

Facultatea de
Informatică - Iași

**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
- **condiția pentru susținerea testelor scrise**
 - prezența la laborator
 - cel mult 2 absențe 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 combinaționale și funcții booleene
- III. Circuite secvențiale și automate
- IV. Reprezentări interne
- V. Arhitectura și organizarea calculatorului

Capitolul I

Introducere

I.1. EVOLUȚIE

Când au apărut mașinile de calcul?

- După fiecare redefinire a noțiunii de **calcul**
 - **abac**: adunări
 - **roți dințate**: Leibniz (2) și Pascal (10): adunări, înmulțiri
 - **Babbage**: instrucțiuni din exterior, calcul ramificat
 - **von Neumann**: program memorat; execuție în secvență de instrucțiuni; ierarhii de memorii
 - **calculatoare "paralele"** (de fapt, calcul paralel)
 - **calcul(atoare) probabilist(e), neuronal(e), evolutiv(e), cuantic(e) ...**
- Încearcă automatizarea calculului în înțelesul respectiv

Mașini de calcul universale

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

Scurtă istorie a ideilor

- inventarea scrierii poziționale 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 înțelesul de astăzi

Scurtă istorie

invenții abstracte și concrete

- 1850: George Boole inventează algebra booleană
 - propoziții logice sunt transformate în simbol
 - calcule cu propoziții logice, folosind reguli de tip matematic
- 1938: Claude Shannon leagă algebra booleană de circuite (comutatoare)
 - în teza sa de dizertație
- 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 înmulțiri pe secundă
- 1947: Shockley, Brittain și 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 științe depind într-un fel sau altul de experiment sau de observații în lumea concretă
 - unele științe sunt – sau au capitole – preponderent empirice (medicina)
- **Repetabilitatea**, inherentă noțiunii de experiment, duce la ideea de *legi empirice*: adevăruri valabile de cele mai multe ori, conform observațiilor
- 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 execuție al unui program este utilizat pentru 10% din instrucțiuni
 - legea lui Amdahl: eficiența maximă în îmbunătățirea unui sistem (concret sau abstract) se atinge dacă se optimizează subsistemul cel mai folosit
 - legile localizării – spațială, temporală

Legea lui Amdahl

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

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

- Pentru creșteri semnificative, trebuie ca **a** și **f_a** să fie cât mai mari
 - **Exemplu:** Dacă procesorul lucrează 50% din timp și devine de două ori mai rapid, atunci $A = 4/3$
 - **Observație:** Sumatorul va fi cel mai intens îmbunătățit

Legile localizării (pentru instrucțiuni)

- Localizare spațială:
 - dacă la un moment dat se execută o instrucțiune i , atunci este probabil ca la momente apropiate să se execute instrucțiuni din apropierea lui i (în ordine fizică)
 - spațiul de memorie, în care se află instrucțiunile
- Localizare temporală:
 - dacă la un moment dat se execută instrucțiunea i , atunci este probabil ca instrucțiunea i să se execute la momente apropiate în timp

Ordine fizică și ordine logică

- Instrucțiunile de executat se află în memorie într-o anumită ordine: ordinea fizică
- Ele se citesc din memorie și se execută
 - regula: în ordinea în care sunt memorate
 - excepția: sărind peste un număr de instrucțiuni
- Rezultă ordinea logică a instrucțiunilor
 - În ordinea logică la o anumită rulare, fiecare instrucțiune poate să apară de 0, 1, 2, ... ori
- Legile localizării indică un tip esențial de relație între ordinea fizică și ordinea logică:
 - **Secvența de instrucțiuni din ordinea logică este alcătuită în general din sub-secvențe construite din porțiuni restrânse ale secvenței de instrucțiuni din ordinea fizică.**

- Legile localizării: în fiecare interval "mic" de timp, programul utilizează o mică porțiune din spațiul de adrese
- Valabil pentru instrucțiuni ca și pentru date
 - Temporală: iterații (program), reutilizări
 - Spațială: ordinea fizică; tablouri de date
- De mai bine de 20 de ani, soluțiile hardware se bazează pe legile localității pentru a câștiga viteză
- Un exemplu: ierarhia de memorii

I.3. IERARHIA DE MEMORII

Ierarhia de memorii

- Soluționează două probleme:
 1. Programele foarte mari nu încap în memoria principală
 - Nivelul inferior al ierarhiei (memoria secundară)
 2. Necesitatea creșterii vitezei de execuție a programelor
 - Nivel superior al ierarhiei (memoria cache)

Ierarhia de memorii

- Niveluri:
 1. regiștrii-generalii – procesor ($\leq 1/4$ Ko)
 2. memoria cache (≤ 1 Mo)
 3. memoria RAM (≤ 4 Go)
 4. memoria secundară – disc (≤ 1 To)
 - Prețul pe bit scade pe măsura depărtării de procesor
- **Timp de comunicare și capacitate de memorare** foarte mici pe primele două niveluri
 - corespunzând "ferestrelor" din legile localizării
- **Dar comparativ foarte mari pe ultimul**
 - permițând memorarea programelor foarte mari

Nivelurile ierarhiei de memorii

Capacitate
Timp de acces
Cost

Regiștrii CPU

4-64 octeți
 1 ns

Memorie Cache

Nivel 1: <16 Kilo-octeți
 Nivel 2: < 2Mega-octeți
 3 ns; 15 ns
 0.1 cenți/bit

Memorie principală

< 4Giga-octeți
 150 ns
 .01-.001 cenți/ bit

Disc

100 Giga-octeți
 5 ms
 (5,000,000 ns)
 10^{-5} - 10^{-6} cenți/bit

Bandă

secunde-minute
 10^{-8} cenți/bit

Regiștri

Instr. / operanzi

Cache

Blocuri

Memorie principală

Pagini

Disc

Fișiere

? bandă

Controlul transferului
Unitatea de transfer

Program utilizator
 1-8 octeți

Hardware
 (Controller cache)
 8-128 octeți

Sistemul de operare
 512 - 4 Kilo-octeți

Utilizator
 Mega-octeți

Nivel superior

Viteză
 mai mare

Capacitate
 mai mare

Nivel inferior

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
- Execuția – direct din cache
 - succes: instrucțiunea următoare este găsită în cache
 - eșec: instrucțiunea următoare trebuie adusă de pe un nivel inferior
 - eșecuri dese ar cauza o execuție mai lentă decât dacă instrucțiunile ar fi aduse direct din memoria secundară
- Legile localizării fac ca ierarhiile de memorii să accelereze execuția programelor
 - pe porțiuni
 - o valoare uzuală a ratei de succes poate fi de ordinul a 95%

Capitolul al II-lea

Circuite combinaționale și funcții booleene

II.1. INTRODUCERE

Semnal analogic și semnal digital

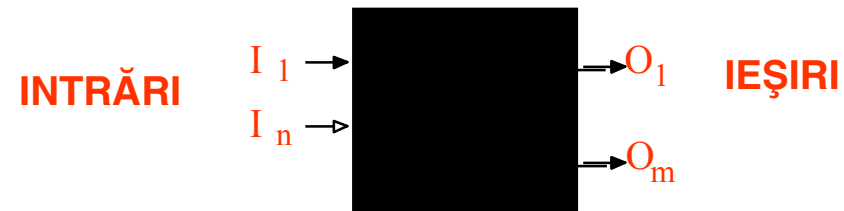
- Semnal analogic
 - continuu
 - dacă a luat valorile a și b , atunci a luat și "toate" valorile din $[a,b]$
- Semnal digital
 - are câteva niveluri – valori – distincte și stabile
 - discontinuu
 - nu ia alte valori (...)
- Indiferent de fenomenul fizic aflat la bază
- Calculator: semnal digital cu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

Aplicații ale circuitelor digitale

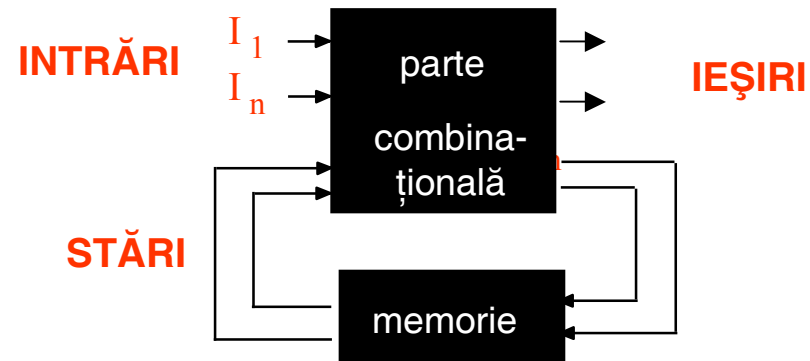
- Calculatoare
 - CPU, memorie, bus, periferice, ...
- Rețele, comunicații
 - telefoane, modemuri, routere
- Bunuri diverse
 - automobile, audio-video, jucării, ...
- Echipament pentru (alte) activități științifice
 - Testare, măsurare etc.
- Lumea calculului se extinde mult dincolo de PC
 - embedded systems
 - definiții restrânse ale calculului

Circuite

- Combinaționale:



- Secvențiale:



Metoda cutiei negre

- Pentru un circuit combinațional a cărui structură / funcționare nu este cunoscută:
 - se aplică fiecare combinație posibilă de valori ale intrărilor
 - se observă valorile ieșirilor pentru fiecare astfel de combinație
 - se obține astfel un **tabel de adevăr**
- Cum unui tabel de adevăr îi corespunde o funcție booleană, rezultă că **fiecărui circuit combinațional îi corespunde o funcție booleană**

De la circuite la funcții booleene

- Circuit combinațional \rightarrow 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 aceeași
- Întotdeauna aceeași corespondență pentru un circuit (determinist)

II.2. FUNCȚII BOOLEENE

Structura algebrică

- Mulțimea nevidă B
- Mulțimea de operații binare $\{ + , \cdot \}$
- O operație unară $\{ \bar{} \}$
- B conține cel puțin două elemente,
 $a, b, a \neq b$
- închidere:
 $a + b$ este în B
 $a \cdot b$ este în B
 \bar{a} este în B

Funcții booleene

- $B = \{0,1\}$
- $f : B^n \rightarrow B^m$
 - funcție: n **variabile**, m **valori**
 - circuit: n **intrări**, m **ieșiri**
- Există $(2^m)^{2^n}$ astfel de funcții
- **n=1, m=1**: 4 funcții 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

Funcții booleene de două variabile

- 16 funcții booleene complet definite de 2 variabile și 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

X and Y

Y

X xor Y

X or Y

X nor Y

X = Y

not Y

not X

X nand Y

1

Axiome și teoreme în algebra booleană

• Identitate	1. $X + 0 = X$	1D. $X \cdot 1 = X$
• Constante	2. $X + 1 = 1$	2D. $X \cdot 0 = 0$
• Idempotență	3. $X + X = X$	3D. $X \cdot X = X$
• Involuție	4. $\overline{\overline{X}} = X$	
• Complementaritate	5. $X + \overline{X} = 1$	5D. $X \cdot \overline{X} = 0$
• Comutativitate	6. $X + Y = Y + X$	6D. $X \cdot Y = Y \cdot X$
• Asociativitate	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. $(X + Y) \cdot (X + \overline{Y}) = X$
• Absorbție	10. $X + X \cdot Y = X$ 11. $(X + \overline{Y}) \cdot Y = X \cdot Y$	10D. $X \cdot (X + Y) = X$ 11D. $(X \cdot \overline{Y}) + Y = X + Y$

Legile lui De Morgan

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

Forme normale

- Plecând de la tabelul de adevăr al unei funcții, se pot obține două expresii diferite pentru acea funcție
- Forma normală disjunctivă
 - Pentru fiecare 1 din ultima coloană, se scrie un termen ce conține doar conjuncții
 - Termenii se leagă prin disjuncție
 - Fiecare termen conține fiecare variabilă a funcției
 - negată, dacă în linia aceluiași 1 variabila apare cu valoarea 0
 - nenegată pentru 1
 - Exemplu: $F_9(x, y) = \bar{x} \cdot \bar{y} + x \cdot y$
- Forma normală conjunctivă: dual
 - Exemplu: $F_9(x, y) = (x + y) \cdot (\bar{x} + \bar{y})$

Operațiile calculatorului la nivel **logic** elementar

- Pentru calculatoarele actuale, operațiile elementare la nivelul circuitelor de bază sunt **operațiile logicii booleene**
 - care simulează și operațiile aritmetice elementare în baza 2
- Un circuit combinațional poate fi văzut ca implementând o funcție booleană

II.3. DIAGRAME LOGICE

Alfabetul diagramelor logice

- Porti elementare

- AND
- OR
- NOT



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



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

- Funcționarea unei porți se poate descrie printr-un tabel de adevăr (funcția booleană atașată)



A	NOT
0	1
1	0

Alfabetul diagramelor logice

- Alte porți utile
 - NAND
 - NOR
 - XOR
- $\text{NAND} = \text{NOT} \circ \text{AND}$
- $\text{NOR} = \text{NOT} \circ \text{OR}$
- XOR implementează funcția **sau-exclusiv**
- Porțile NAND și NOR necesită doar 2 tranzistori
 - pentru AND și OR e nevoie de câte 3 tranzistori



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

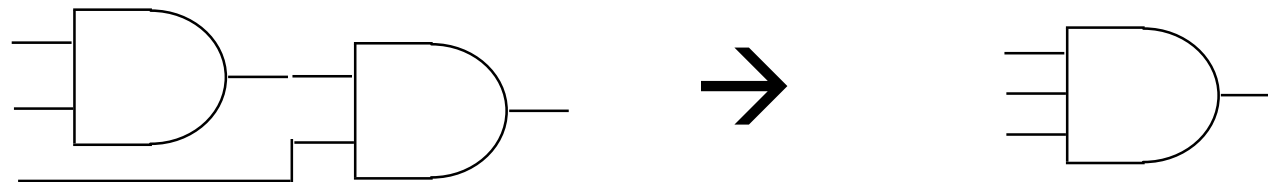


A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0



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

- În diagrame logice apar și porți "elementare" cu mai multe intrări
- Operațiile binare **asociative** pot fi extinse la operații cu orice număr finit de operanzi

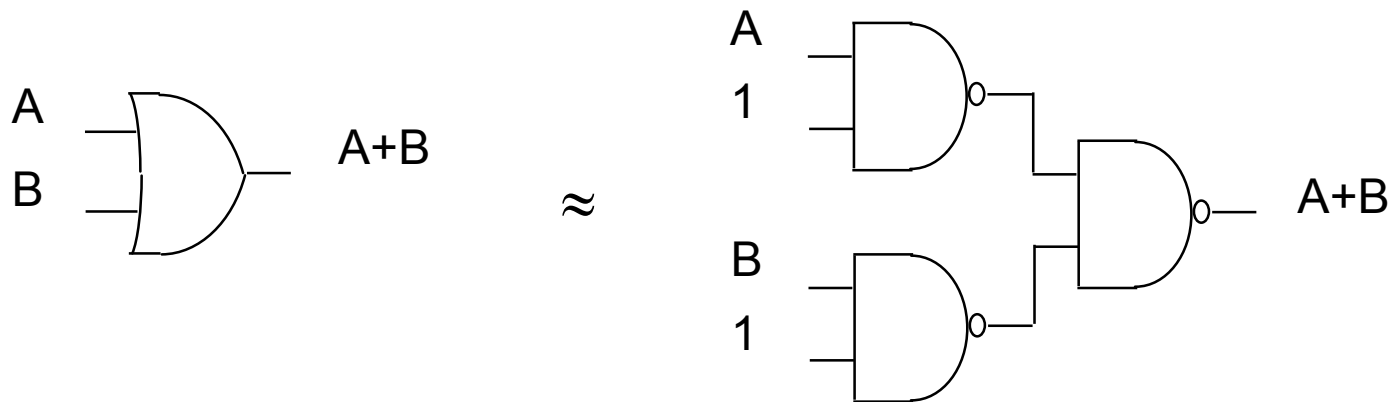


Set minimal de generatori

- Care este numărul minim de **tipuri** de porți ce ar trebui produse pentru a putea obține din ele circuite care implementează orice funcție booleană?
- 3 – anume {NOT, OR, AND} – este un răspuns parțial
 - forme normale (disjunctivă, conjunctivă)
- Și două ajung: NOT și una din celelalte două
- Răspunsul corect este 1, cu două soluții: {NAND} și {NOR}

Temă

- Arătați că {NAND} și {NOR} sunt mulțimi de generatori pentru funcțiile booleene
 - Indicație: se folosește FND sau FNC
 - În particular:



II.4. IMPLEMENTAREA CIRCUITELOR PRIN FUNCȚII BOOLEENE

Definirea funcțiilor booleene

- Funcțiile logice booleene pot fi definite în mai multe moduri:
 - prin tabel de adevăr
 - prin expresii conținând variabile și operații logice
 - în formă grafică
 - în sigma-notație (Σ)
- Exemplu: funcția "majoritatea dintre k "
 - are k variabile și o valoare
 - valoarea funcției este 1 dacă majoritatea variabilelor au valoarea 1
 - vom studia funcția pentru $k=3$

Σ -notația

- $f = \Sigma(3, 5, 6, 7)$
 - Fiecare număr din paranteză reprezintă un termen
 - Σ denotă disjuncția 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: $4 = 2^2 < 7 < 2^3 = 8$
- $f(x_1, x_2, x_3) = \Sigma(3, 5, 6, 7)$

Σ -notația

- Fiecare număr din paranteză se scrie în baza 2 pe n poziții
 - $3 \rightarrow 011$
- Termenul corespunzând unui număr conține:
 - toate variabilele,
 - fiecare negată dacă îi corespunde un 0 și nenegată pentru 1,
 - legate prin conjuncție
 - $3 \rightarrow 011 \rightarrow \bar{x}_1 \cdot x_2 \cdot x_3$

Σ -notația

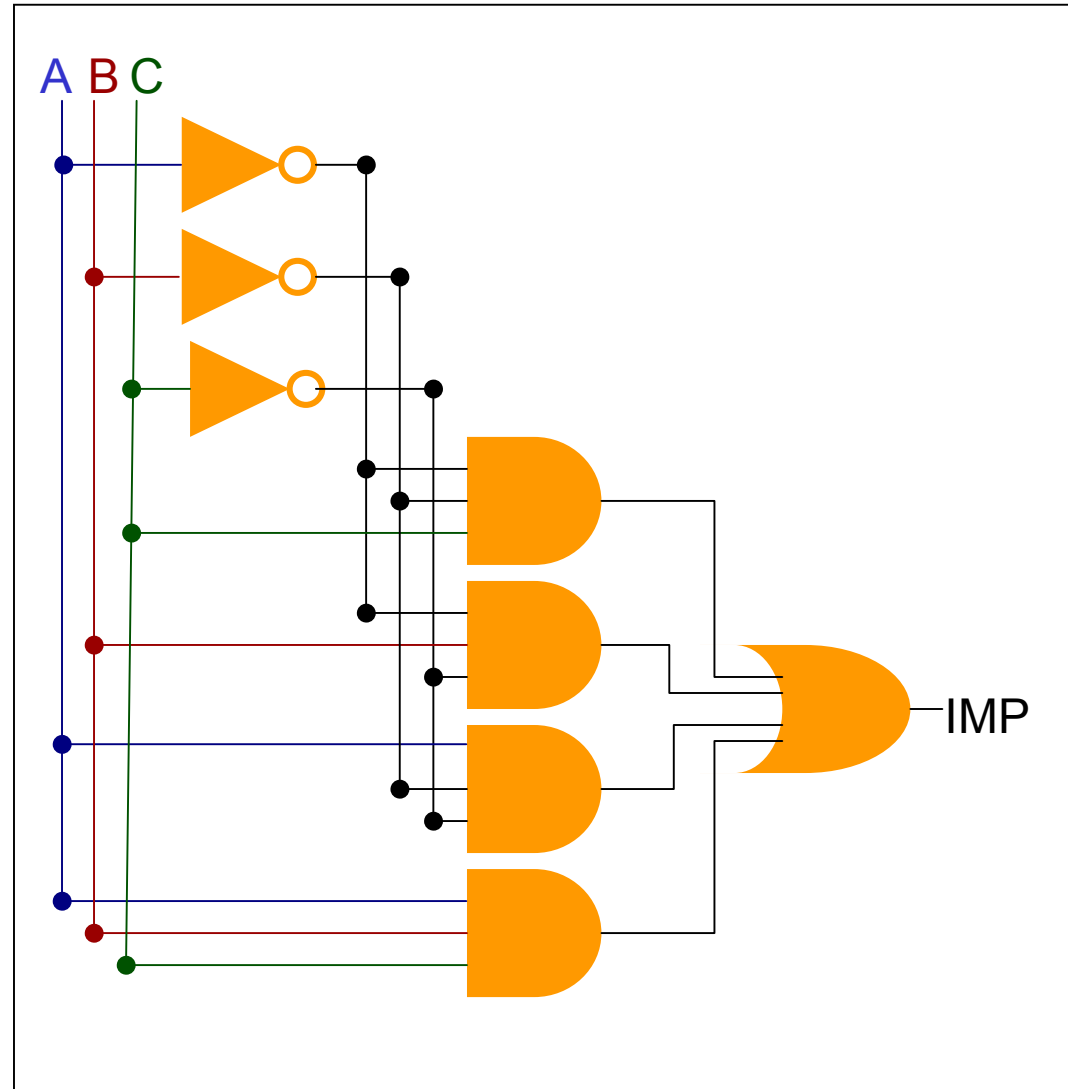
$$f(A,B,C) = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C$$

- 11 aplicări ale funcțiilor elementare OR / AND
- Găsirea unei expresii echivalente (**aceeași funcție booleană**) cu mai puține aplicări de operatori ar face respectivul circuit
 - mai **rapid**
 - mai **ieftin**
 - mai **fiabil**

Funcția "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 **forme normale disjunctive**



Implementarea funcției "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 conținând doar conjuncții
 - corespund celor 4 linii cu ieșirea 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 și – pentru o funcție dată – număr minimal de aplicări ale acestora (AND / OR)
- FND – un număr de apariții ale operatorilor
 - 11 OR / AND în exemplu
- Proceduri de minimizare – reduc expresia
 - Rescriere echivalentă
 - Inducție perfectă
 - Metoda Veitch-Karnaugh
 - Metoda Quine-McCluskey
- Hibridizare (ex: V-K urmat de distributivitate)

Minimizare prin rescriere algebrică

- Funcția "majoritate din 3"

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

Idempotență



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

- Expresia devine:

$$B C + A C + A B$$

- Metodă dificilă pentru expresii complexe