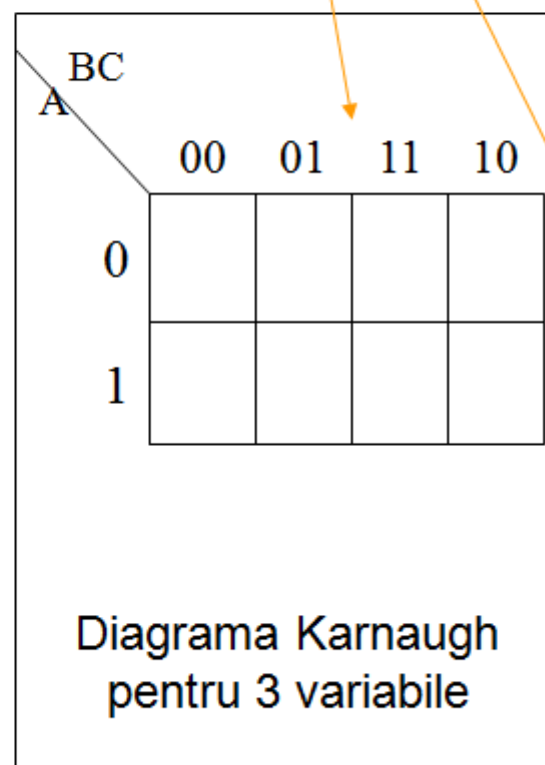
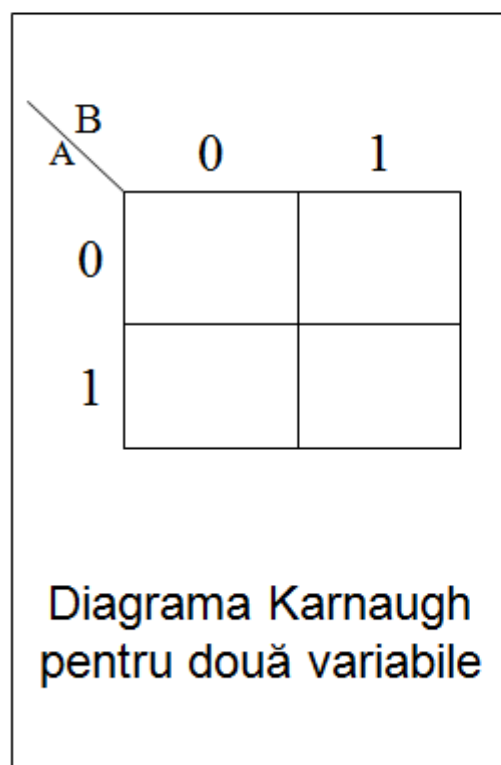
Structura unei diagrame Karnaugh

pentru **n** variabile

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

Metoda Karnaugh

Ordinea codului
Grey



Etichete: codul Grey

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

Adiacențe în diagrame Karnaugh

- Două poziții sunt adiacente dacă etichetele corespunzătoare diferă pe un singur bit
- Generalizează "vecinătatea" intuitivă
- 4 variabile: cele patru colțuri sunt adiacente!
- Pentru o funcție de n variabile, o locație are n locații adiacente
 - < 5 variabile – vizual
 - 5 sau mai multe: și alte adiacențe decât cele vizibile

Pașii minimizării Karnaugh

1. Se trec în locațiile corespunzătoare (conform etichetelor) valorile de 1 ale funcției
2. Se caută blocuri conținând numai valori 1, astfel încât:
 - fiecare valoare 1 să fie inclusă în cel puțin un bloc
 - blocurile să fie cât mai mari și mai puține
 - un bloc să conțină un număr de locații egal cu o putere a lui 2
 - eventual puterea 0
 - dacă blocul conține 2^k locații, atunci pentru fiecare locație blocul să conțină exact **k** locații adiacente cu ea

Pașii minimizării Karnaugh

3. Se scrie expresia minimizată a funcției astfel:
 - ◆ fiecărui bloc cu 2^k locații 1 îi corespunde un termen conținând **n-k** variabile legate prin conjuncție
 - ◆ în termen apar acele variabile ale căror etichete sunt constante pentru toate locațiile din bloc
 - ◆ o variabilă apare negată dacă eticheta sa constantă este 0 și nenegată altfel
 - ◆ termenii astfel obținuți (după considerarea tuturor blocurilor) sunt legați prin disjuncție

Example: "majoritatea din 3"; imparitate

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

Diagram illustrating the truth table for the 3-input majority function (majoritatea din 3). The table has columns for BC (00, 01, 11, 10) and rows for A (0, 1). The output is 1 for the following combinations: (A=0, B=1, C=1), (A=1, B=0, C=1), (A=1, B=1, C=0), and (A=1, B=1, C=1). The output is 0 for all other combinations. The function is labeled "Funcția majoritate din 3".

Funcția *majoritate din 3*

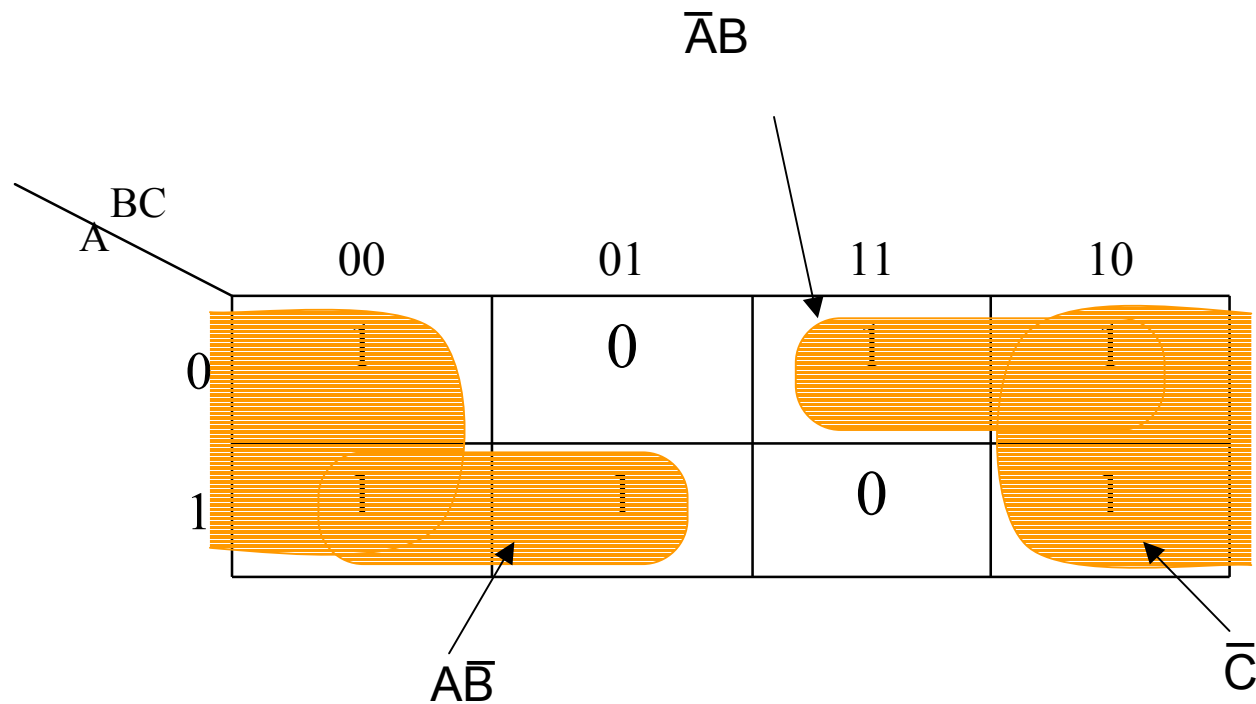
		$\bar{A}\bar{B}C$		$\bar{A}B\bar{C}$	
		00	01	11	10
A	0	0	1	0	1
	1	1	0	1	0

Diagram illustrating the truth table for the 3-input parity function (imparitate). The table has columns for BC (00, 01, 11, 10) and rows for A (0, 1). The output is 1 for the combinations: (A=0, B=0, C=1), (A=0, B=1, C=0), (A=1, B=0, C=0), and (A=1, B=1, C=1). The output is 0 for all other combinations. The function is labeled "Funcția imparitate".

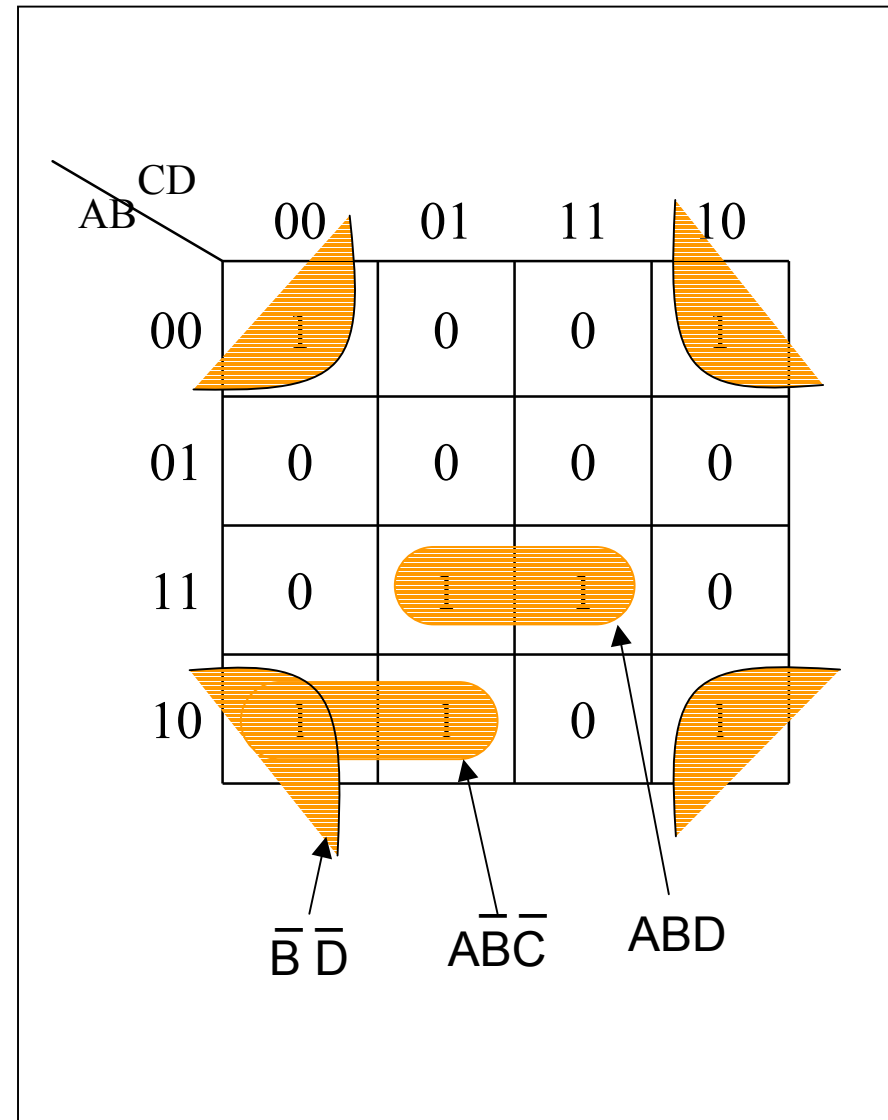
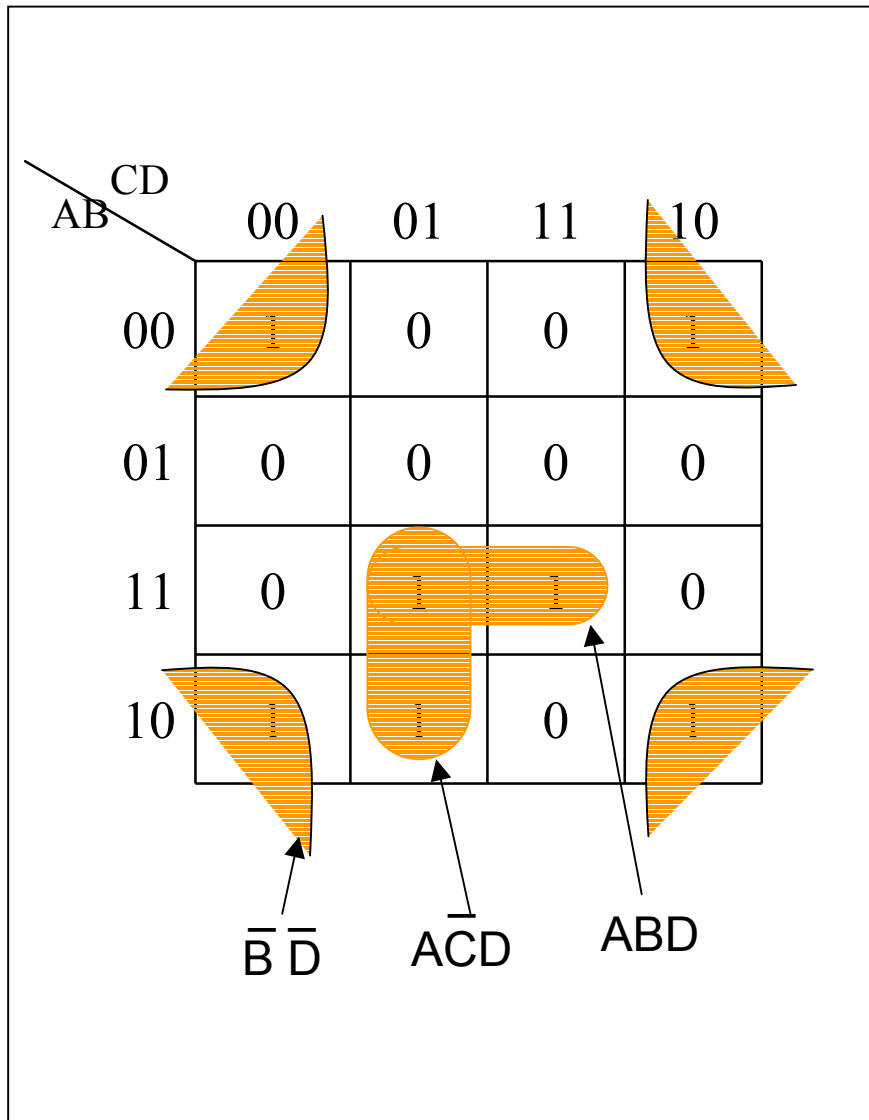
Funcția *imparitate*

Adiacența liniilor/**coloanelor** extreme

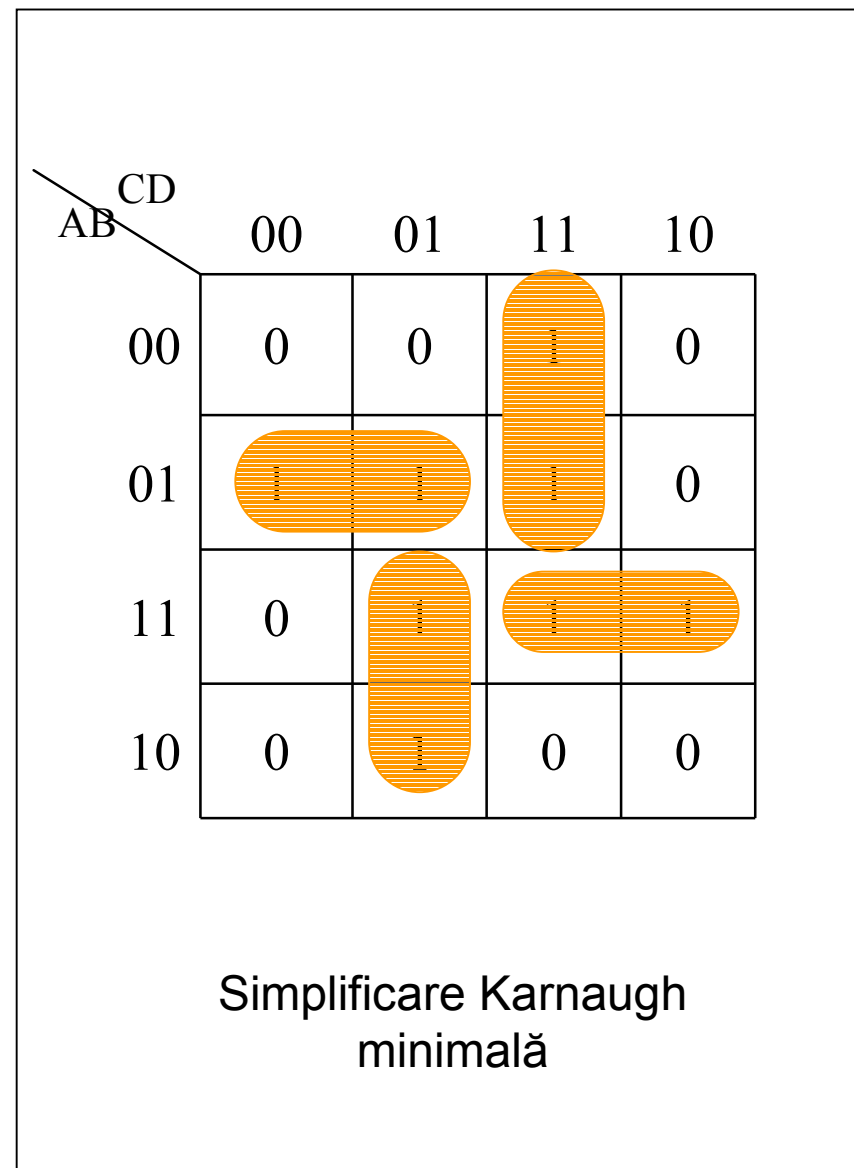
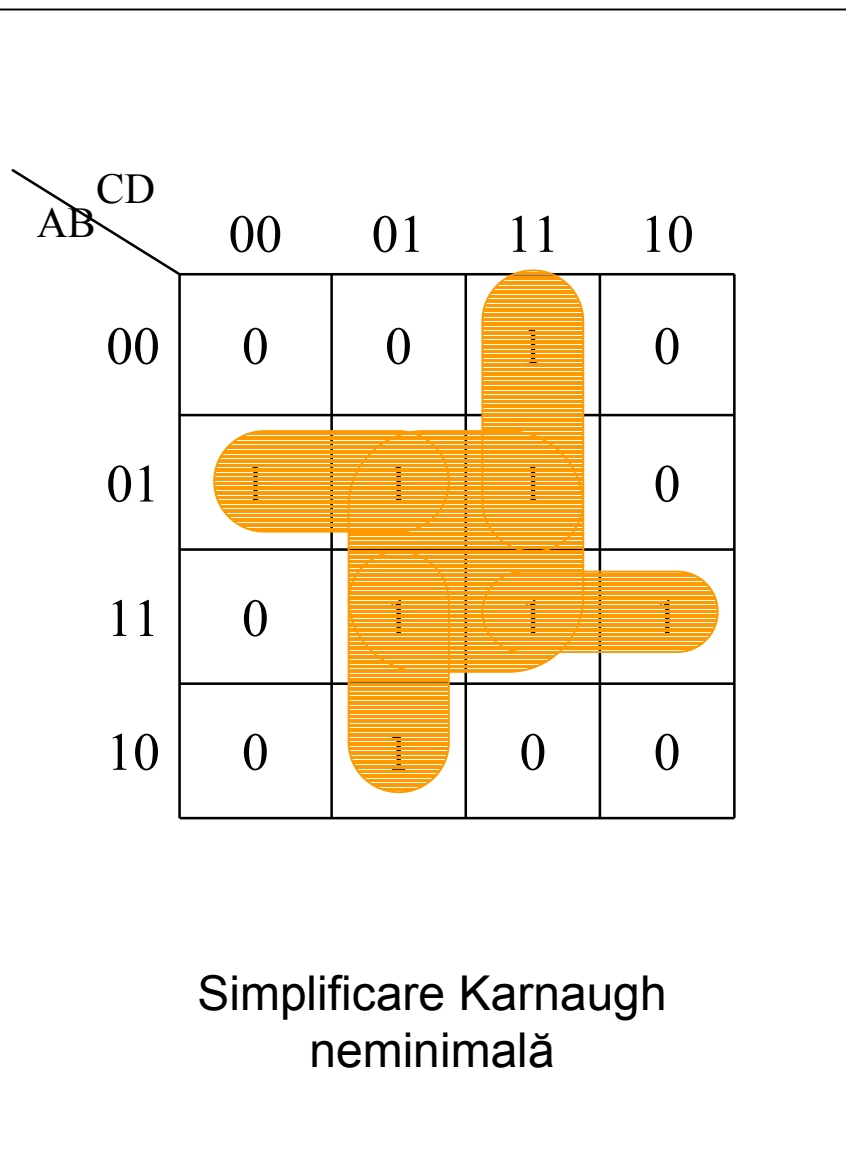
$f(A,B,C)=\Sigma(0,2,3,4,5,6)$



Expresia depinde de grupare



Evitarea redundanțelor

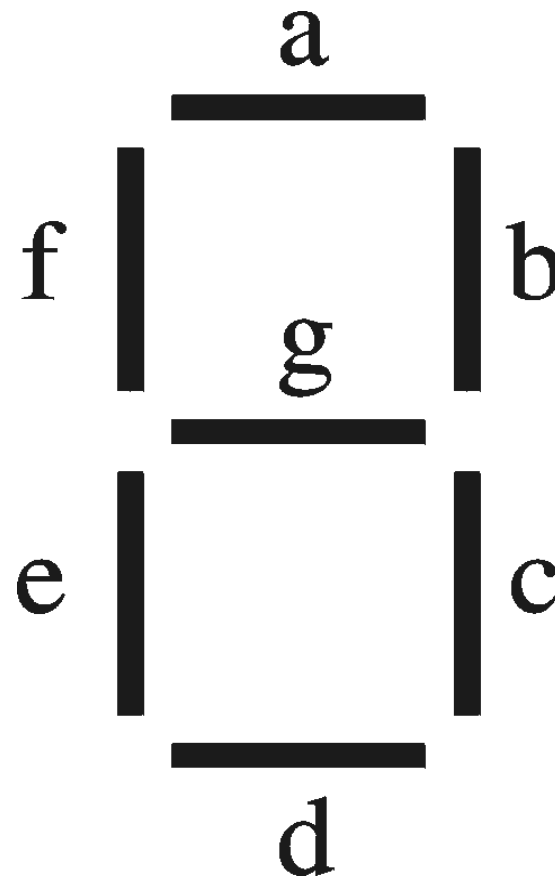


Combinații imposibile de valori

- variabilele nu vor avea niciodată acele combinații de valori
- se poate deci considera **restricția** funcției booleene doar la subdomeniul de definiție al combinațiilor permise
 - doar aceasta va fi "vizibilă" în funcționarea circuitului
- se consideră cea mai convenabilă – din punctul de vedere al minimizării – extensie la combinațiile imposibile
 - se consideră valoarea 0 sau 1, după cum convine

Exemplu – afișarea cifrelor zecimale

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



Funcția booleană atașată 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	*

Combinațiile imposibile pot simplifica expresia

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	0	0	0	0
10	1	1	0	0

Simplificare “funcționare de
siguranță”

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	*	*	*	*
10	*	*	*	*

Simplificare exploitând combinațiile
imposibile

Temă: comparator pe 2 biți

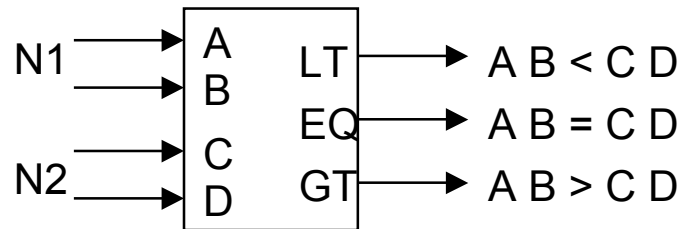


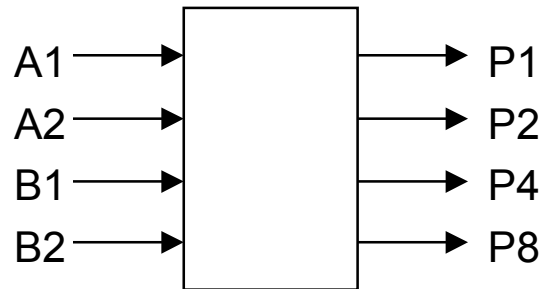
Diagrama bloc

și tabelul de adevăr

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

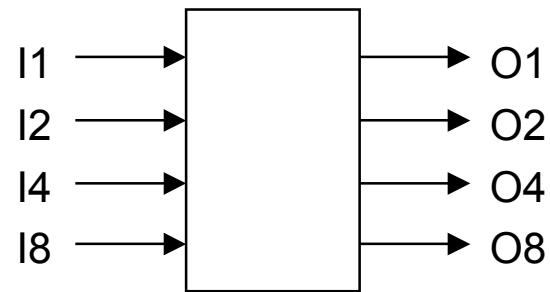
A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Temă: multiplicator pe 2 biți



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 COMBINAȚIONALE

Circuite combinaționale

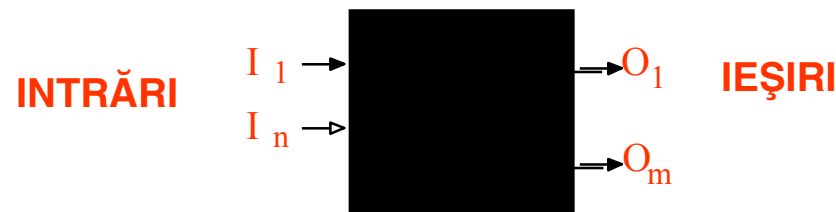


Diagrama bloc a unui circuit combinațional

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

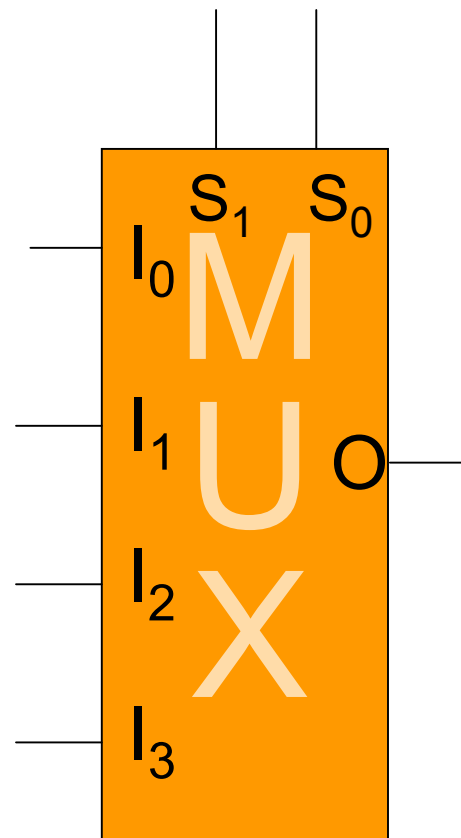
II.6.1. MULTIPLEXORUL

Multiplexorul

- 2^n intrări
- 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
- controlul selectează o valoare de la intrare (bit) care devine valoare de ieșire

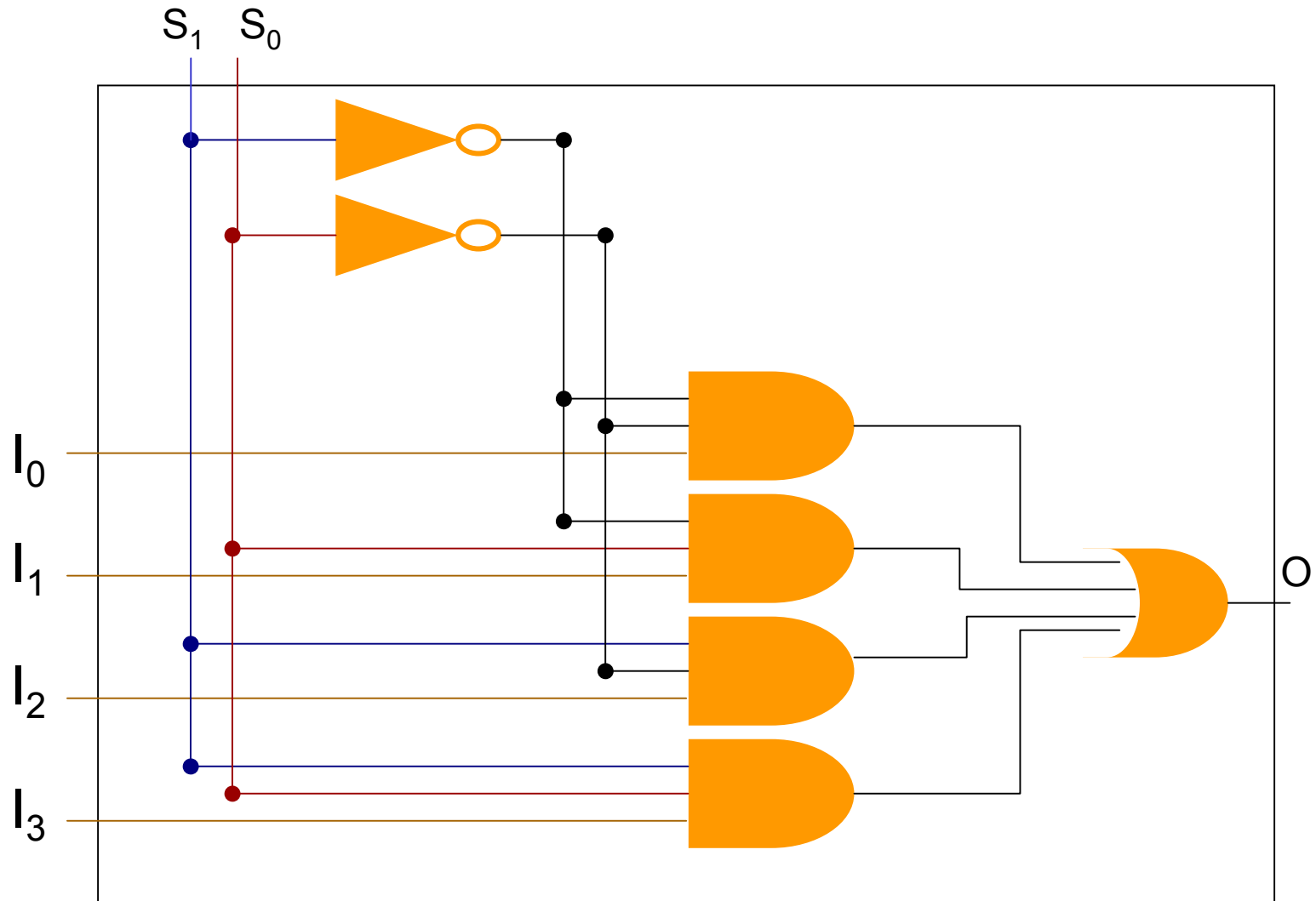
MUX 4 \rightarrow 1

diagrama bloc și tabelul de adevăr



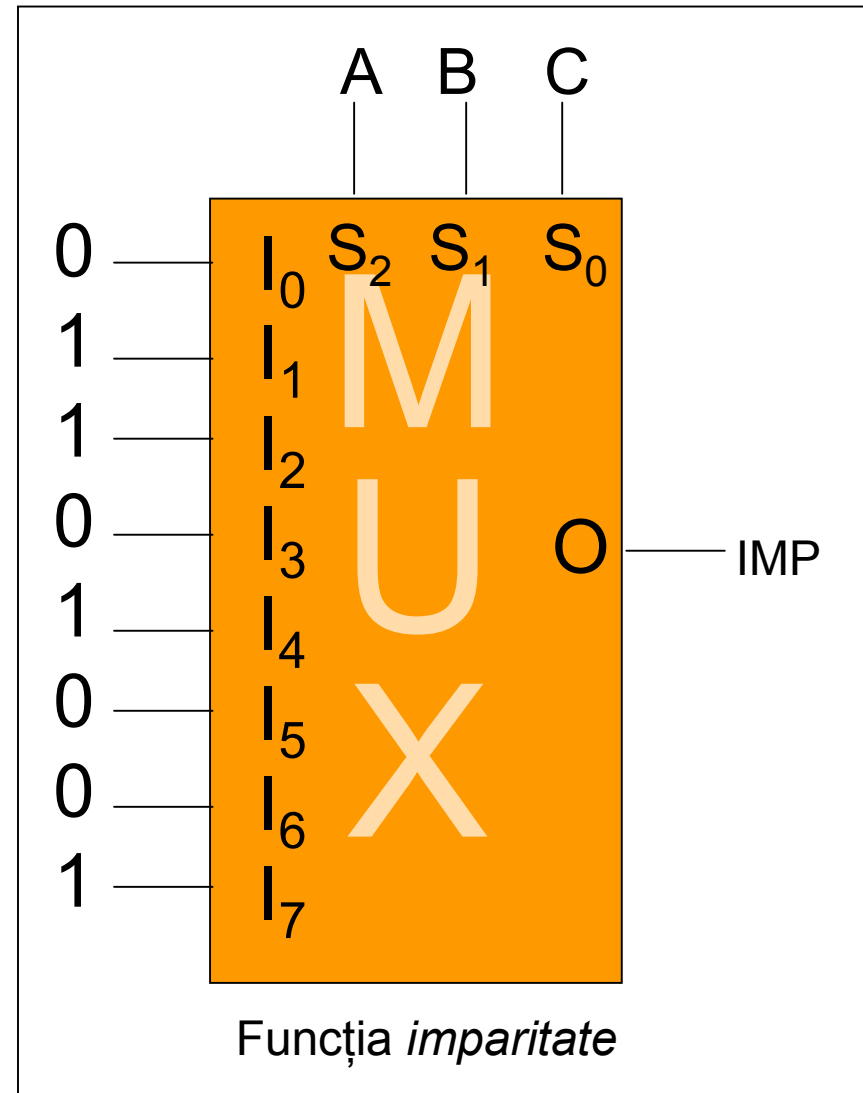
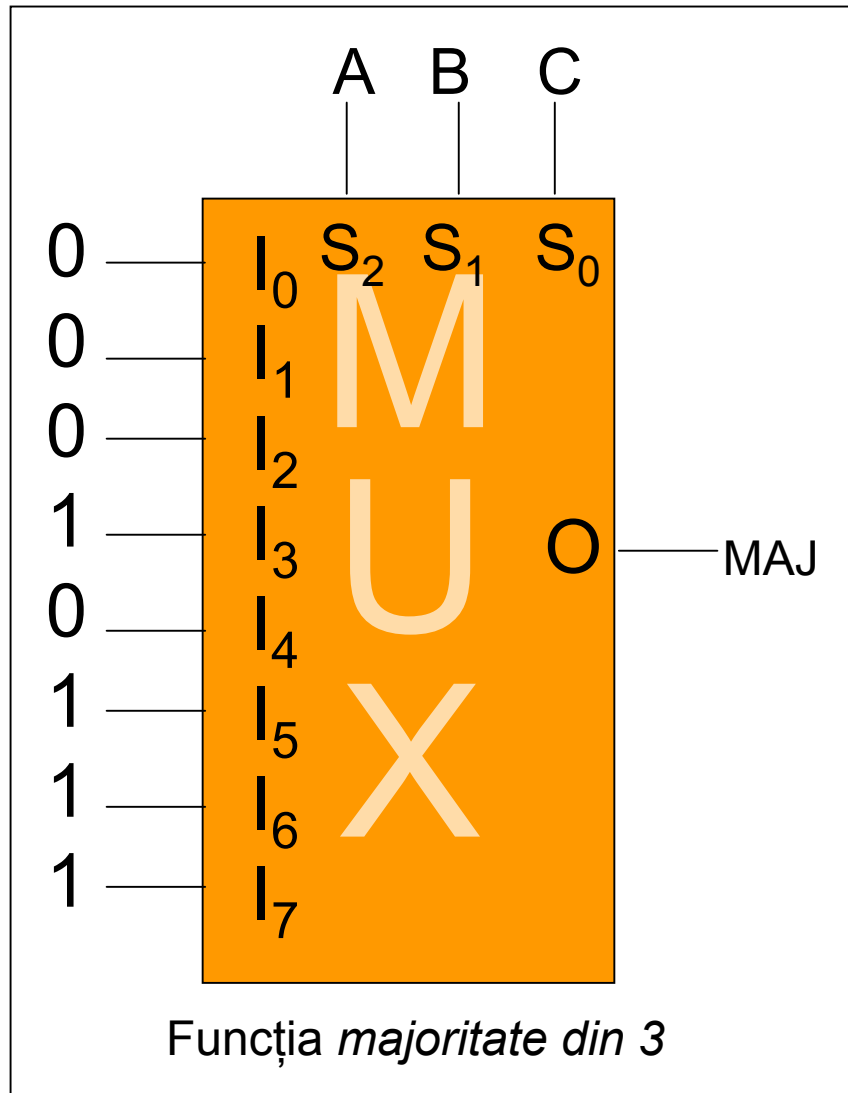
S_1	S_0	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Multiplexorul $4 \rightarrow 1$: diagrama logică



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

Funcții booleene pot fi implementate prin multiplexoare



"Majoritatea din 3": implementare eficientă prin multiplexor

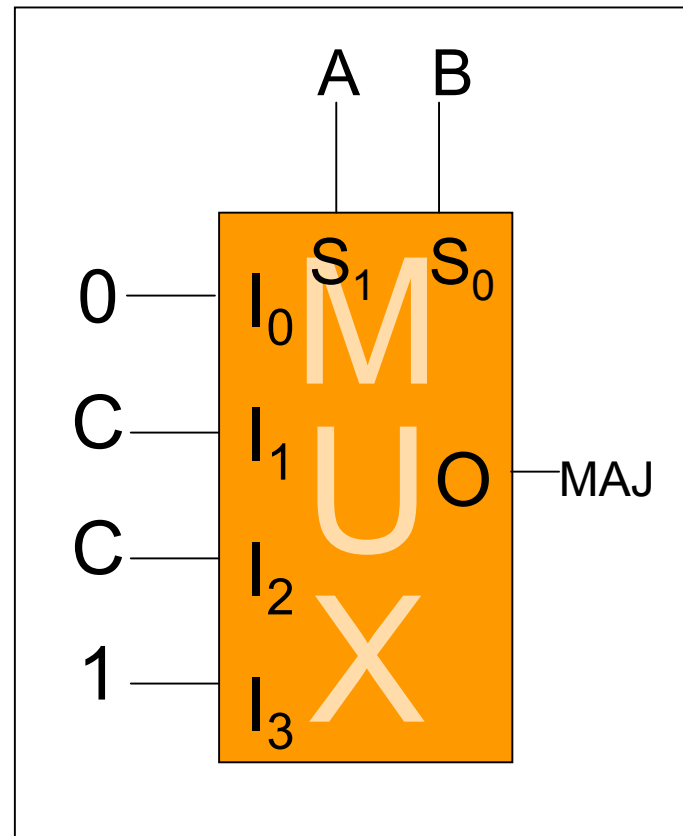
Eficient: *folding*

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 original

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

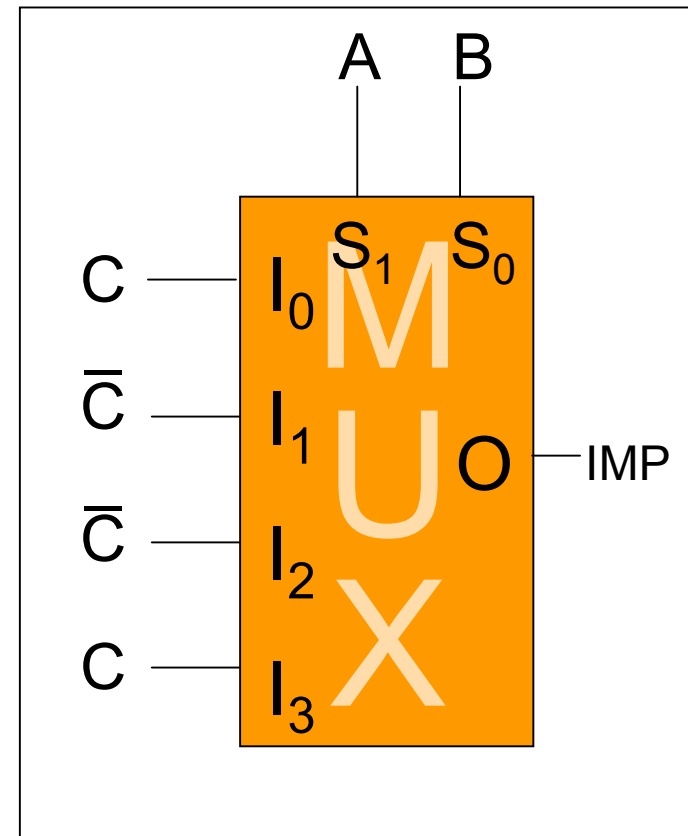
folding

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
original

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

Tabel de adevăr
pentru multiplexor



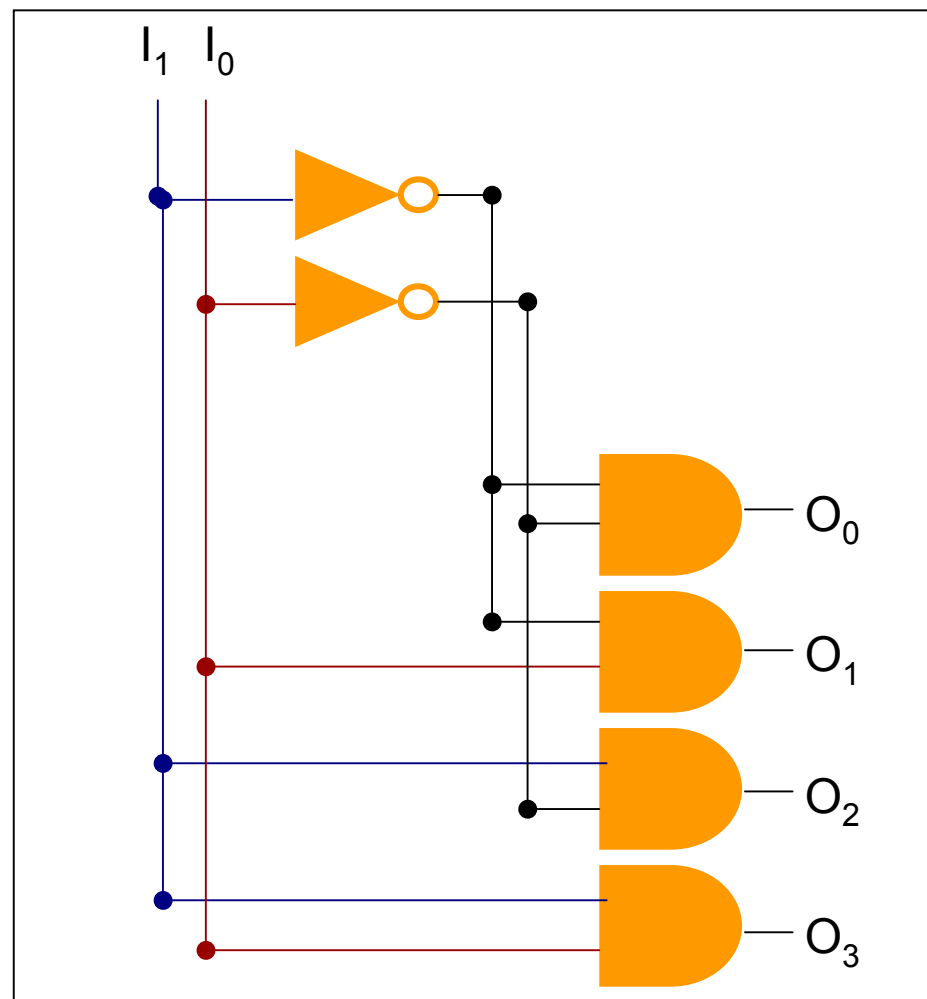
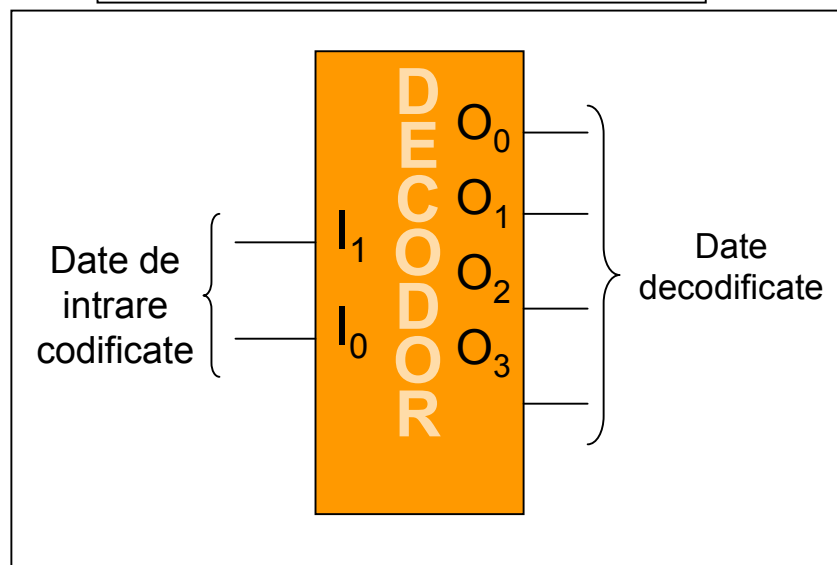
II.6.2. DECODORUL, COMPARATORUL

Decodorul

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

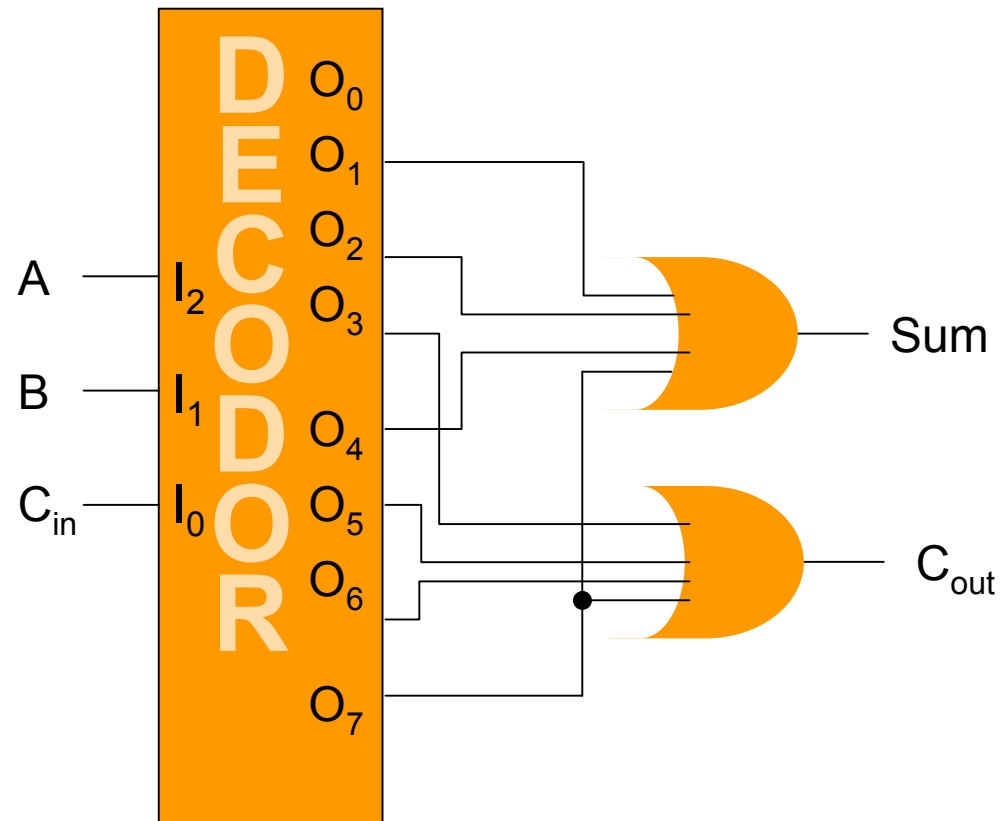
Decodorul: k=2

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



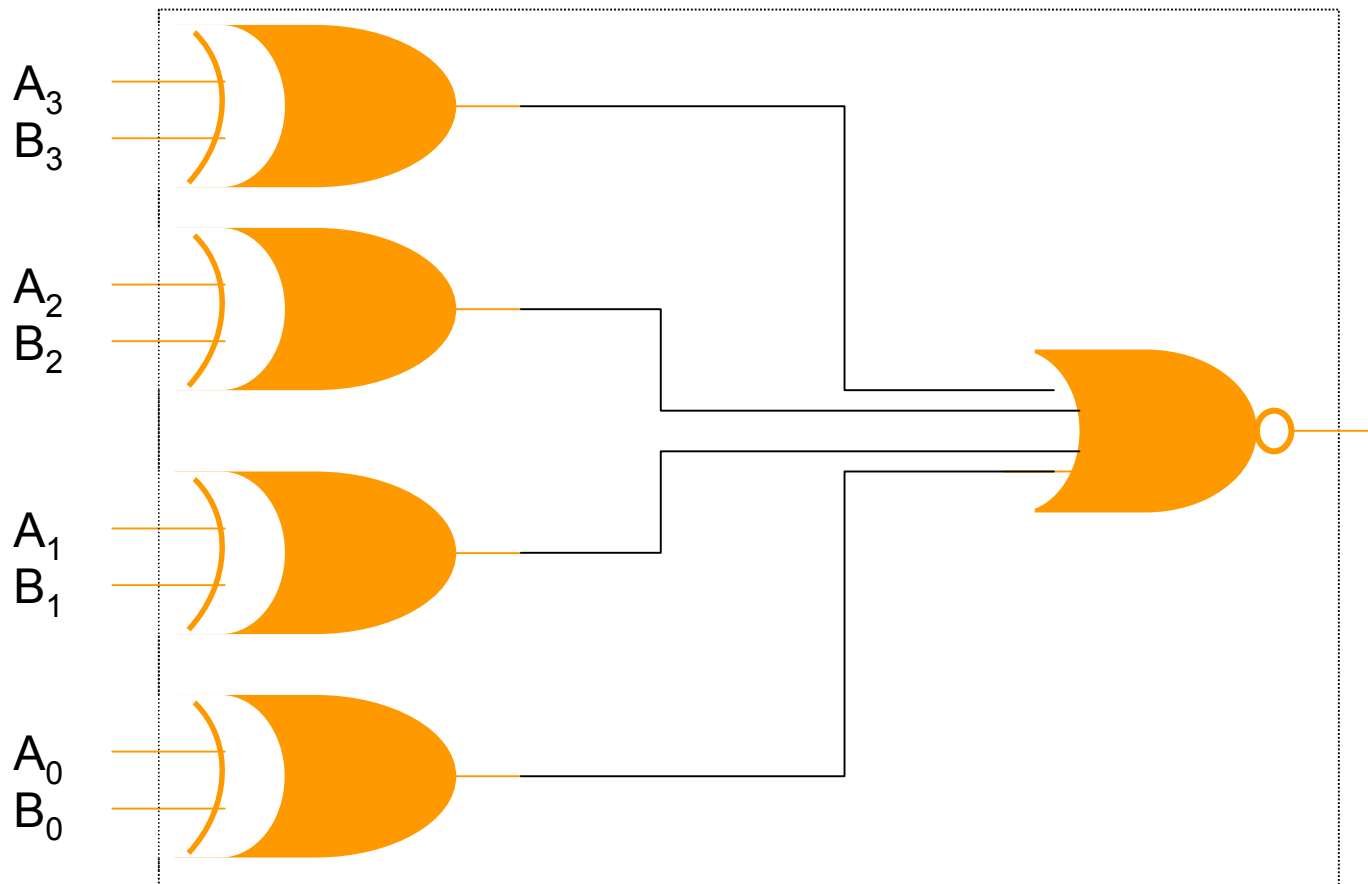
Implementarea adunării prin decodoare

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 ($=$, $>$, $<$, \geq , \leq)
- Exemplu: egalitate pe 4 biți
 - Temă: comparator complet ($<$, $=$, $>$)



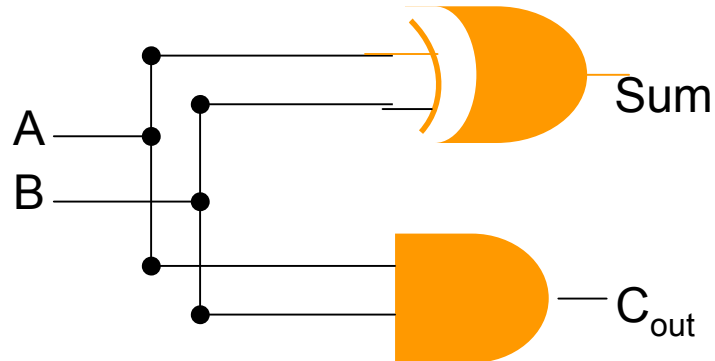
II.6.3. SUMATOARE

Semi-sumatorul și sumatorul complet

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

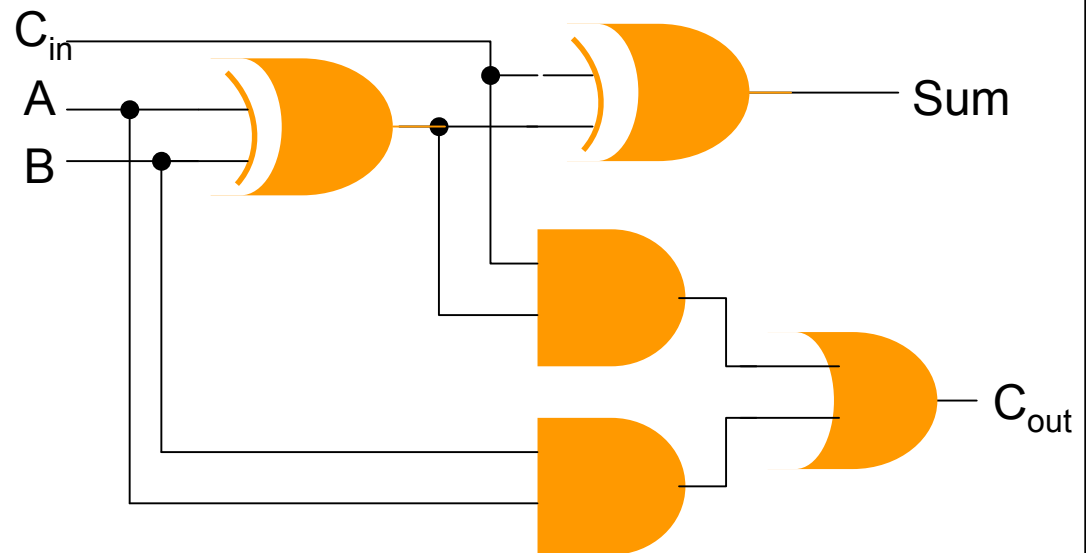
Semi-sumator și sumator: diagrame logice

A	B	Sum	C _{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



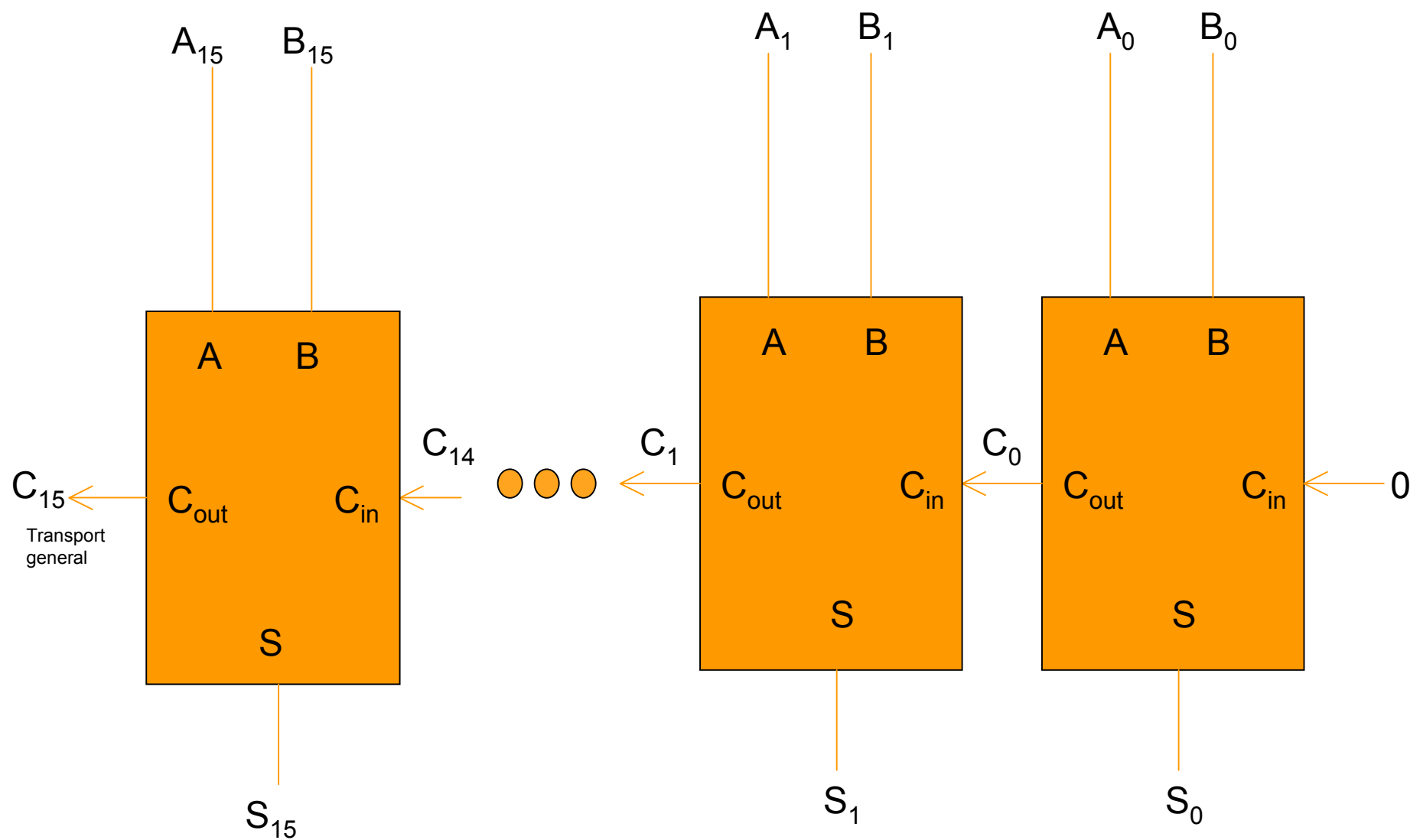
Tabelul de adevăr și diagrama logică ale semi-sumatorului

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 și diagrama logică ale sumatorului complet

Sumatorul serial pe 16 biți (+)



Sumatoare seriale

- Cu propagarea transportului
 - de la un rang la următoarele
- Și 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 - proporțională cu numărul de biți
- Cazul cel mai relevant:
1111111 + 0000001

Accelerarea adunării

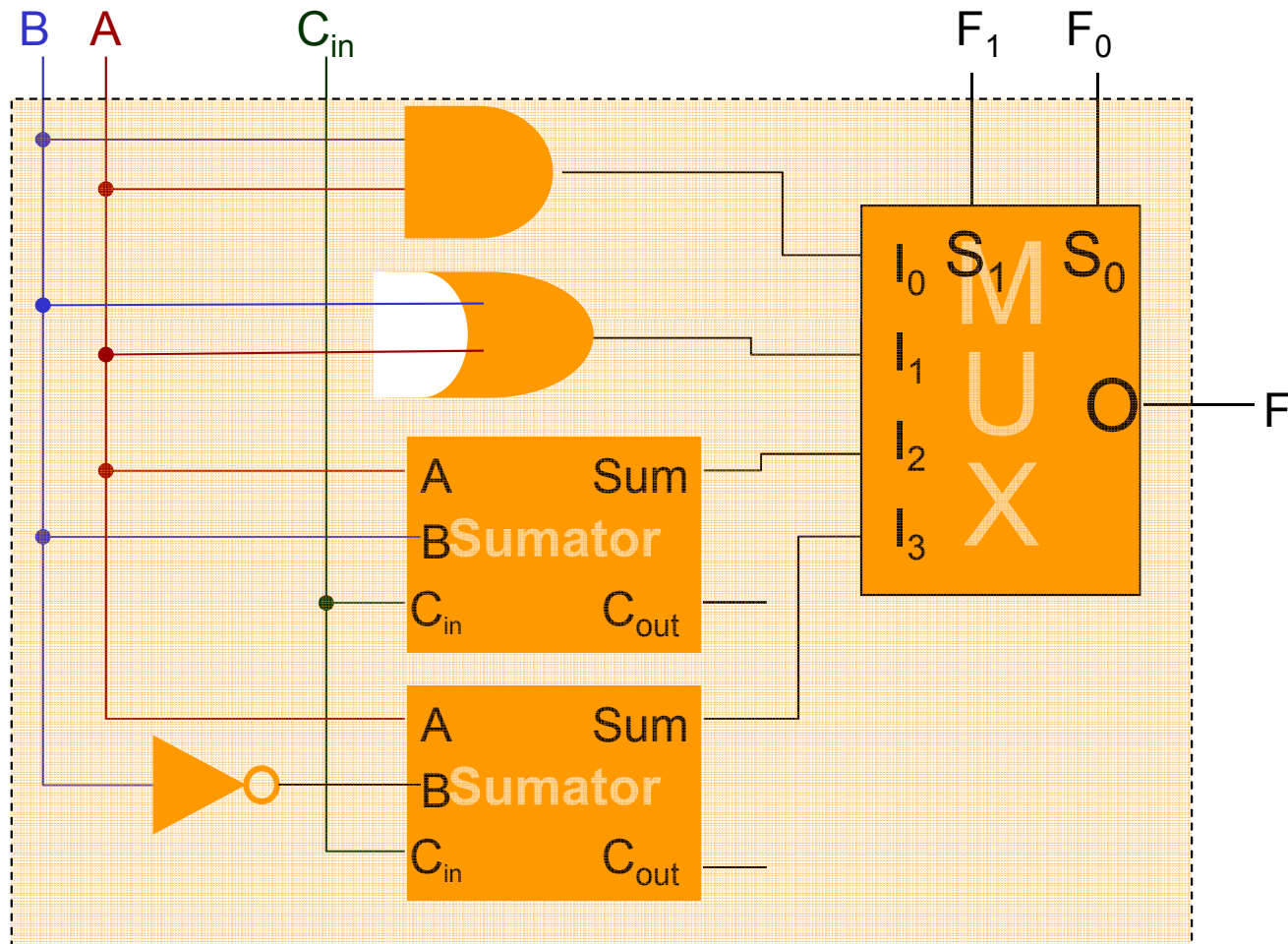
- Sumatoare cu anticiparea transportului
 - *Carry lookahead adders*
 - Elimină întârzierea datorată propagării transportului
 - Transportul-intrare (carry-in) se generează independent pentru fiecare rang
 - $C_0 = A_0 B_0$
 - $C_1 = A_0 B_0 A_1 + A_0 B_0 B_1 + A_1 B_1$
 - ...
 - $C_i = G_i + P_i \cdot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

II.6.4. O UNITATE ARITMETICĂ ȘI LOGICĂ ELEMENTARĂ

Unitate Aritmetică și Logică (1 bit):

AND, OR, +, -

Proiectare inițială



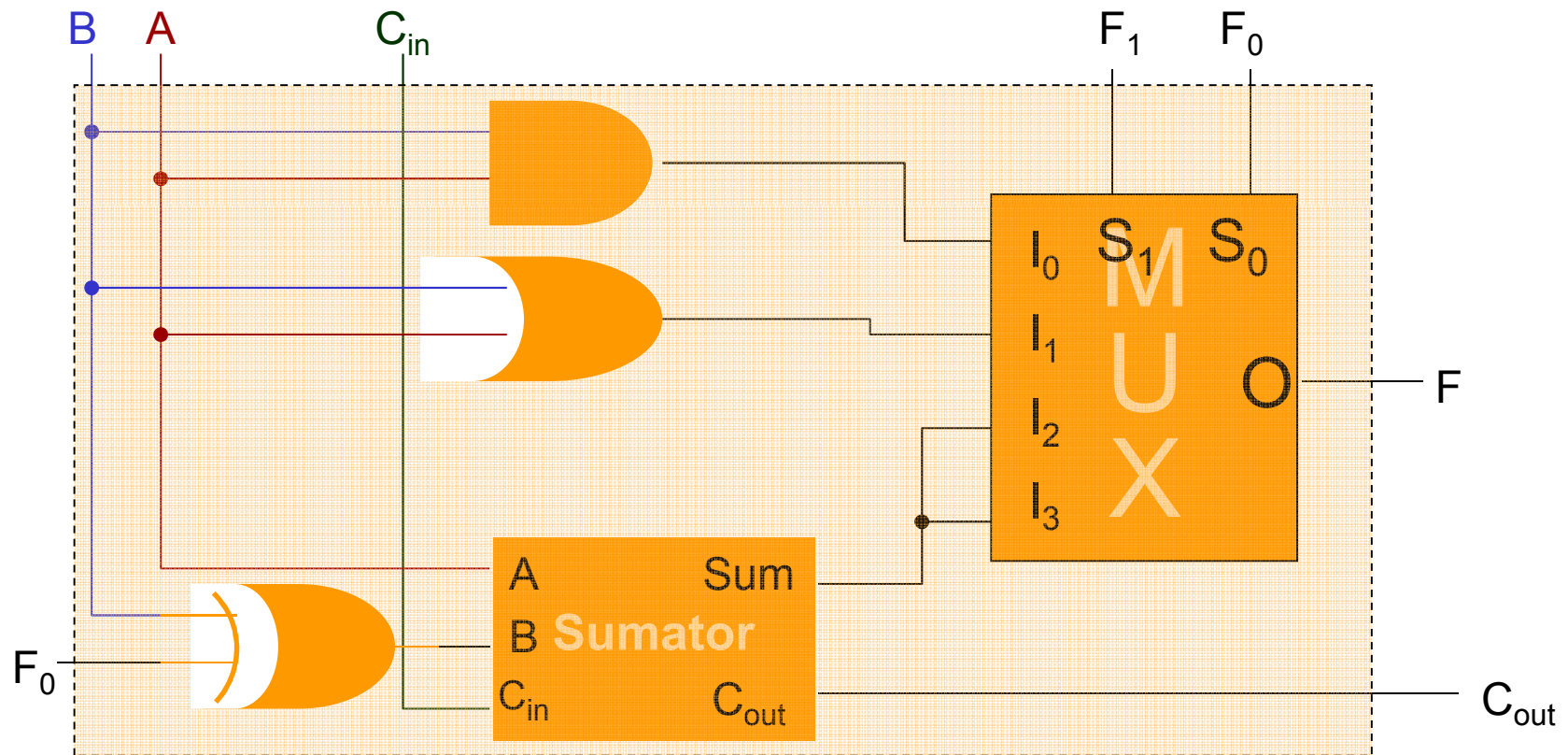
F_1	F_0	F
0	0	A and B
0	1	A or B
1	0	A+B
1	1	A-B

Semnale de control: F_0 F_1

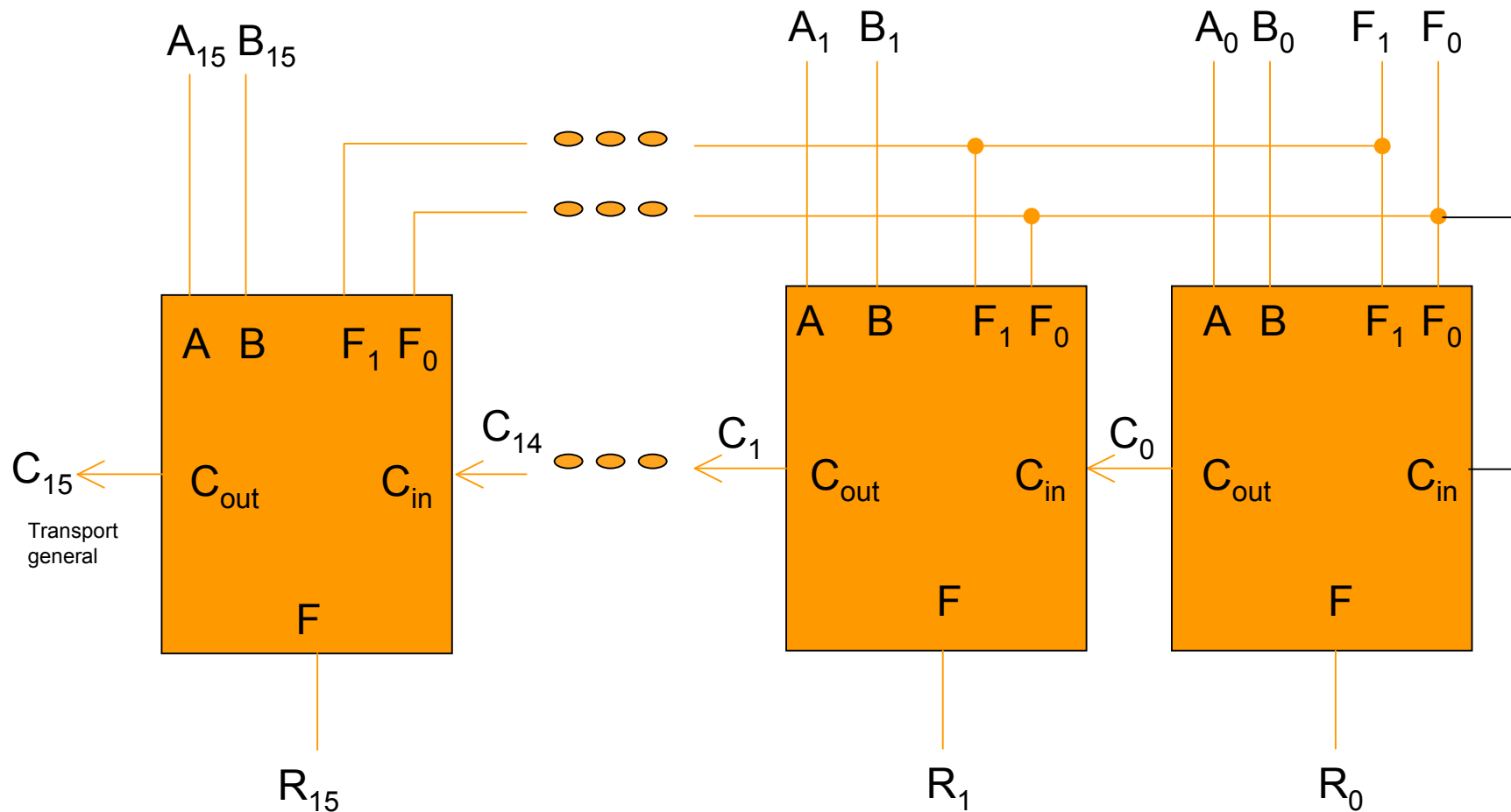
Unitate Aritmetică și Logică (1 bit):

AND, OR, +, -

Proiectare îmbunătățită



Unitate Aritmetică și Logică (16 biți): AND, OR, +, -



Capitolul al III-lea

Circuite secvențiale

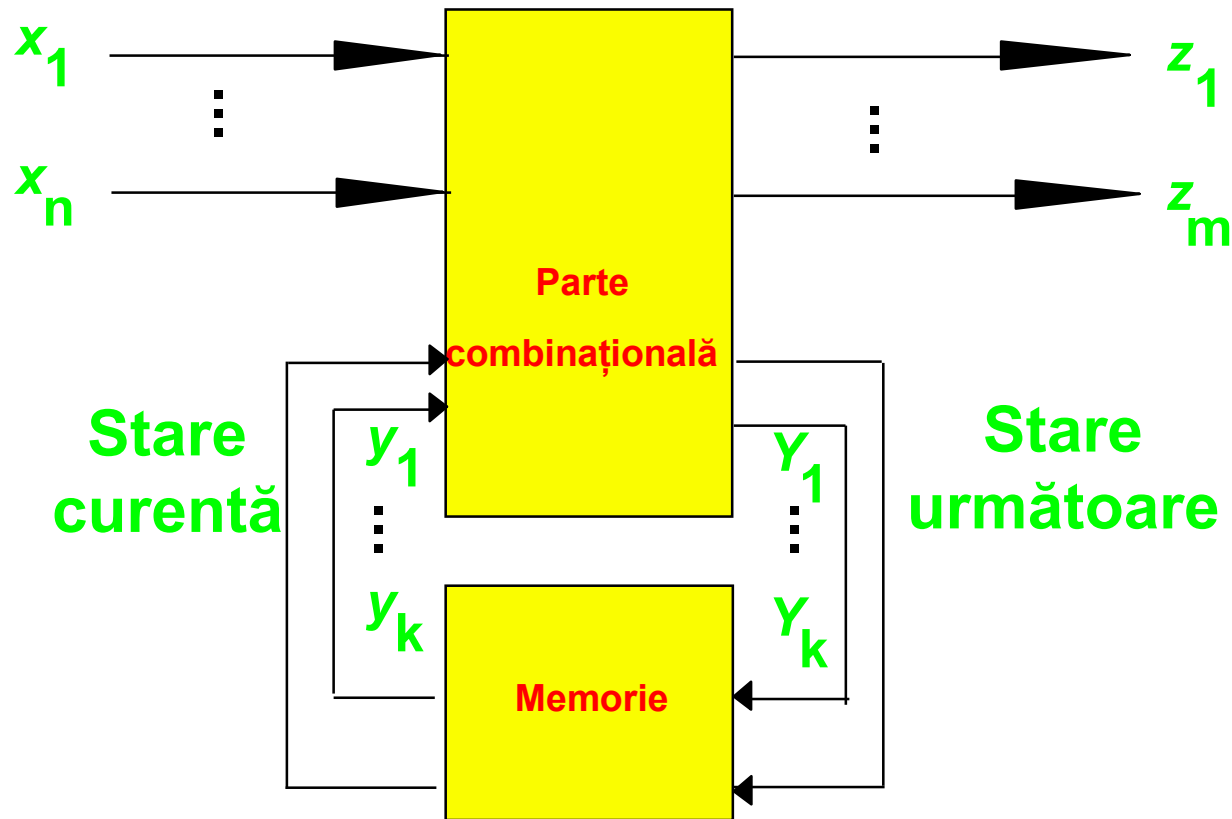
- "Circuite cu memorie"
- Exemplu: ascensorul
 - Input: număr (natural) indicând etajul-destinație
 - Output: număr (întreg) indicând câte etaje trebuie urcate/coborâte
 - Nu sunt suficiente toate valorile de intrare pentru a avea descrierea completă a funcționării circuitului
 - Memorie: suma algebrică a tuturor inputurilor

- Outputul depinde de intrare și de memorie – starea internă
- Ascensor:
 - Starea se schimbă la momente precise, stabilite
 - fie la momentul unui semnal regulat (ceas)
 - fie la momentul în care se întâmplă un eveniment
 - liftul ajunge în dreptul ușii unui nou etaj
- Primul mod se numește **sincron**, al doilea – mod **asincron**
- Ascensorul se află la începutul funcționării într-o stare numită **stare inițială**
 - etajul de plecare în prima cursă

Circuit secvențial

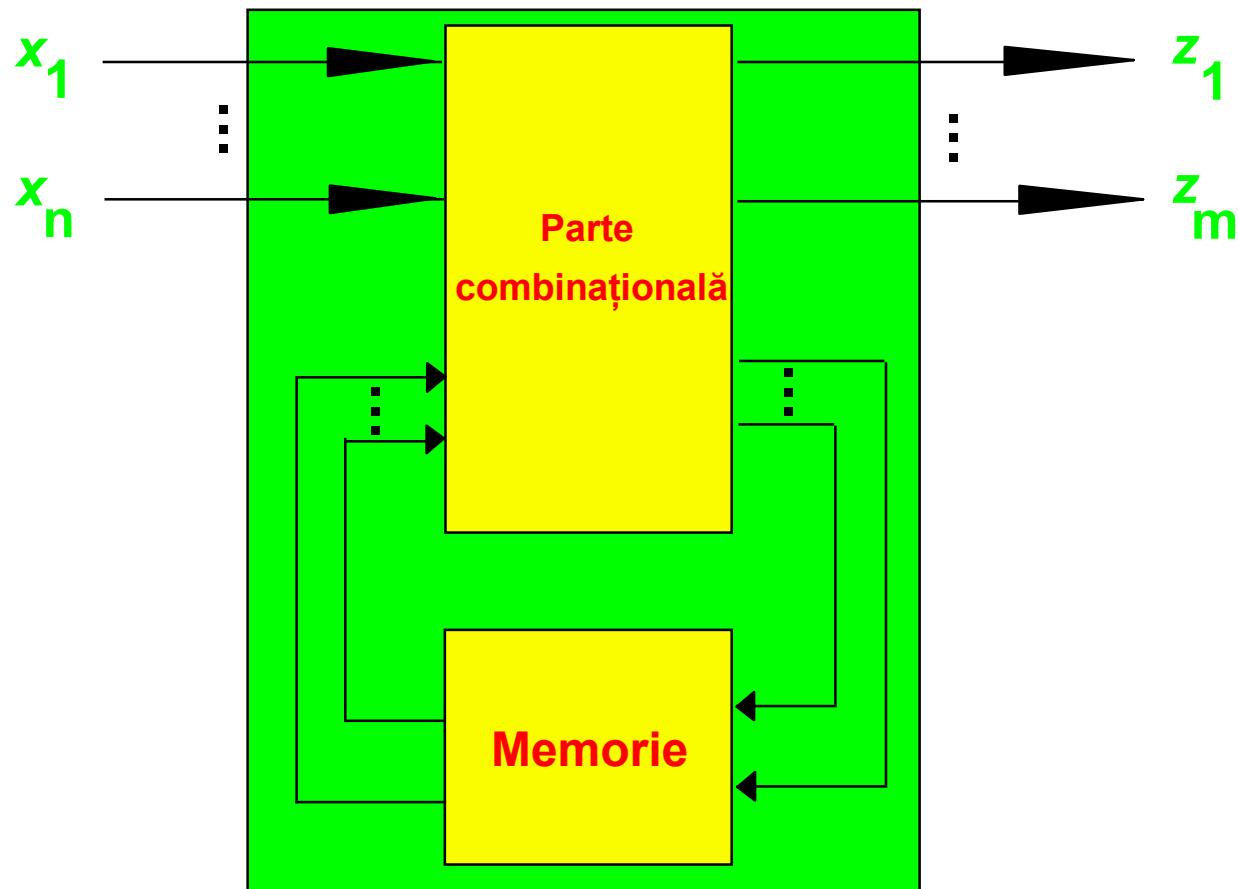
n variabile de intrare

m valori (“funcții”) de ieșire



Circuit secvențial – diagrama bloc

Control doar asupra input-urilor independente



n variabile de intrare

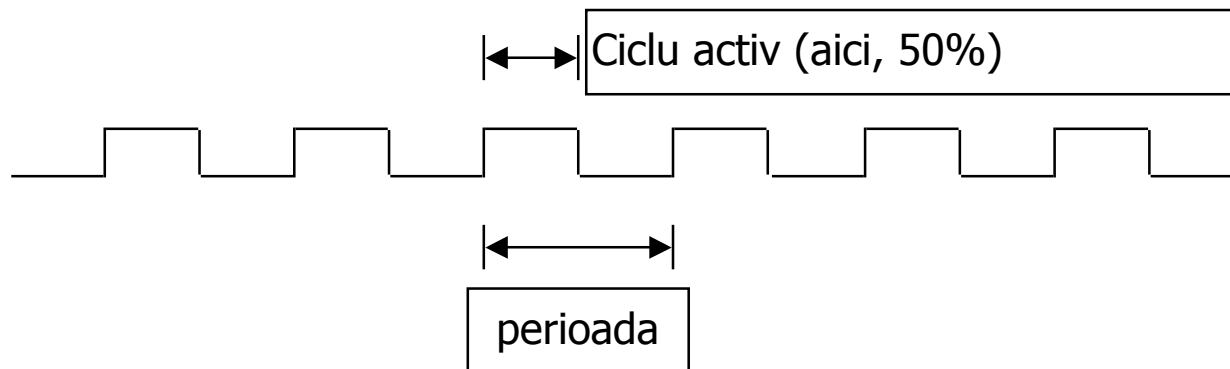
m valori ("funcții") de ieșire

Ceas

- Transmiterea semnalului prin porți și prin liniile de comunicare se face cu întârzieri:
 - Timp de tranziție între cele două niveluri
 - Întârziere de propagare input-output
- O soluție pentru funcționare corectă este utilizarea unui semnal ceas (Clock), care să **sincronizeze** funcționarea, oferind și suficient timp pentru stabilizarea tuturor semnalelor.

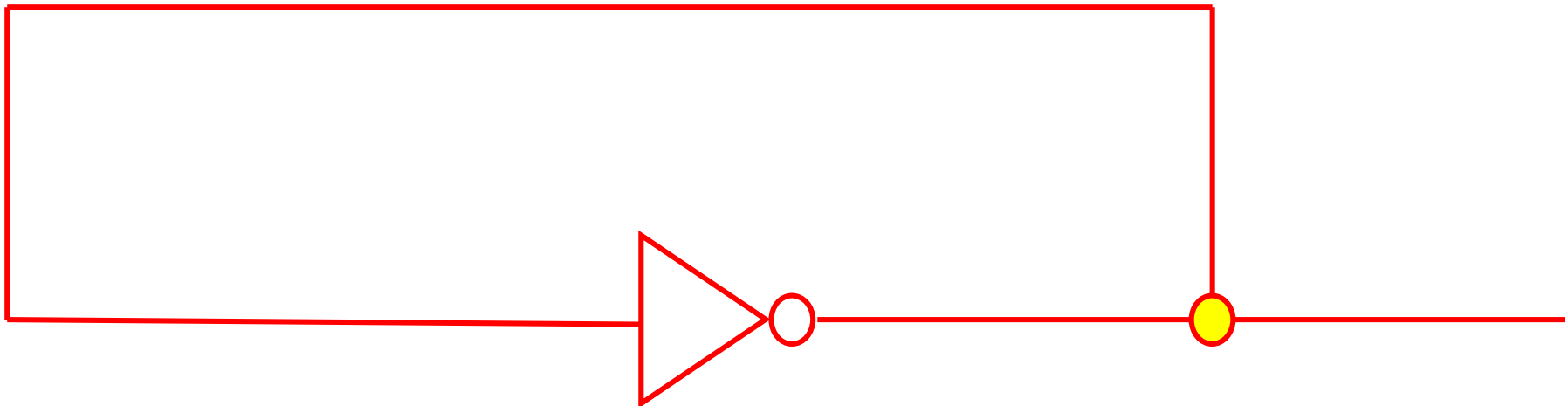
CEAS

- Circuit pentru controlul prin tacte de timp
 - Durata tactului suficientă pentru stabilizarea inputului
 - Permițând apoi efectul asupra stării memorate
- Semnalul de ceas este periodic
 - Perioada – timpul scurs între două "bătăi" ale ceasului
 - Ciclul activ ("de lucru") – în exemplu, timpul cât semnalul ceas are nivelul "High", exprimat ca procentaj din perioadă
 - » Sunt posibile implementări "activ pe nivel" (maxim, ca în exemplu, sau minim), sau "activ pe front" (crescător sau descrescător)



Diagramă logică

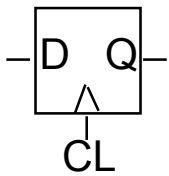
- Cea mai simplă diagramă logică ce produce semnal tip "ceas"
 - Conexiune inversă (feed- back)
 - Starea inițială



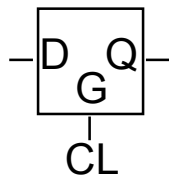
Tipuri de circuite secvențiale

- După modul de "sesizare" a ceasului
 - Latch-uri - elemente de memorie RS
 - Cu ceas, activ pe nivel; pot fi și asincrone
 - Flip-flop-uri / bistabili (activ pe front)
 - RS, D, JK, T
- Regiștri, contori
- Modelare abstractă: mașini cu număr finit de stări (automate)

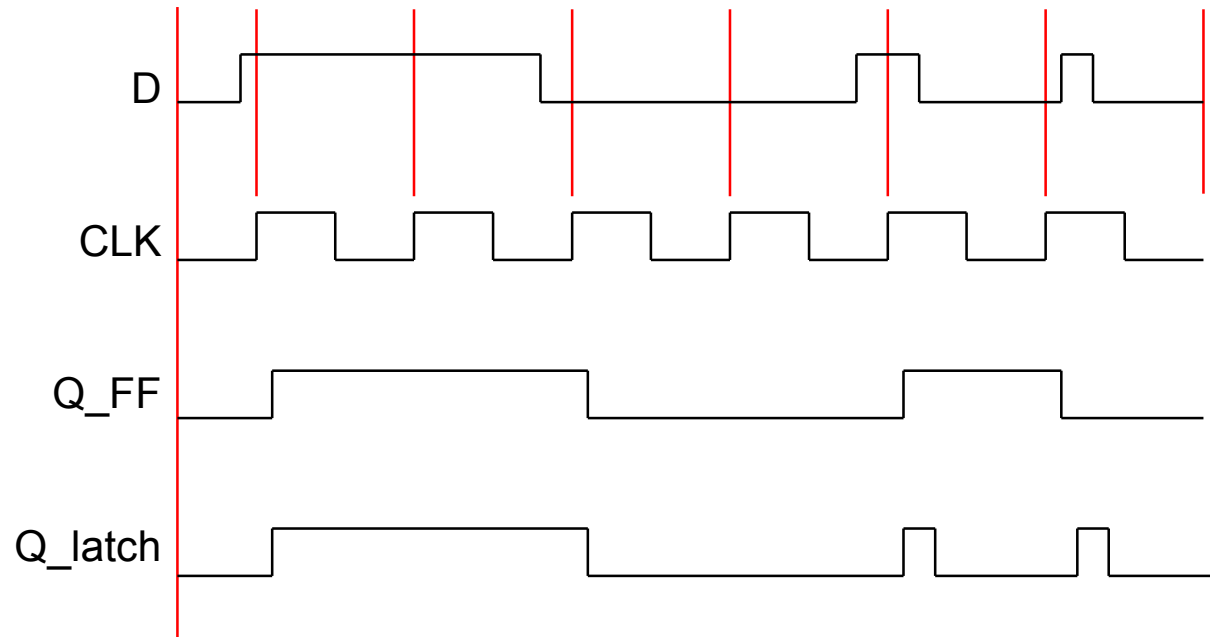
Comparație între funcționarea latch-urilor și a flip-flop-urilor



Activare
pe front crescător
(flip-flop)



Activare
pe nivel
(latch cu ceas)



Comportare identică, cu excepția cazului în care
inputul se schimbă când semnalul ceas este la nivel "High"

- Studiul latch-urilor și al flip-flop-urilor răspunde la întrebarea "Cum trebuie să arate un circuit care implementează **bitul**?"
- Specificațiile circuitului "bit":
 - Să se poată scrie în el un 0 sau un 1
 - Să se memoreze acea valoare până la scrierea alteia
 - Să se poată citi ultima valoare scrisă
- Circuitul "bit" nu poate fi circuit combinațional (condiția a doua)
- Cum poate arăta circuitul secvențial "bit"?