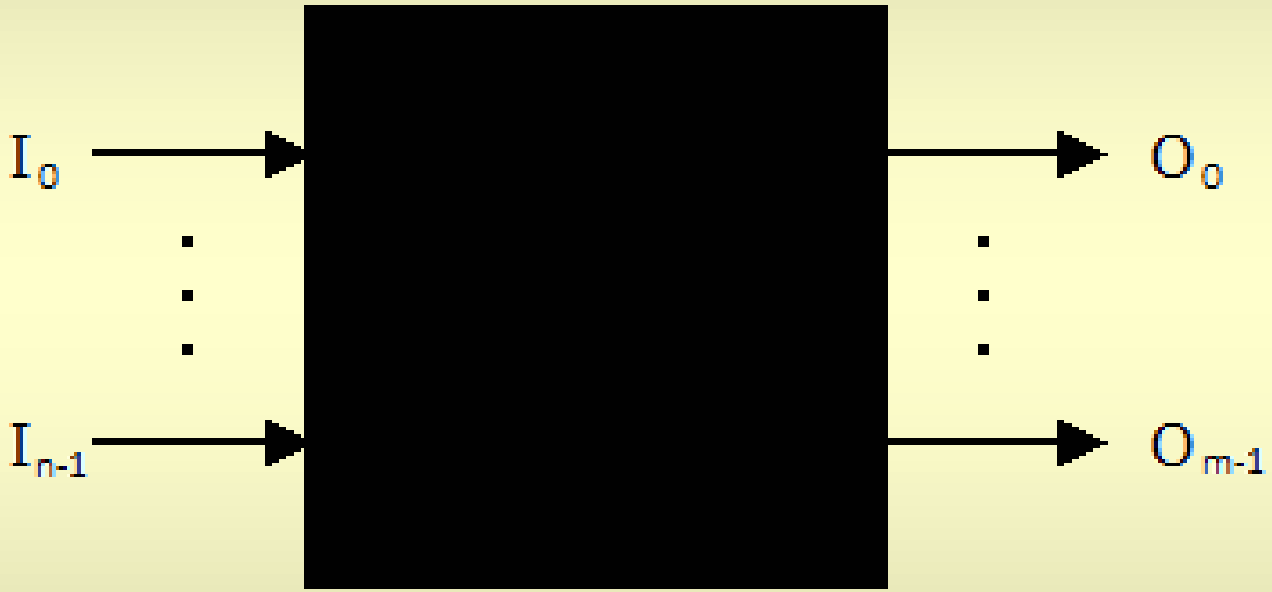
# II.5. Circuite combinaţionale

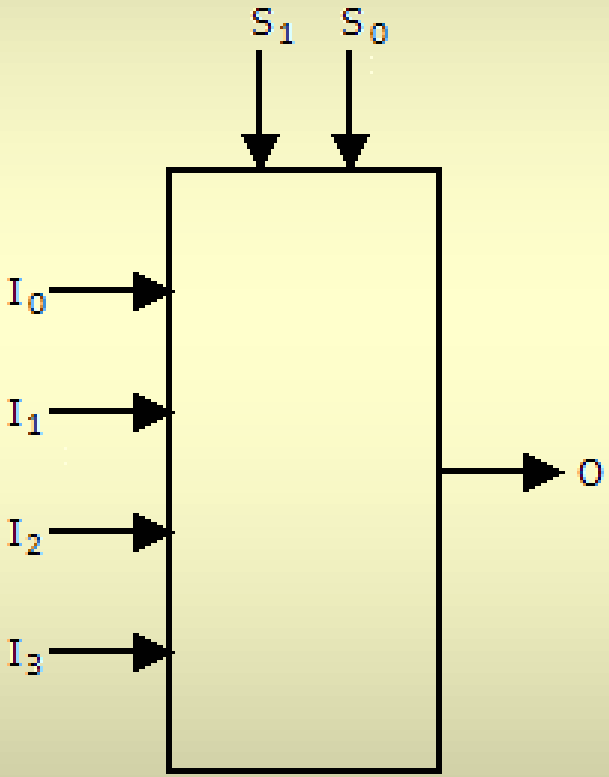



- valorile de la ieşire depind doar de valorile de la intrare din momentul respectiv

# Multiplexorul

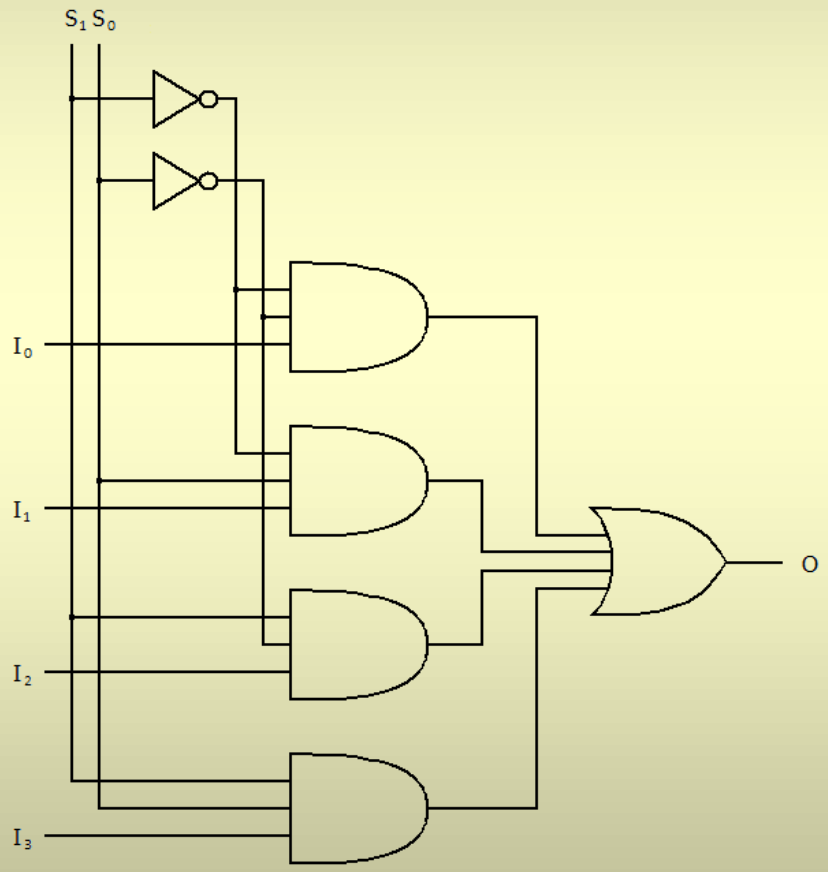
- $2^n$ intrări (de date)
- *n* intrări de selecţie (variabile de control)
  - biţi de control (de adresă)
- o singură ieşire
- fiecare intrare corespunde unui termen FND cu variabile de control
- una dintre intrări (bit) este selectată - devine valoare de ieşire

# 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)

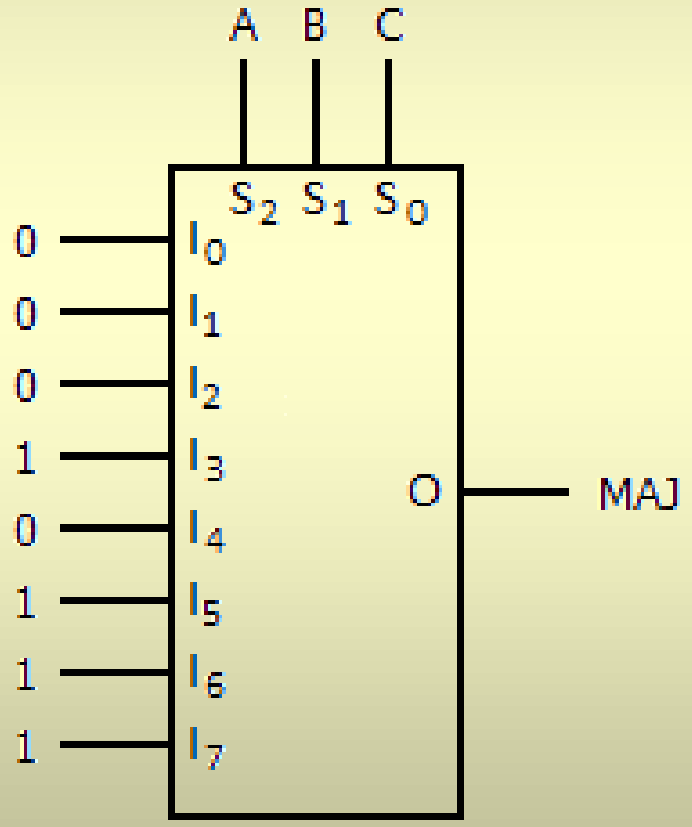


# Implementare funcții booleene

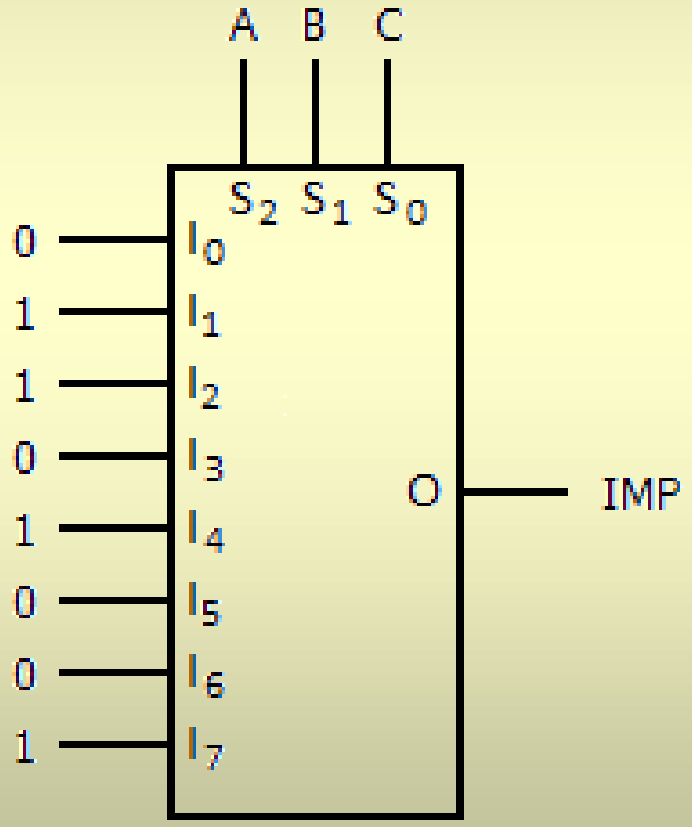
- intrările de selecție formează un număr
- care reprezintă indicele intrării de date care este selectată ca valoare de ieșire
- putem astfel implementa funcții booleene cu ajutorul multiplexorului
  - intrări de date - ieșirile corespunzătoare liniilor din tabelul de adevăr
  - intrări de selecție -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

# Exemple

majoritate din 3




imparitate

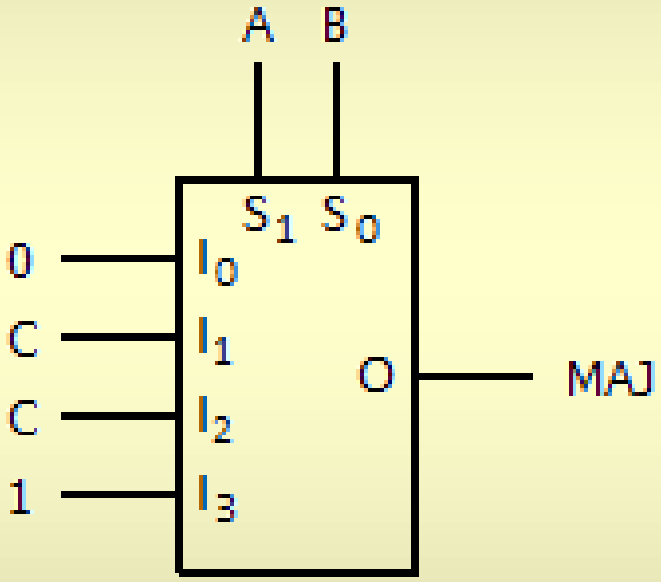


# Implementare eficientă - *folding*

majoritate din 3

| A | B | C | |
|---|---|---|---|
| 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 |

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

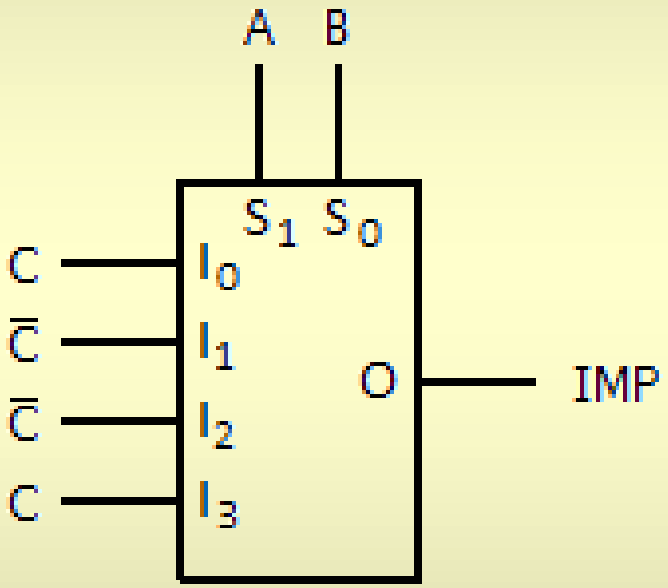


# Implementare eficientă - *folding*

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 |

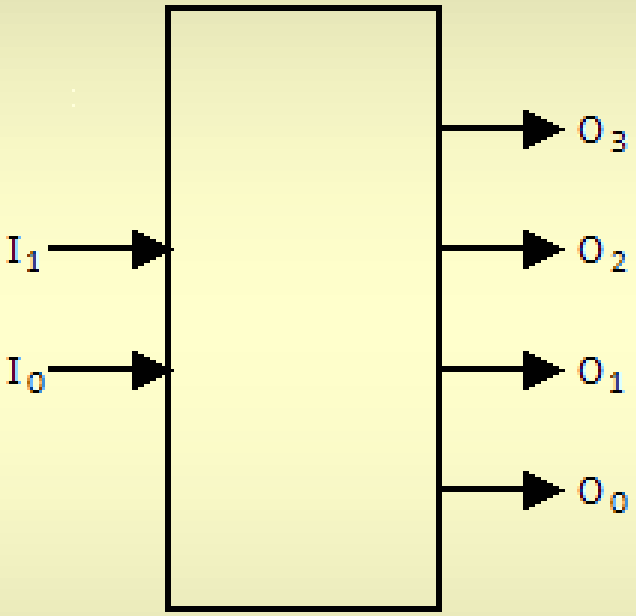
| A | B | |
|---|---|---|
| 0 | 0 | C |
| 0 | 1 | $\overline{C}$ |
| 1 | 0 | $\overline{C}$ |
| 1 | 1 | C |



# Decodorul

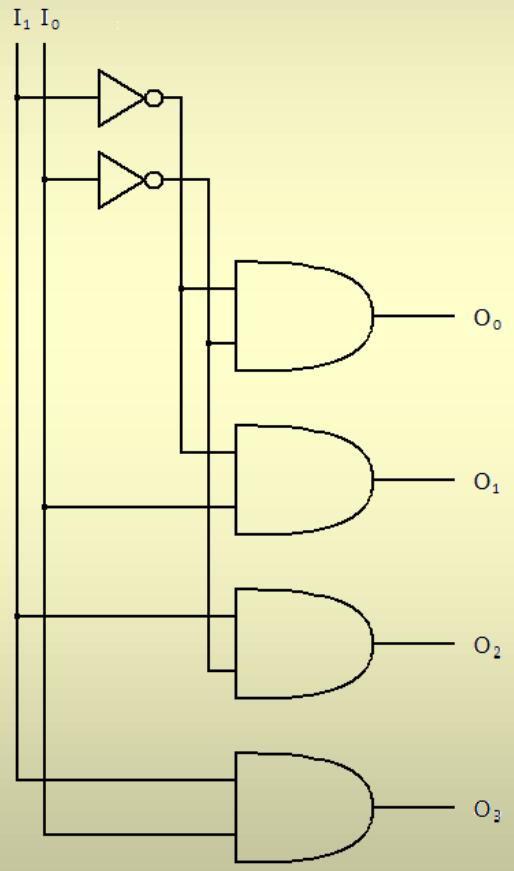
- $n$ intrări
- $2^n$ ieşiri
- în fiecare moment, exact una din ieșiri este activată
  - cea al cărei indice este egal cu numărul format de intrări
  - fiecare ieşire corespunde unui termen FND scris cu variabilele de intrare

# Decodor *n*=2



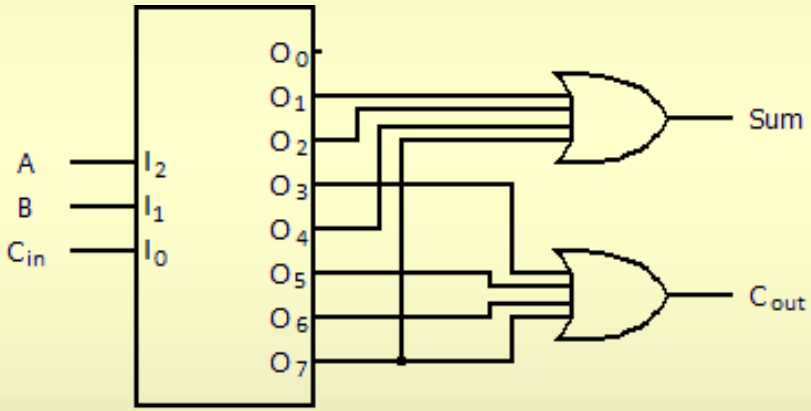

| $I_1$ | $I_0$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

# Diagrama logică (*n*=2)



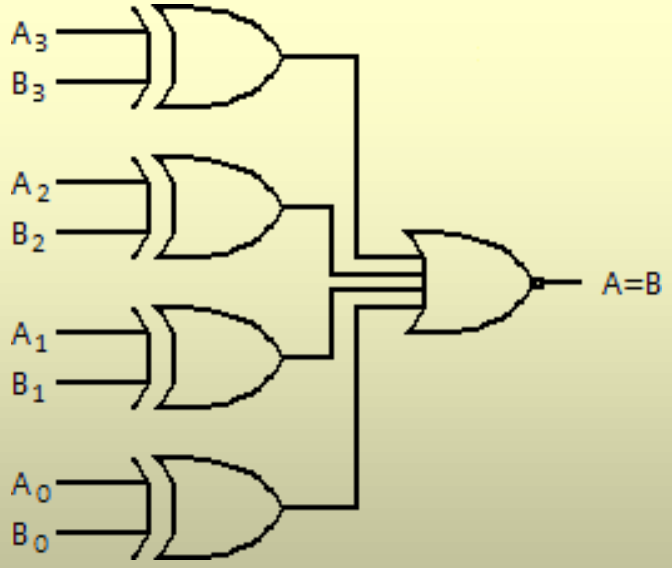
# Adunare - implementare cu decodor

| A | B | $C_{in}$ | 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 |



# Comparatorul

- operatorii de comparare: = , > , < , ≥, ≤
  - exemplu de implementare: egalitate pe 4 biţi
  - temă: comparator complet (<, =, >)

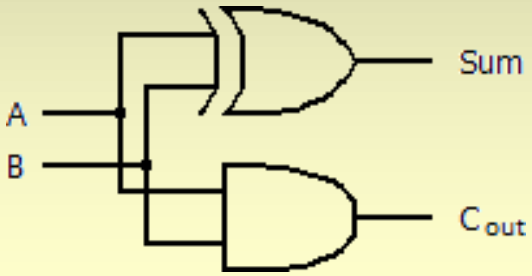


# Sumatorul

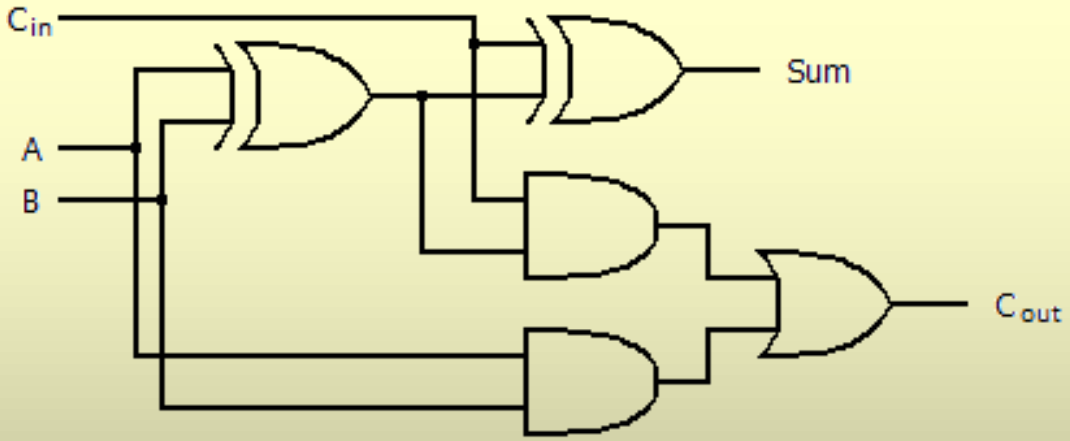
- semi-sumatorul (*half-adder*)
  - adună cei doi biţi de intrare
  - ieşire: un bit sumă şi un bit transport
  - nu poate fi extins pentru mai multe cifre
- sumatorul complet (*full adder*)
  - adună cei trei biţi de intrare (inclusiv transport)
  - aceeași ieşire: un bit sumă şi un bit transport
 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

# Diagrame logice

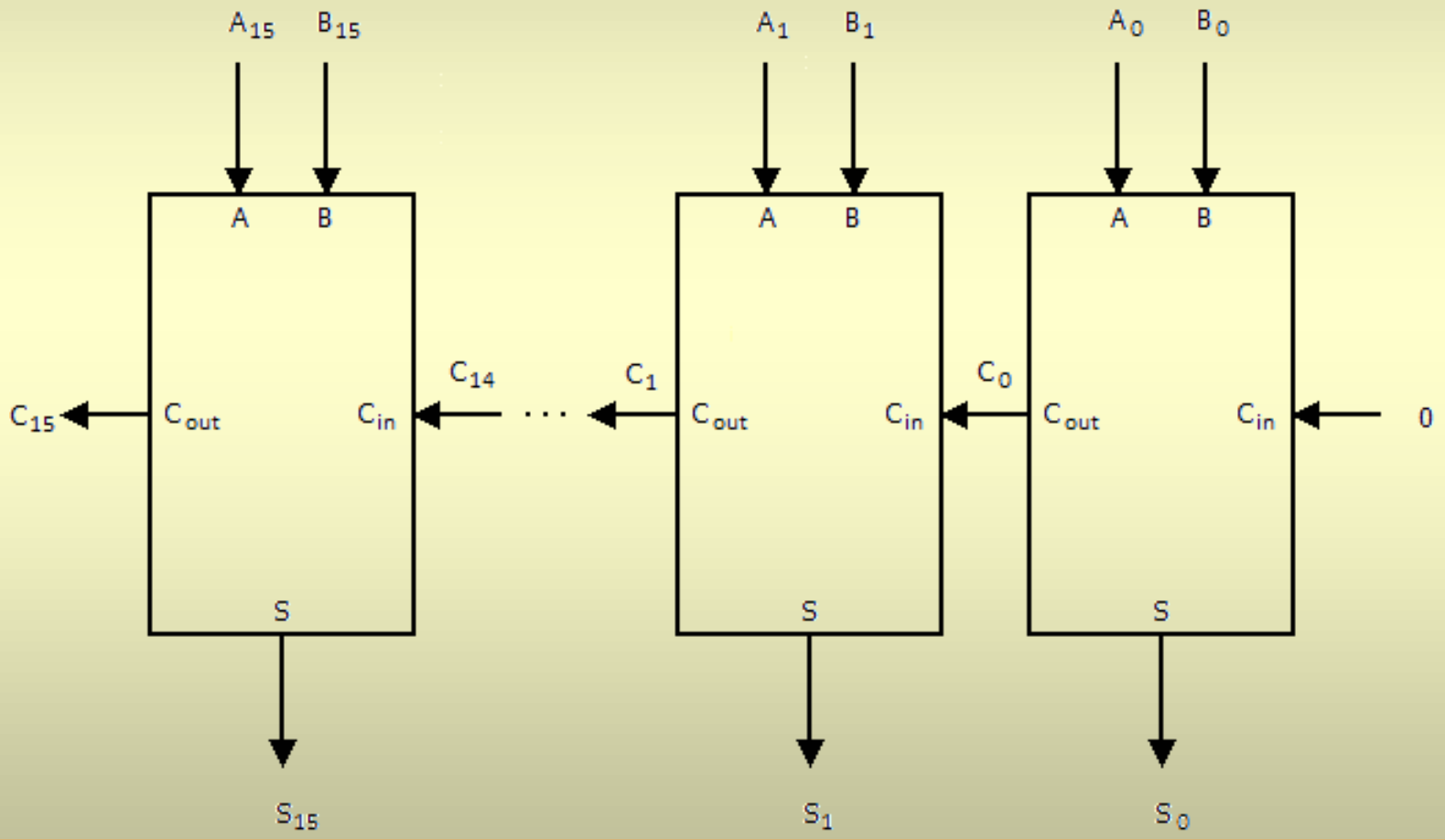
| A | B | Sum | $C_{out}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |




| 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 |



# 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 proporţională cu numărul de biţi

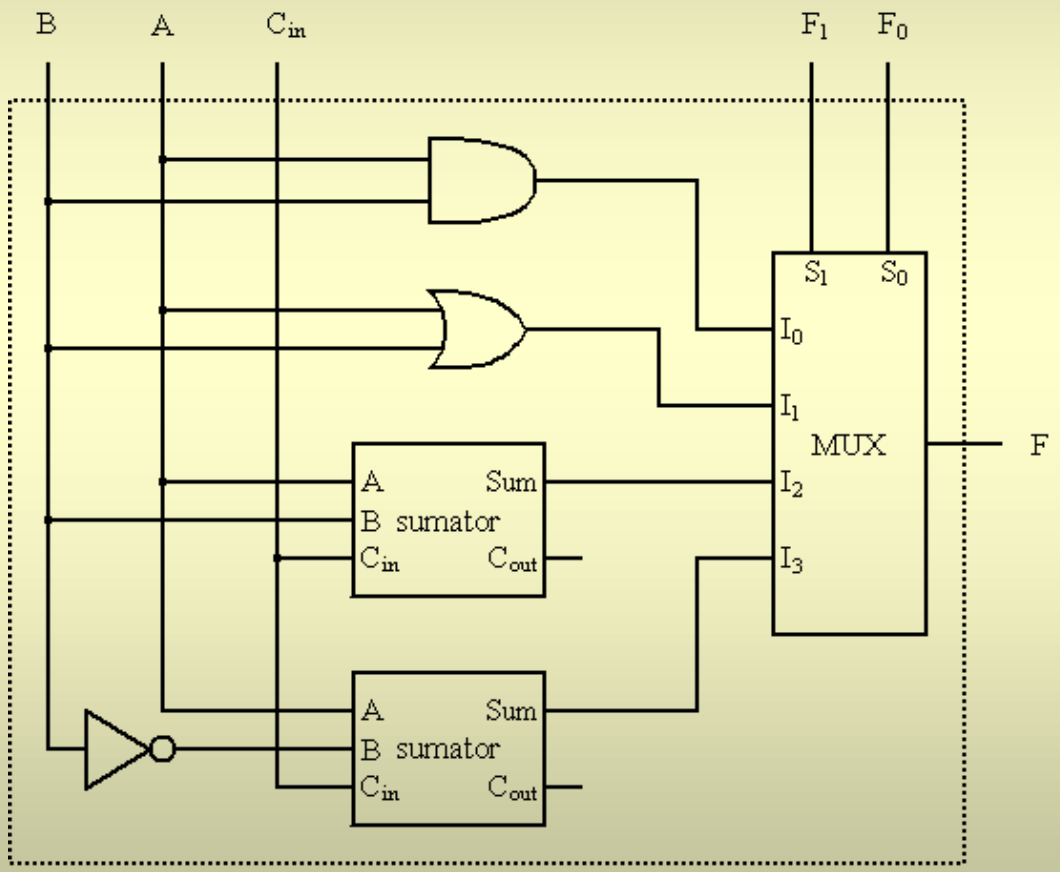
# Accelerarea adunării (1)

- 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 selecţia transportului
  - la fiecare rang se calculează suma pentru $C_{in}$=0 și $C_{in}$=1, apoi se selectează cea corectă

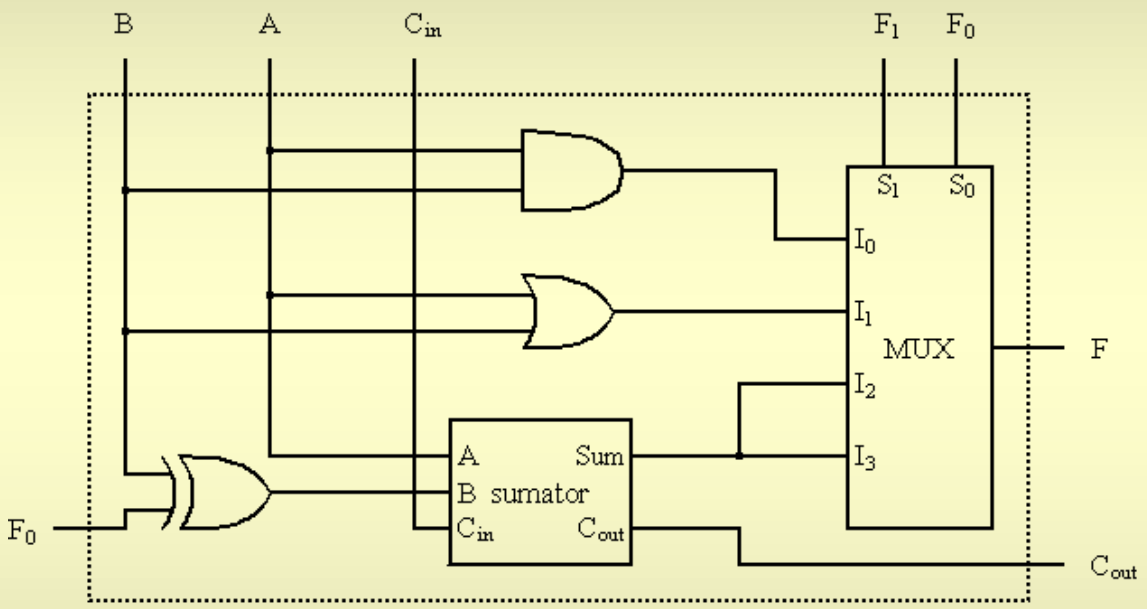
# Unitate aritmetică și logică elementară (1 bit)




| $F_1$ | $F_0$ | 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

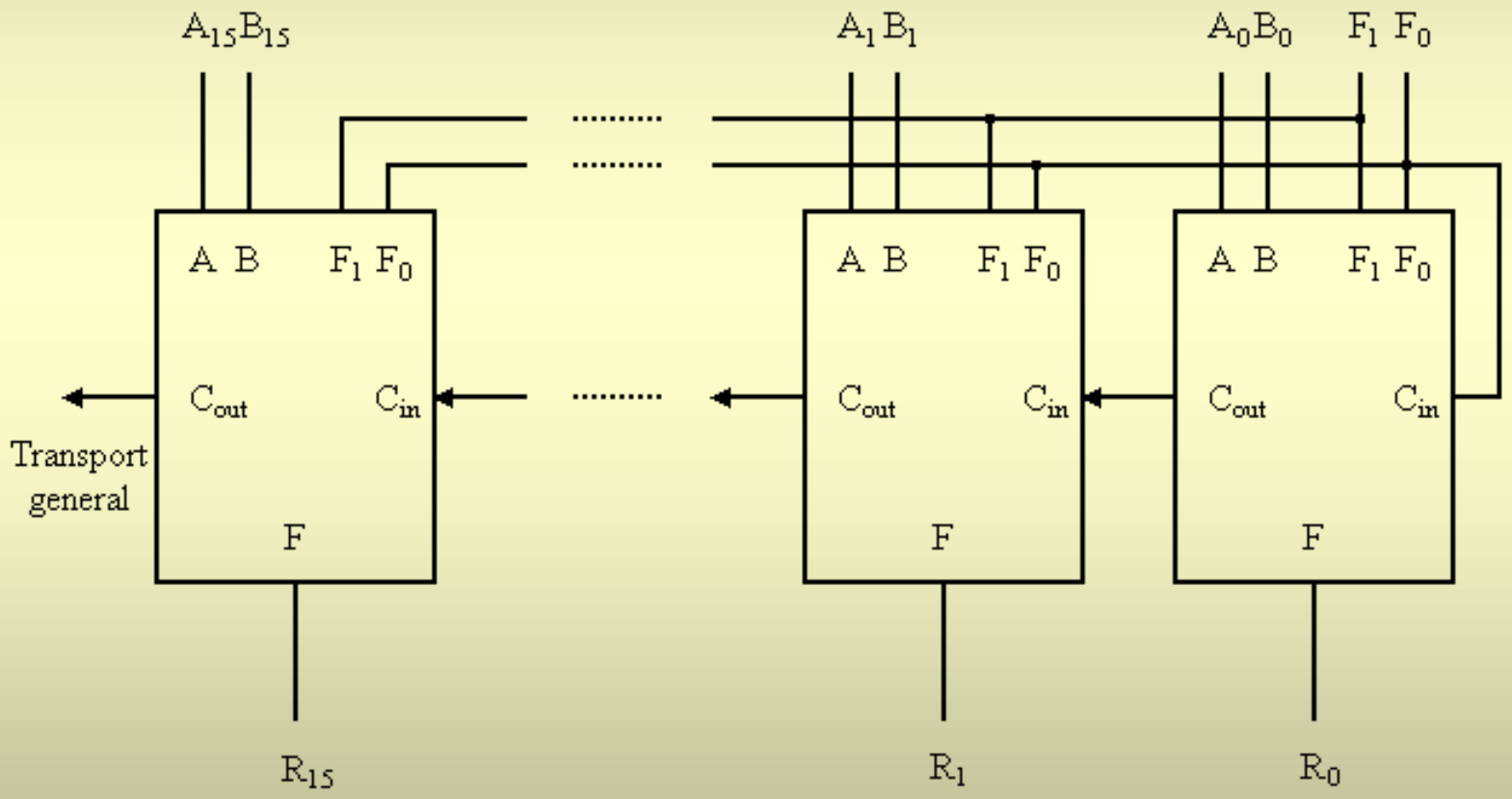
# Variantă îmbunătăţită




| $F_1$ | $F_0$ | 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

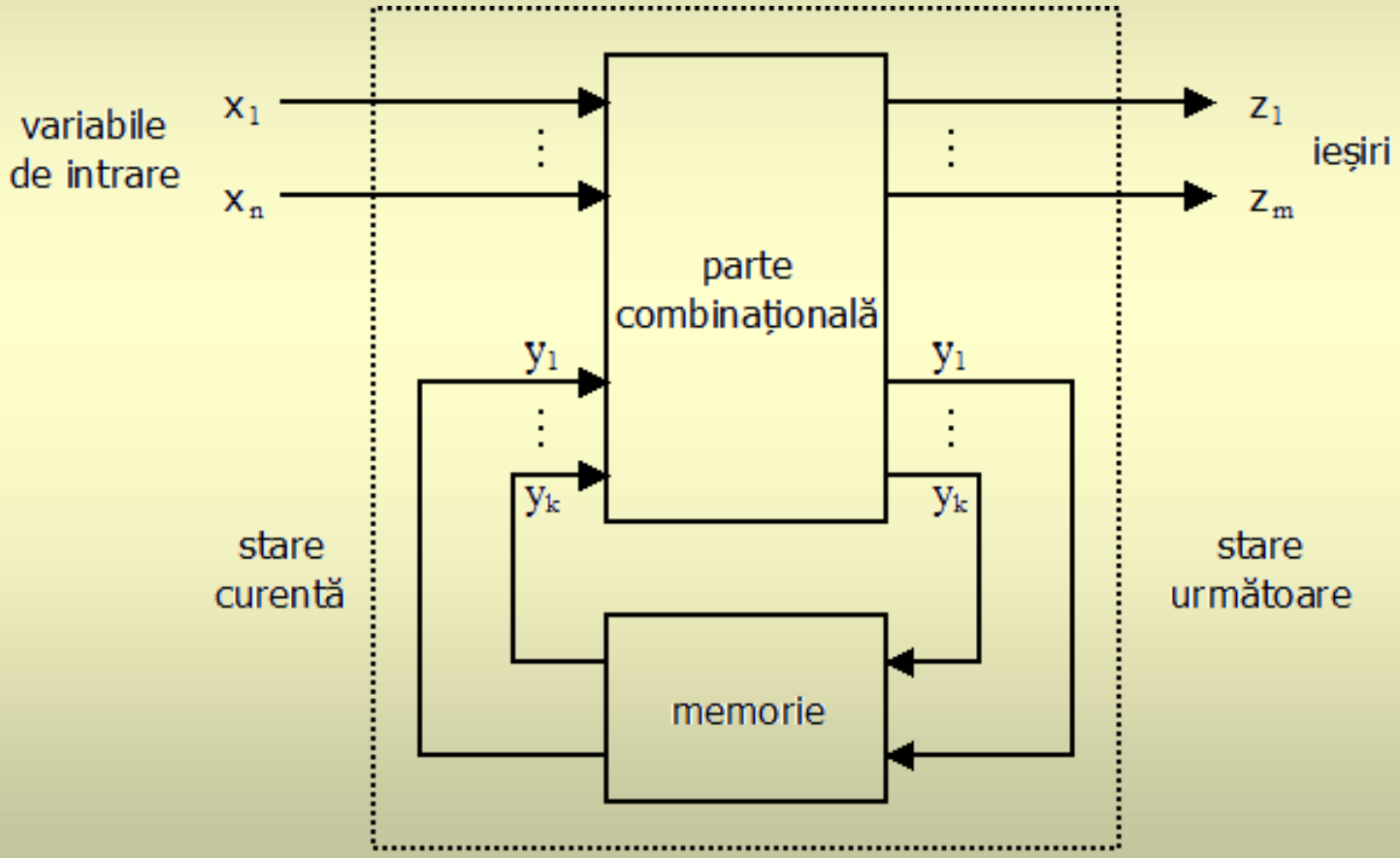


# III. Circuite secvenţiale

# Circuit secvenţial

- ieșirea la orice moment depinde de
  - intrare
  - starea internă
- deci pentru aceeași intrare se pot obţine valori diferite la ieșire, la momente diferite
- starea internă
  - este memorată de către circuit
  - evoluează în timp

# Diagrama bloc



# Evoluţia stării

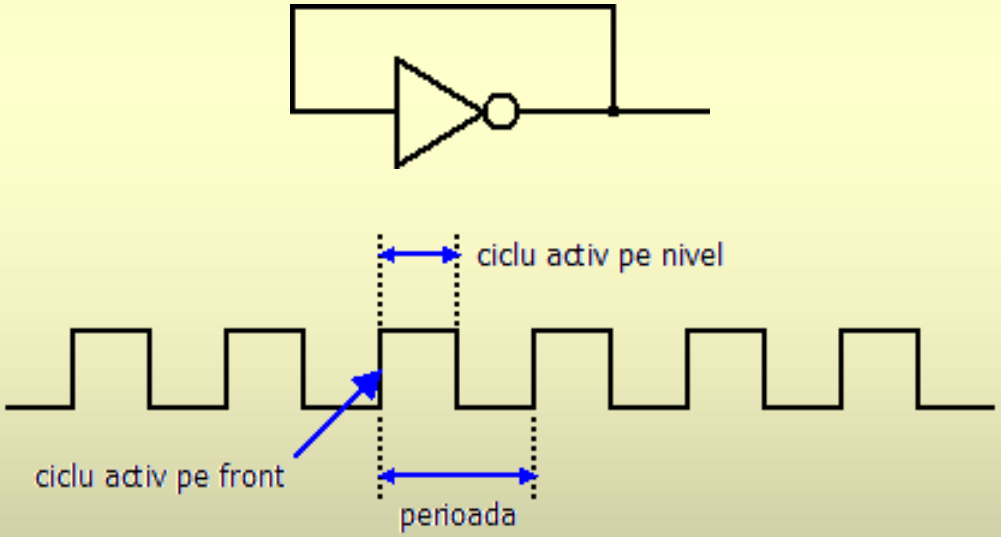
- starea se modifică la anumite momente
  - sincron: la intervale regulate de timp
    - date de un semnal special (ceas)
  - asincron: la momentul apariţiei unui eveniment
    - evenimentele sunt definite în funcţie de activitatea circuitului
  - de ce nu se modifică permanent?
    - transmiterea semnalului prin porţi şi prin liniile de comunicare se face cu întârzieri
    - semnalele - luate în considerare după stabilizare

# Ceasul

- semnal periodic
  - ciclu activ - procentajul din perioadă în care semnalul este activ
  - depinde de ce înseamnă semnal activ
    - pe nivel (0 sau 1)
    - pe front (trecerea de la 0 la 1 sau invers)
- durata perioadei
  - suficient de mare pentru a avea intrări stabile

# Implementare

- cea mai simplă variantă
  - esenţială este conexiunea inversă




- în general se folosesc scheme mai complexe

# Tipuri de circuite secvenţiale

- la nivel de bit - circuite bistabile
- după cum este detectat semnalul activ
  - *latch* - activ pe nivel
  - *flip-flop* - activ pe front
- circuite pe mai mulți biți
  - regiștri, numărătoare (contoare)
  - formate din mai multe circuite bistabile

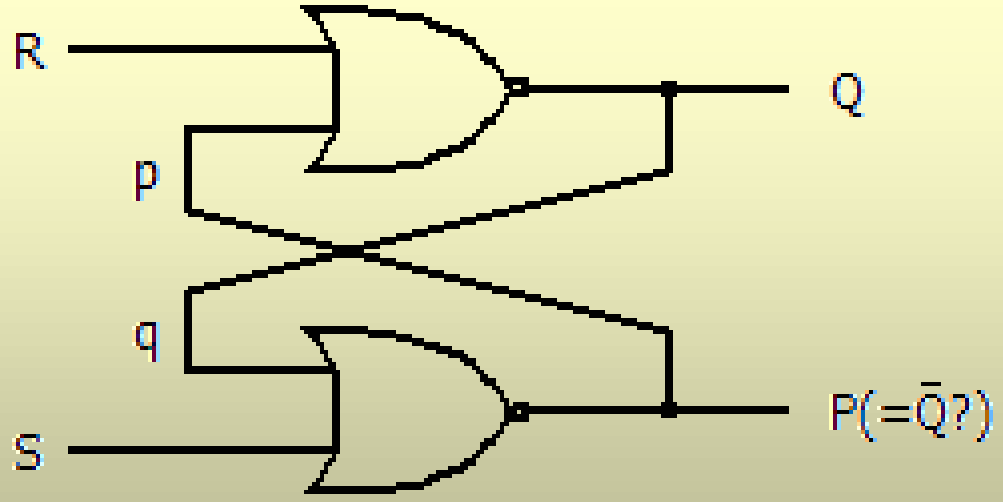
# III.1. Circuite bistabile

# Bistabil

- cum trebuie să arate un circuit care implementează bitul?
- specificații
  - să se poată scrie în el un 0 sau un 1
  - să memoreze acea valoare până la scrierea alteia
  - să se poată citi ultima valoare scrisă
- nu poate fi circuit combinaţional (memorare)

# Bistabil RS fără ceas

- două intrări (R,S), două ieșiri (Q,P), două conexiuni inverse
  - circuitul implementează un singur bit: $P = \overline{Q}$



# Funcționarea bistabilului RS (1)

- la prima vedere avem simultan

  $q = Q$ și $p = P$

  $Q = \overline{p + R}$

  $P = \overline{q + S}$

- de fapt, ieșirile nu se modifică instantaneu la modificarea intrărilor
  - datorită timpilor de propagare prin porți
  - deci putem studia evoluția prin tabele de adevăr

# Funcționarea bistabilului RS (2)

- considerăm (q,p) valorile curente ale ieșirilor
- iar (Q,P) valorile viitoare
  - în funcție de (q,p) și de intrările (R,S)
  - care vor deveni efective după timpii de propagare

Diagrama Karnaugh

ieșirile: QP

| qp\RS | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 11 | 10 | 00 | 01 |
| 01 | 01 | 00 | 00 | 01 |
| 11 | 00 | 00 | 00 | 00 |
| 10 | 10 | 10 | 00 | 00 |

$$Q = \overline{p + R}$$

$$P = \overline{q + S}$$

# Stări stabile

- în principiu, (Q,P) se modifică permanent
- dar atunci când (Q,P)=(q,p), avem o stare stabilă
  - dorim identificarea acestor stări
  - circuitul poate fi controlat dacă trece doar prin stările stabile

| qp\RS | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 11 | 10 | **00** | 01 |
| 01 | **01** | 00 | 00 | **01** |
| 11 | 00 | 00 | 00 | 00 |
| 10 | **10** | **10** | 00 | 00 |

# Funcționare (1)

| **stare inițială (q,p)** | **evoluție (q,p)** | **concluzie** |
|---|---|---|
| *R=0, S=1* | | |
| 00 | 00 → 10 stabil | circuitul evoluează întotdeauna spre starea stabilă (Q,P)=(1,0) |
| 01 | 01 → 00 → 10 stabil | |
| 10 | 10 stabil | |
| 11 | 11 → 00 → 10 stabil | |
| *R=1, S=0* | | |
| 00 | 00 → 01 stabil | circuitul evoluează întotdeauna spre starea stabilă (Q,P)=(0,1) |
| 01 | 01 stabil | |
| 10 | 10 → 00 → 01 stabil | |
| 11 | 11 → 00 → 01 stabil | |

# Funcționare (2)

| **stare inițială (q,p)** | **evoluție (q,p)** | **concluzie** |
|---|---|---|
| *R=0, S=0* | | |
| 00 | 00 → 11 → 00 → ... | pentru q=p, circuitul oscilează la infinit<br>pentru q≠p, circuitul își păstrează starea (stabilă) |
| 01 | 01 stabil | |
| 10 | 10 stabil | |
| 11 | 11 → 00 → 11 → ... | |
| *R=1, S=1* | | |
| 00 | 00 stabil | circuitul evoluează întotdeauna spre starea stabilă (Q,P)=(0,0) |
| 01 | 01 → 00 stabil | |
| 10 | 10 → 00 stabil | |
| 11 | 11 → 00 stabil | |

# Funcționare (3)

- reamintim condiția $P=\overline{Q}$

(R,S)=(0,0): păstrare stare existentă (memorare)

(R,S)=(0,1): stabilizare la Q=1 (set)

(R,S)=(1,0): stabilizare la Q=0 (reset)

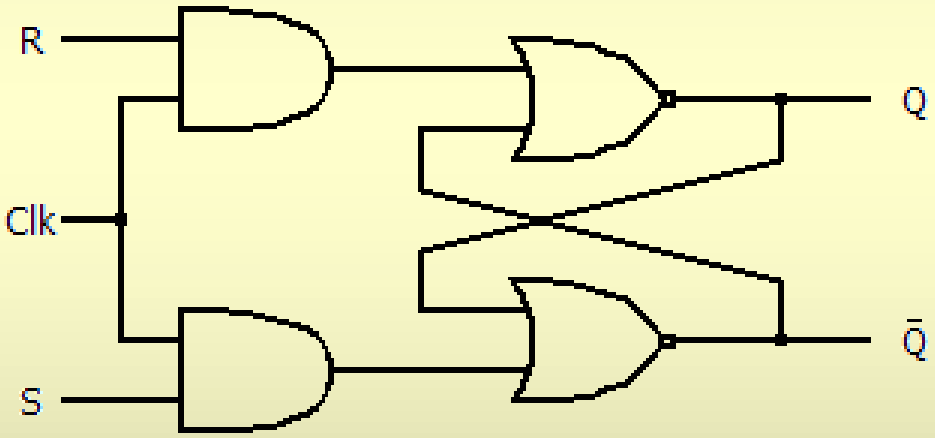
(R,S)=(1,1): combinație interzisă

  – deoarece P=Q - nu implementează un bit

# Circuite secvenţiale sincrone

- se adaugă bistabilului RS un semnal de sincronizare (ceas)
- pornind de la acesta se pot realiza alte circuite bistabile
  - D, JK, T
- toate sunt de tip latch (active pe nivel)

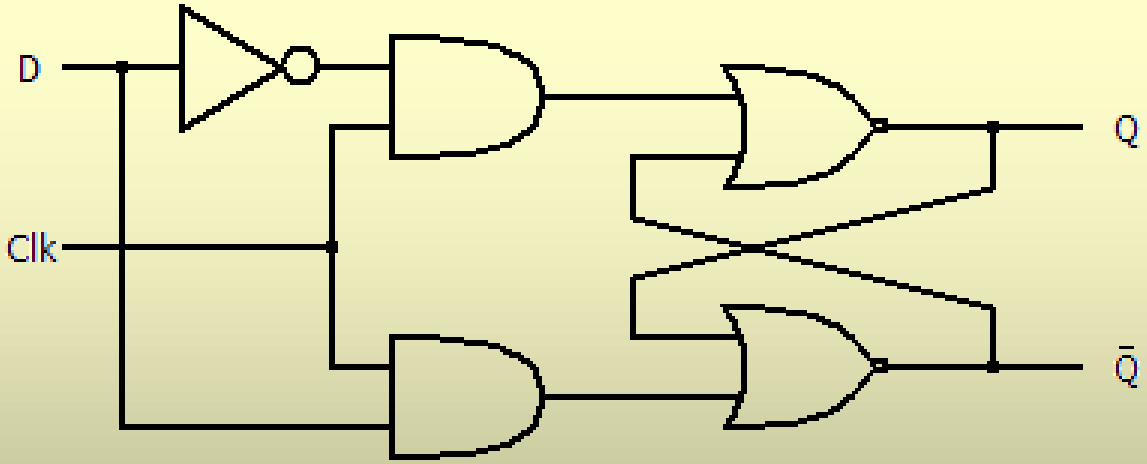
# Latch RS cu ceas




| $q(=Q_t)$ | R | S | $Q_{t+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | * |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | * |

# Latch D

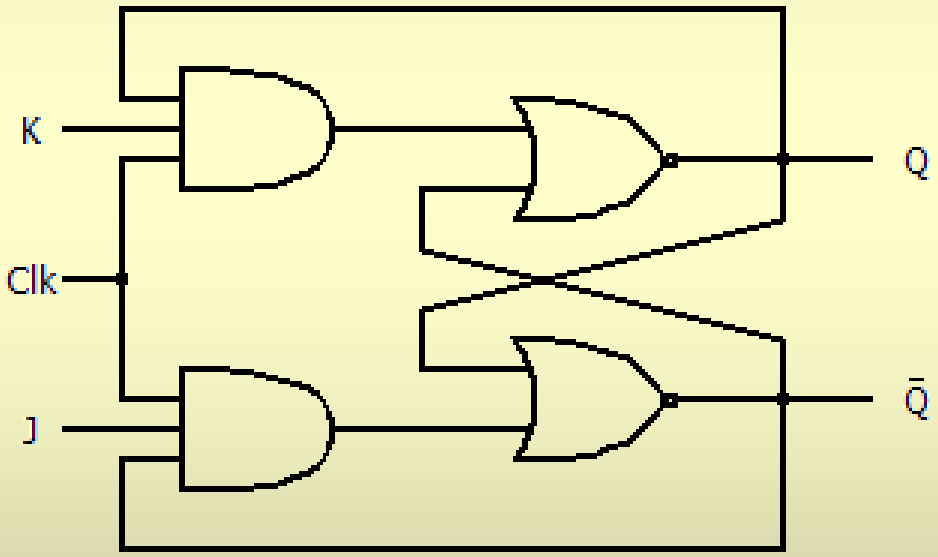
- modelează doar situaţiile R≠S
- elimină combinaţiile interzise
- aici ieșirea nu depinde de fapt de starea anterioară




| $q(=Q_t)$ | D | $Q_{t+1}$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# Latch JK

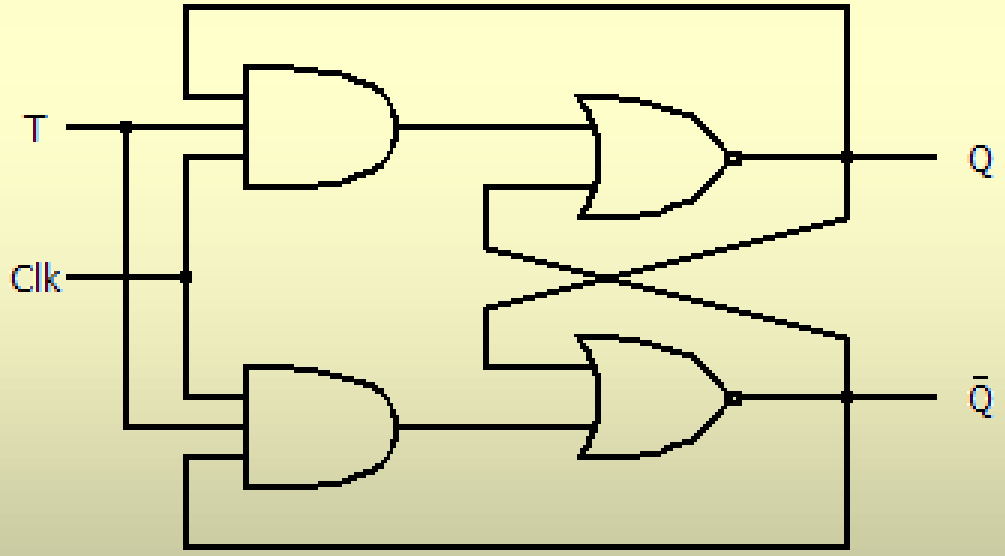
- elimină combinaţia imposibilă de la bistabilul RS




| $q(=Q_t)$ | J | K | $Q_{t+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# Latch T

- derivat din bistabilul JK
- modelează doar situaţiile J=K




| $q(=Q_t)$ | T | $Q_{t+1}$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Evoluţia stărilor

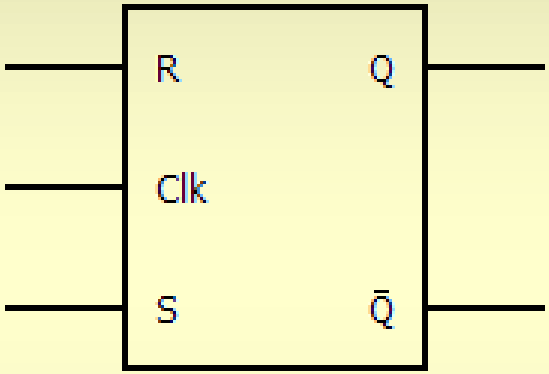
| R | S | $Q_{t+1}$ | |
|---|---|---|---|
| 0 | 0 | $Q_t$ | neschimbat |
| 0 | 1 | 1 | scriere 1 |
| 1 | 0 | 0 | scriere 0 |
| 1 | 1 | * | interzis |

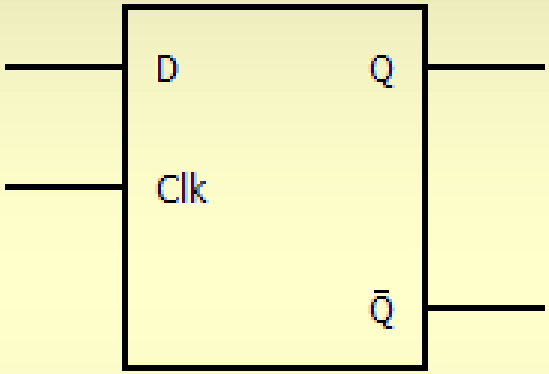
| J | K | $Q_{t+1}$ | |
|---|---|---|---|
| 0 | 0 | $Q_t$ | neschimbat |
| 0 | 1 | 0 | scriere 0 |
| 1 | 0 | 1 | scriere 1 |
| 1 | 1 | $\overline{Q_t}$ | inversare |

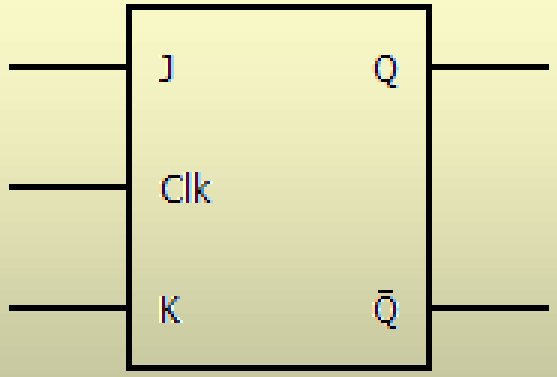
| D | $Q_{t+1}$ | |
|---|---|---|
| 0 | 0 | scriere 0 |
| 1 | 1 | scriere 1 |

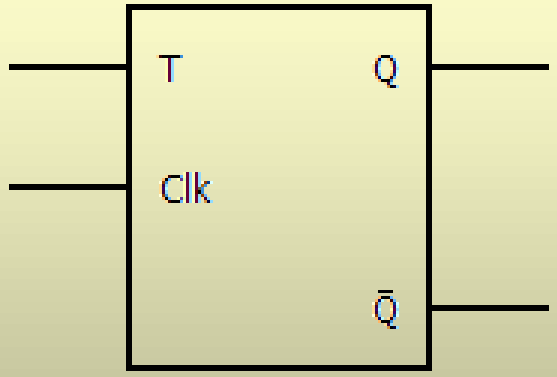
| T | $Q_{t+1}$ | |
|---|---|---|
| 0 | $Q_t$ | neschimbat |
| 1 | $\overline{Q_t}$ | inversare |

# Diagrame bloc pentru bistabili



# Temă

- implementaţi și analizaţi comportamentul bistabilului RS (fără ceas) utilizând porţi NAND în locul porţilor NOR
- similar pentru bistabilii latch RS, D, JK, T