Facultatea de Informatică - Iaşi

Arhitectura Calculatoarelor

Curs pentru anul I Informatică

Prof. dr. Henri Luchian

Lect. dr. Vlad Rădulescu

- examinare
  - două teste scrise din materia de curs
    - câte unul pentru fiecare jumătate de semestru
  - un test practic la laborator
    - limbaj de asamblare
- condiţia pentru susţinerea testelor scrise
  - prezenţa la laborator
    - cel mult 2 absenţe permise în fiecare jumătate de semestru

Autori ale căror prezentări publice au fost folosite pentru pregătirea acestei forme a cursului

- Sivarama Dandamundi
- Jerry Breecher
- Randy Katz
- Michel Allemand
- Daniel Amyot
- John Morris

CUPRINS

prima jumătate a semestrului

I. Introducere

II. Circuite combinaţionale şi funcţii booleene

III. Circuite secvenţiale şi automate

IV. Reprezentări interne

V. Arhitectura şi organizarea calculatorului

Capitolul I
Introducere

I.1. EVOLUȚIE

Când au apărut maşinile de calcul?

- După fiecare redefinire a noţiunii de **calcul**
  - **abac**: adunări
  - **roţi dinţate**: Leibniz (2) şi Pascal (10): adunări, înmulţiri
  - **Babbage**: instrucţiuni din exterior, calcul ramificat
  - **von Neumann**: program memorat; execuţie în secvenţă de instrucţiuni; ierarhii de memorii
  - **calculatoare "paralele"** (de fapt, calcul paralel)
  - **calcul**(atoare) **probabilist**(e)**, neuronal**(e)**, evolutiv**(e)**, cuantic**(e) **…**
- Încearcă automatizarea calculului în înţelesul respectiv

Maşini de calcul universale

- O maşină de calcul universală se poate comporta ca oricare maşină de calcul particulară
- Exemple:
  - Introducând în calculator un program corect de lucru cu
    - matrici, calculatorul se va comporta ca o maşină de calcul cu matrici
    - linii, unghiuri, forme – maşină de proiectare grafică
    - cuvinte, paragrafe, texte – maşină de tehnoredactat

Scurtă istorie
a ideilor

- inventarea scrierii poziţionale a numerelor
  - indieni, arabi
- inventarea logaritmilor
  - John Napier of Edinburgh
- algebra booleană
  - George Boole, 1854
- teorema de incompletitudine
  - Kurt Gödel, 1935
- conceptul de calculator neumannian
  - John von Neumann, 1946
- Toate sunt legate de **calcul**(**ator**) în înţelesul de astăzi

Scurtă istorie
invenţii abstracte şi concrete

- 1850: George Boole inventează algebra booleană
  - propoziţii logice sunt transformate în simboli
  - calcule cu propoziţii logice, folosind reguli de tip matematic
- 1938: Claude Shannon leagă algebra booleană de circuite (comutatoare)
  - în teza sa de dizertaţie
- 1945: John von Neumann proiectează primul calculator cu program memorat
  - comutatoarele erau **lămpi**
- 1946: ENIAC – primul calculator electronic
  - 18000 de lămpi
  - 5000 de adunări, sute de înmulţiri pe secundă
- 1947: Shockley, Brittain şi Bardeen inventează **tranzistorul**
  - permite integrarea mai multor circuite într-un modul
  - deschide drumul electronicii moderne

Scurtă istorie
a calculatoarelor

- 1642-1945: Calculatoare mecanice
  - Leibniz, Pascal, Babbage; Z1, Mark I
- 1945-1955: Lămpi
  - ENIAC, EDSAC, UNIVAC, IBM 70x
    - 1952: primul succes comercial – 19 calculatoare IBM vândute
- 1955-1965: Tranzistori
  - PDP-1, IBM 7094, CDC 6600
- 1965-1980: Circuite integrate
  - IBM 360, PDP-11, 4004, 8008, VAX
  - VLSI, ULSI

Scurtă istorie
a microprocesoarelor

- IBM-PC (8086): 1980
- Mac Plus (68000): 1984
- 80486: 1990
- Pentium: 1994
- Pentium II: 1997
- Pentium III: 1999
- Pentium IV: 2001

I.2. LEGI EMPIRICE

Legi empirice

- Legi ale oricărei ştiinţe depind într-un fel sau altul de experiment sau de observaţii în lumea concretă
  - unele ştiinţe sunt – sau au capitole – preponderent empirice (medicina)
- **Repetabilitatea**, inerentă noţiunii de experiment, duce la ideea de *legi empirice*: adevăruri valabile de cele mai multe ori, conform observaţiilor
- Formulate în Informatică încă de la începuturi
  - inginerie

Legi empirice în informatică

- Există legi empirice aplicabile hard-ului sau soft-ului:
  - legea "90:10" (Donald Knuth): 90% din timpul de execuţie al unui program este utilizat pentru 10% din instrucţiuni
  - legea lui Amdahl: eficienţa maximă în îmbunătăţirea unui sistem (concret sau abstract) se atinge dacă se optimizează subsistemul cel mai folosit
  - legile localizării – spaţială, temporală

Legea lui Amdahl

- Îmbunătăţirea unui sistem trebuie făcută în partea cel mai frecvent folosită

$$A(a, f_a) = \frac{1}{(1 - f_a) + \frac{f_a}{a}}$$

- Pentru creşteri semnificative, trebuie ca **a** şi $\mathbf{f_a}$ să fie cât mai mari
  - **Exemplu**: Dacă procesorul lucrează 50% din timp şi devine de două ori mai rapid, atunci A = 4/3
  - **Observaţie:** Sumatorul va fi cel mai intens îmbunătăţit

Legile localizării
(pentru instrucţiuni)

- Localizare spaţială:
  - dacă la un moment dat se execută o instrucţiune **i**, atunci este probabil ca la momente apropiate să se execute instrucţiuni din apropierea lui **i (**în ordine fizică)
    - **spaţiul** de memorie, în care se află instrucţiunile
- Localizare temporală:
  - dacă la un moment dat se execută instrucţiunea **i**, atunci este probabil ca instrucţiunea **i** să se execute la momente apropiate în timp

Ordine fizică şi ordine logică

- Instrucţiunile de executat se află în memorie într-o anumită ordine: ordinea fizică
- Ele se citesc din memorie şi se execută
  - regula: în ordinea în care sunt memorate
  - excepţia: sărind peste un număr de instrucţiuni
- Rezultă ordinea logică a instrucţiunilor
  - În ordinea logică la o anumită rulare, fiecare instrucţiune poate să apară de 0, 1, 2, … ori
- Legile localizării indică un tip esenţial de relaţie între ordinea fizică şi ordinea logică:
  - **Secvenţa de instrucţiuni din ordinea logică este alcătuită în general din sub-secvenţe construite din porţiuni restrânse ale secvenţei de instrucţiuni din ordinea fizică.**

- Legile localizării: în fiecare interval "mic" de timp, programul utilizează o mică porţiune din spaţiul de adrese
- Valabil pentru instrucţiuni ca şi pentru date
  - Temporală: iteraţii (program), reutilizări
  - Spaţială: ordinea fizică; tablouri de date
- De mai bine de 20 de ani, soluţiile hardware se bazează pe legile localităţii pentru a câştiga viteză
- Un exemplu: ierarhia de memorii

I.3. IERARHIA DE MEMORII

Ierarhia de memorii

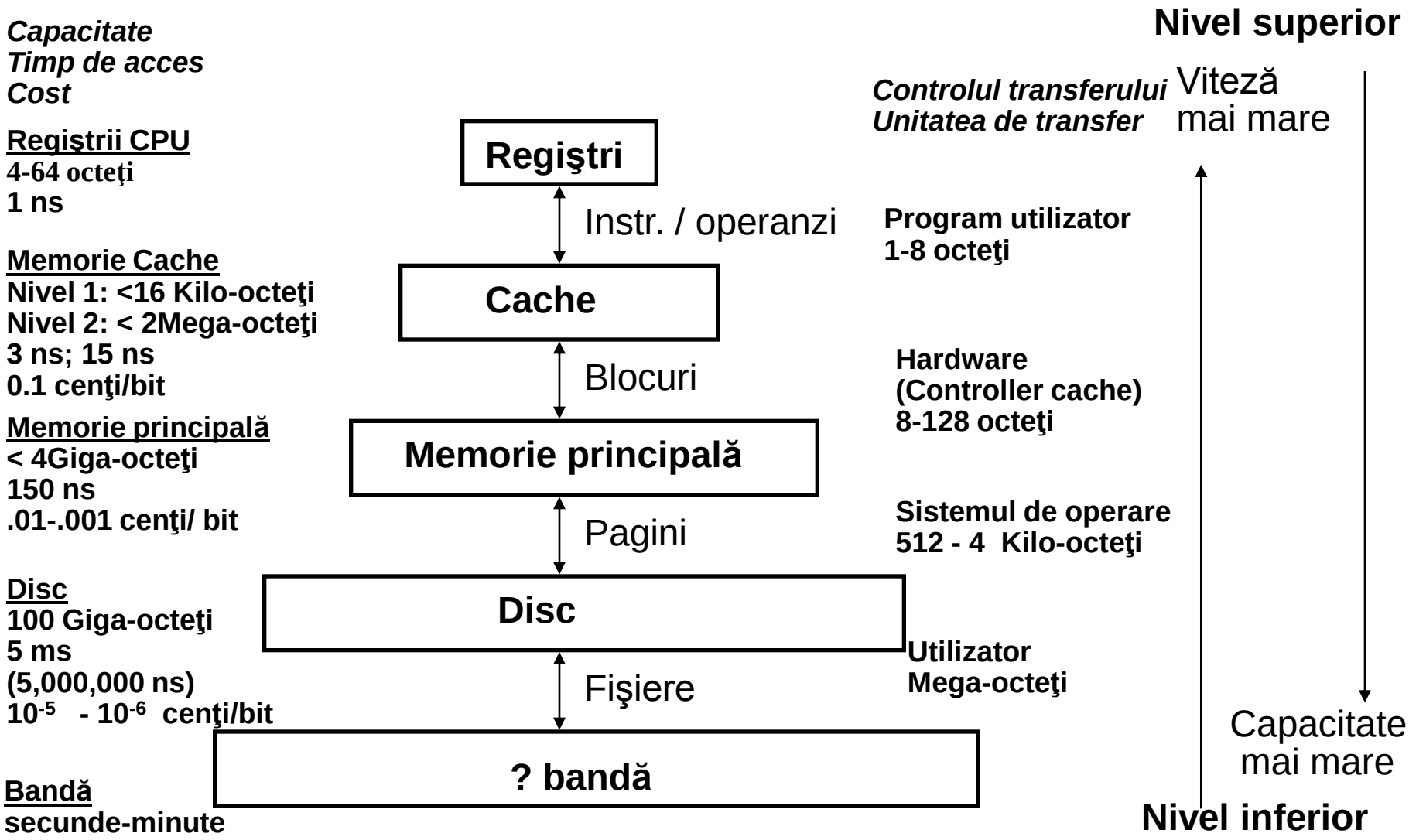
- Soluţionează două probleme:

1. Programele foarte mari nu încap în memoria principală
   - Nivelul inferior al ierarhiei (memoria secundară)
2. Necesitatea creşterii vitezei de execuţie a programelor
   - Nivel superior al ierarhiei (memoria cache)

Ierarhia de memorii

- Niveluri:
  1. regiştrii-generali – procesor (≤ 1/4 Ko)
  2. memoria cache (≤ 1 Mo)
  3. memoria RAM (≤ 4 Go)
  4. memoria secundară – disc (≤1 To)
  - Preţul pe bit scade pe măsura depărtării de procesor
- **Timp de comunicare** şi **capacitate de memorare** foarte mici pe primele două niveluri
  - corespunzând "ferestrelor" din legile localizării
- Dar comparativ foarte mari pe ultimul
  - permiţând memorarea programelor foarte mari

Nivelurile ierarhiei de memorii



Ierarhia de memorii

- Programul întreg se află în memoria secundară
- Cu fiecare nivel urcat în ierarhie, sunt aduse de pe nivelul inferior "ferestre" din ce în ce mai mici ale programului
- Execuţia – direct din cache
  - succes: instrucţiunea următoare este găsită în cache
  - eşec: instrucţiunea următoare trebuie adusă de pe un nivel inferior
  - eşecuri dese ar cauza o execuţie mai lentă decât dacă instrucţiunile ar fi aduse direct din memoria secundară
- Legile localizării fac ca ierarhiile de memorii să accelereze execuţia programelor
    - pe porţiuni
    - o valoare uzuală a ratei de succes poate fi de ordinul a 95%

Capitolul al II-lea
Circuite combinaţionale şi funcţii booleene

II.1. INTRODUCERE

Semnal analogic şi semnal digital

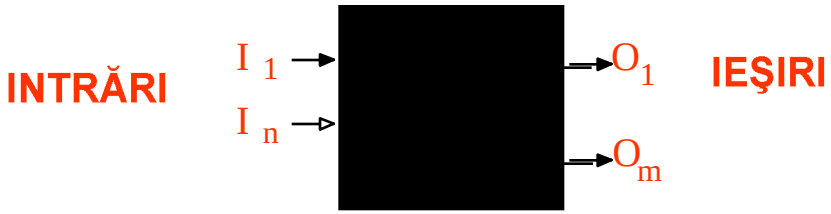
- Semnal analogic
  - continuu
    - dacă a luat valorile a şi b, atunci a luat şi "toate" valorile din [a,b]
- Semnal digital
  - are câteva niveluri – valori – distincte şi stabile
  - discontinuu
  - nu ia alte valori (…)
- Indiferent de fenomenul fizic aflat la bază
- Calculator: semnal digital cu 2 niveluri
    - "0" şi "1" …
- **Modem**: comunicare digital – analogic
    - Conducerea proceselor industriale
    - Folosirea reţelelor telefonice analogice

Aplicaţii ale circuitelor digitale

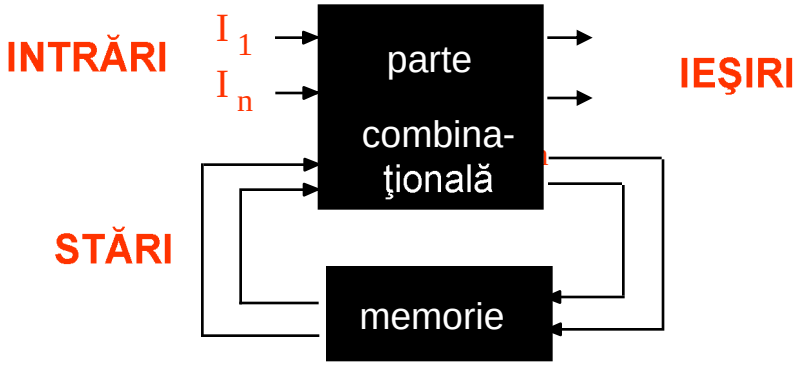
- Calculatoare
  - CPU, memorie, bus, periferice, …
- Reţele, comunicaţii
  - telefoane, modemuri, routere
- Bunuri diverse
  - automobile, audio-video, jucării, …
- Echipament pentru (alte) activităţi ştiinţifice
  - Testare, măsurare etc.
- Lumea calculului se extinde mult dincolo de PC
  - embedded systems
  - definiţii restrânse ale calculului

Circuite

- Combinaţionale:



- Secvenţiale:



Metoda cutiei negre

- Pentru un circuit combinaţional a cărui structură / funcţionare nu este cunoscută:
  - se aplică fiecare combinaţie posibilă de valori ale intrărilor
  - se observă valorile ieşirilor pentru fiecare astfel de combinaţie
  - se obţine astfel un tabel de adevăr
- Cum unui tabel de adevăr îi corespunde o funcţie booleană, rezultă că fiecărui circuit combinaţional îi corespunde o funcţie booleană

De la circuite la funcţii booleene

- Circuit combinaţional → **tabel de adevăr**

I_1	…….	I_n	O_1	…….	O_m
0	0….0	0	?	?….?	?
0	0….0	1	?	?….?	?
…….	…….	…….	…….	…….	…….
1	1….1	1	?	?….?	?

- **n** fixat: partea de intrare e întotdeauna aceeaşi
- Întotdeauna aceeaşi corespondenţă pentru un circuit (determinist)

II.2. FUNCȚII BOOLEENE

Structura algebrică

- Mulţimea nevidă B
- Mulţimea de operaţii binare { + , • }
- O operaţie unară { $\bar{\ }$ }
- B conţine cel puţin două elemente,

  a, b, $a \neq b$
- închidere: a + b este în B

  a • b este în B

  $\bar{a}$ este în B

Funcţii booleene

- $B = \{0,1\}$
- $f : B^n \rightarrow B^m$
  - funcţie: n **variabile**, m **valori**
  - circuit: n **intrări**, m **ieşiri**
- Există $(2^m)^{2^n}$ astfel de funcţii
- **n=1, m=1**: 4 funcţii unare cu o valoare

x	$f_0(x) = \mathbf{0}$	$f_1(x) = \mathbf{x}$	$f_2(x) = \overline{\mathbf{x}}$	$f_3(x) = \mathbf{1}$
0	0	0	1	1
1	0	1	0	1

Funcţii booleene de două variabile

- 16 funcţii booleene complet definite de 2 variabile şi cu o valoare

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

0 **X and Y** **X** **Y** **X xor Y** **X or Y** **X nor Y** **X = Y** **not Y** **not X** **X nand Y** **1**

Axiome şi teoreme în algebra booleană

• Identitate	$1.\ X+0=X$	$1D.\ X\cdot 1=X$
• Constante	$2.\ X+1=1$	$2D.\ X\cdot 0=0$
• Idempotenţă	$3.\ X+X=X$	$3D.\ X\cdot X=X$
• Involuţie	$4.\ \overline{\overline{X}}=X$	
• Complementaritate	$5.\ X+\overline{X}=1$	$5D.\ X\cdot\overline{X}=0$
• Comutativitate	$6.\ X+Y=Y+X$	$6D.\ X\cdot Y=Y\cdot X$
• Associativitate	$7.\ (X+Y)+Z=X+(Y+Z)$	
	$7D.\ (X\cdot Y)\cdot Z=X\cdot(Y\cdot Z)$	
• Distributivitate	$8.\ X\cdot(Y+Z)=(X\cdot Y)+(X\cdot Z)$	
	$8D.\ X+(Y\cdot Z)=(X+Y)\cdot(X+Z)$	
• Unificare	$9.\ X\cdot Y+X\cdot\overline{Y}=X$	$9D.\ (X+Y)\cdot(X+\overline{Y})=X$
• Absorbţie	$10.\ X+X\cdot Y=X$	$10D.\ X\cdot(X+Y)=X$
	$11.\ (X+\overline{Y})\cdot Y=X\cdot Y$	$11D.\ (X\cdot\overline{Y})+Y=X+Y$

Legile lui De Morgan

- 12. $\overline{X + Y + \ldots} = \overline{X} \cdot \overline{Y} \cdot \ldots$
- 12D. $\overline{X \cdot Y \cdot \ldots} = \overline{X} + \overline{Y} + \ldots$
- Generalizare:
- 13. $\overline{f(X_1, \ldots X_n, 0, 1, +, \cdot)} = f(\overline{X_1}, \ldots \overline{X_n}, 1, 0, \cdot, +)$
- Dualitate

Forme normale

- Plecând de la tabelul de adevăr al unei funcţii, se pot obţine două expresii diferite pentru acea funcţie
- Forma normală disjunctivă
  - Pentru fiecare 1 din ultima coloană, se scrie un termen ce conţine doar conjuncţii
  - Termenii se leagă prin disjuncţie
  - Fiecare termen conţine fiecare variabilă a funcţiei
    - negată, dacă în linia acelui 1 variabila apare cu valoarea 0
    - nenegată pentru 1
    - Exemplu: $F_9(x, y) = \overline{x} \cdot \overline{y} + x \cdot y$
- Forma normală conjunctivă: dual
    - Exemplu: $F_9(x, y) = (x + y) \cdot (\overline{x} + \overline{y})$

Operaţiile calculatorului la nivel **logic** elementar

- Pentru calculatoarele actuale, operaţiile elementare la nivelul circuitelor de bază sunt operaţiile logicii booleene
  - care simulează şi operaţiile aritmetice elementare în baza 2
- Un circuit combinaţional poate fi văzut ca implementând o funcţie booleană

II.3. DIAGRAME LOGICE

Alfabetul diagramelor logice

- Porţi elementare
  - AND
  - OR
  - NOT

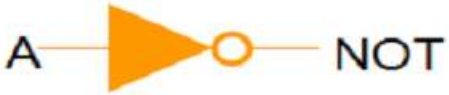
- Funcţionarea unei porţi se poate descrie printr-un tabel de adevăr (funcţia booleană ataşată)

Poarta AND

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

Poarta OR

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

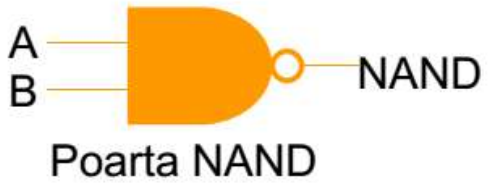


Poarta NOT

A	NOT
0	1
1	0

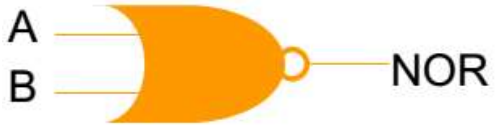
Alfabetul diagramelor logice

- Alte porţi utile
  - NAND
  - NOR
  - XOR
- NAND = NOT ◦ AND
- NOR = NOT ◦ OR
- XOR implementează funcţia **sau-exclusiv**
- Porţile NAND şi NOR necesită doar 2 tranzistori
  - pentru AND şi OR e nevoie de câte 3 tranzistori



Poarta NAND

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0



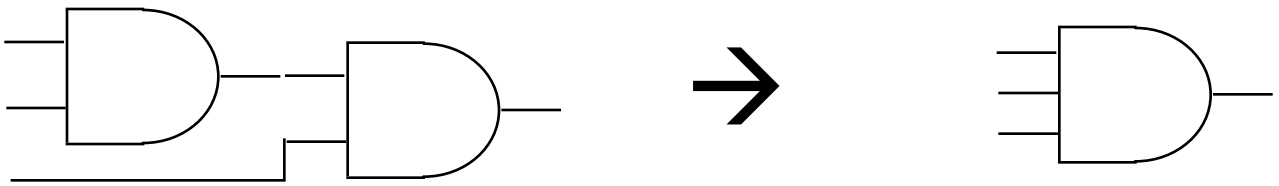
Poarta NOR

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Poarta XOR

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

- În diagrame logice apar şi porţi "elementare" cu mai multe intrări
- Operaţiile binare **asociative** pot fi extinse la operaţii cu orice număr finit de operanzi

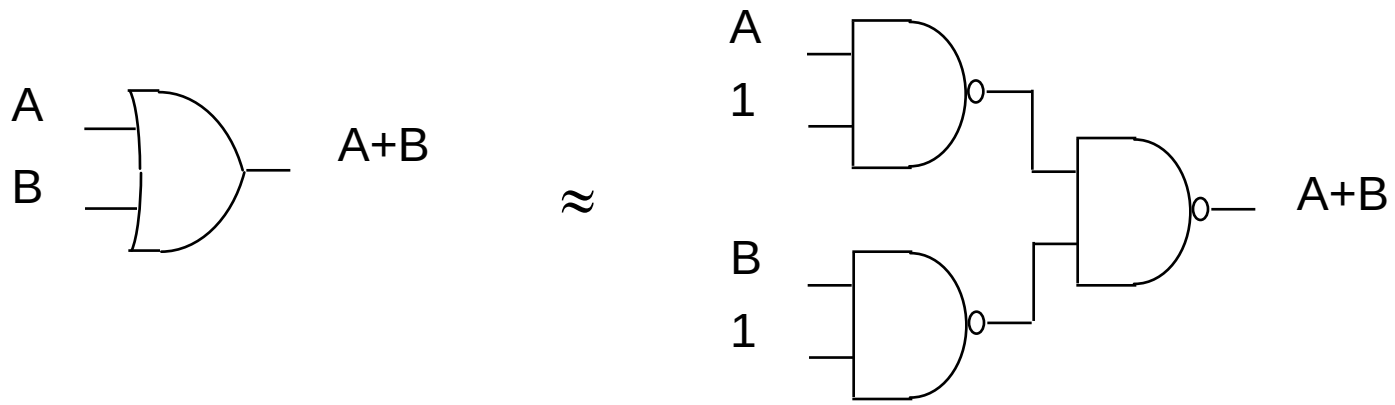


Set minimal de generatori

- Care este numărul minim de **tipuri** de porţi ce ar trebui produse pentru a putea obţine din ele circuite care implementează orice funcţie booleană?
- 3 – anume {NOT, OR, AND} – este un răspuns parţial
  - forme normale (disjunctivă, conjunctivă)
- Şi două ajung: NOT şi una din celelalte două
- Răspunsul corect este 1, cu două soluţii: {NAND} şi {NOR}

Temă

- Arătaţi că {NAND} şi {NOR} sunt mulţimi de generatori pentru funcţiile booleene
  - Indicaţie: se foloseşte FND sau FNC
  - În particular:



II.4. IMPLEMENTAREA CIRCUITELOR PRIN FUNCŢII BOOLEENE

Definirea funcţiilor booleene

- Funcţiile logice booleene pot fi definite în mai multe moduri:
  - prin tabel de adevăr
  - prin expresii conţinând variabile şi operaţii logice
  - în formă grafică
  - în sigma-notaţie (Σ)
- Exemplu: funcţia "majoritatea dintre **k**"
  - are **k** variabile şi o valoare
  - valoarea funcţiei este 1 dacă majoritatea variabilelor au valoarea 1
  - vom studia funcţia pentru **k**=3

Σ-notaţia

- $f=\sum(3,5,6,7)$
  - Fiecare număr din paranteză reprezintă un termen
  - $\sum$ denotă disjuncţia termenilor
  - Numărul de variabile **n** este cea mai mică putere a lui 2 strict mai mare decât cel mai mare număr ce apare în paranteză
    - **n** = 3: $4 = 2^2 < 7 < 2^3 = 8$
- $f(x_1,x_2,x_3) = \sum(3,5,6,7)$

Σ-notaţia

- Fiecare număr din paranteză se scrie în baza 2 pe **n** poziţii
  - 3 → 011
- Termenul corespunzând unui număr conţine:
  - toate variabilele,
  - fiecare negată dacă îi corespunde un 0 şi nenegată pentru 1,
  - legate prin conjuncţie
  - 3 → 011 → $\bar{x}_1 \cdot x_2 \cdot x_3$

Σ-notaţia

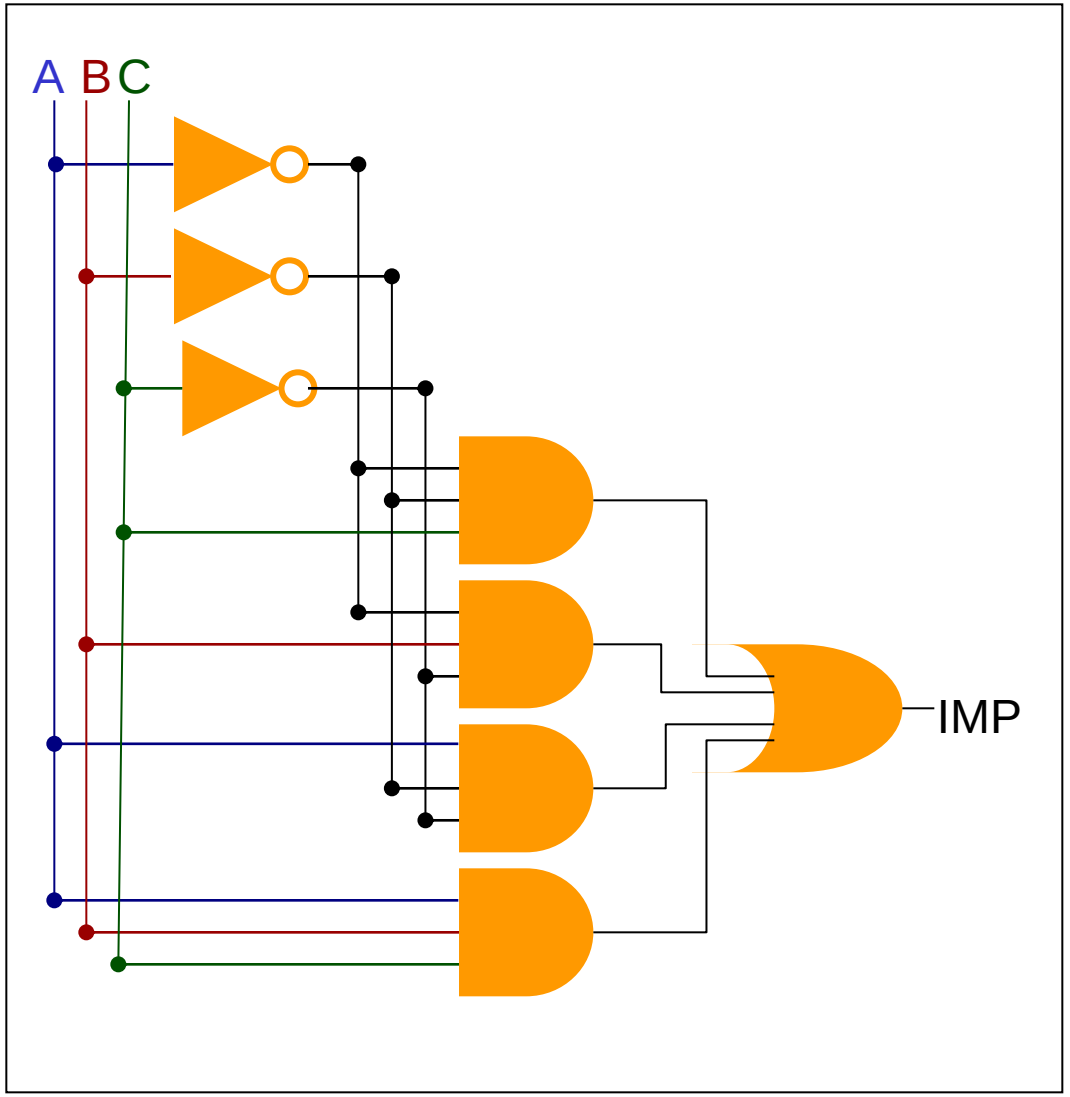
$$f(A,B,C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

- 11 aplicări ale funcţiilor elementare OR / AND
- Găsirea unei expresii echivalente (**aceeaşi funcţie booleană**) cu mai puţine aplicări de operatori ar face respectivul circuit
  - mai **rapid**
  - mai **ieftin**
  - mai **fiabil**

Funcţia "imparitate" cu 3 intrări

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

- Implementarea **formei normale disjunctive**



Implementarea funcţiei "majoritate din 3"

A	B	C	F
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

- Forma normală disjunctivă: patru termeni, fiecare conţinând doar conjuncţii
  - corespund celor 4 linii cu ieşirea 1
- În fiecare termen, o variabilă apare negată numai dacă valoarea sa pe acea linie este 0

$$F = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

- Se poate implementa o expresie echivalentă mai simplă?

Minimizare

- Nu doar set minimal de operatori, ci şi – pentru o funcţie dată – număr minimal de aplicări ale acestora (AND / OR)
- FND – un număr de apariţii ale operatorilor
  - 11 OR / AND în exemplu
- Proceduri de minimizare – reduc expresia
  - Rescriere echivalentă
  - Inducţie perfectă
  - Metoda Veitch-Karnaugh
  - Metoda Quine-McCluskey
- Hibridizare (ex: V-K urmat de distributivitate)

Minimizare prin rescriere algebrică

- Funcţia "majoritate din 3"

$$\overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot C + A\cdot B\cdot\overline{C} + A\cdot B\cdot C =$$

Idempotenţă

$$\overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot C + A\cdot B\cdot\overline{C} + A\cdot B\cdot C + A\cdot B\cdot C + A\cdot B\cdot C$$

- Expresia devine:

$$B\,C + A\,C + A\,B$$

- Metodă dificilă pentru expresii complexe