

# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x00**

**INFORMAȚII ADMINISTRATIVE**

Cristian Rusu

# CUPRINS

- **cadre didactice**
- **organizare**
- **evaluare**
- **structura cursului**
- **obiectivele cursului**
- **referințe bibliografice generale**

# CADRE DIDACTICE

- **Cristian Rusu**
  - curs și seminar
  - contact: [cristian.rusu@unibuc.ro](mailto:cristian.rusu@unibuc.ro)
  - pagină web curs: <https://cs.unibuc.ro/~crusu/asc/index.html>
- **Bogdan Macovei și Ruxandra Bălucea**
  - laborator
  - contact
    - [bogdan.macovei.fmi@gmail.com](mailto:bogdan.macovei.fmi@gmail.com)
    - [ruxandra.balucea@unibuc.ro](mailto:ruxandra.balucea@unibuc.ro)

# ORGANIZARE ȘI EVALUARE

- **organizare:**
  - 2h curs / săptămână
  - 2h seminar / 2 săptămâni
  - 2h laborator / 2 săptămâni
- **evaluare:**
  - 70% examen (online, cu materiale, ultimul curs)
    - doar examenul final se repetă la restanță
  - 30% laborator
    - 2 x 10% două teme (predare în lunile noiembrie și decembrie)
    - 10% evaluare finală la laborator (ultimul laborator)
  - 10% extra
    - proiecte (în grup) discutate la curs, vor fi 3 sau 4
- **condiții de promovare:**
  - peste 50% la laborator, pentru fiecare din cele două criterii separat
    - **atenție**, dacă acest punctaj nu este îndeplinit laboratorul trebuie refăcut în următorul an universitar
  - peste 50% la examenul final

# **ORGANIZARE ȘI EVALUARE**

- din cauza pandemiei COVID-19
- **cursul va fi integral online, prezența nu este obligatorie**
- **laboratorul va fi integral online, prezența obligatorie**
- **seminarul va fi online**
  - începem online
  - în cazul în care valul COVID-19 se termină, putem să ne vedem
  - prezența nu va fi obligatorie, vine doar cine dorește
  - cel puțin un seminar va fi online și înregistrat (pentru cei care nu vor/pot să ajungă fizic)
- **binențeles, ne supunem regulilor statului și universității**

# ORGANIZARE ȘI EVALUARE

- **pentru curs/seminar/laborator**
  - vă rog să aveți funcționale microfon/camera video (vrem să comunicăm cu voi, deci avem nevoie să vă vedem și să vă auzim)
  - în timpul activităților vă rog să țineți microfon/camera dezactivate
  - vă vom ruga noi să activați microfon/camera pentru a comunica
  - fiți pregătiți cu hartie/pix pentru a nota idei fundamentale și pentru exerciții (materialele le aveți și electronic, dar unele probleme vor fi lucrate împreună atât la curs cât și la seminar)
  - aveți nevoie de un laptop/computer pentru laborator
- **pentru examen: intră orice e prezentat la curs/seminar/laborator (cu excepția unor slide-uri/concepte care sunt explicit menționate)**

# ORGANIZARE ȘI EVALUARE

- **pentru seminar**
  - vor fi exerciții cu noțiunile pe care le întâlnim la curs
  - mereu încerc să vă ofer mai multe probleme decât putem face în cele două ore o dată la două săptămâni
  - ce rămâne, este temă
  - ce nu puteți să faceți puteți întreba la cursul/seminarul următor
  - seminarul ține pasul cu ce facem la curs
- **pentru laborator**
  - multă programare
  - Assembly x86
  - veți vedea și Linux/Git/limbajul C
  - sunt două părți: câte 3 laboratoare fiecare
  - *la curs acoperim teoria de la laborator de abia din săptămâna 6*
- **după aproximativ o lună veți putea evalua dificultatea materiei**

# ORGANIZARE ȘI EVALUARE

- nu mai sunteți la liceu
- acum nu mai repetăm același tip de probleme/exerciții
- cursul
  - accentul este pus pe concepte
  - accentul este pus pe a înțelege ce se întâmplă
  - ideea este cel mai imporant lucru
- seminar și laborator
  - exerciții cu conceptele de la curs
  - cimentăm ce am văzut la curs
  - aici e multă muncă individuală
  - vă încurajăm să lucrați împreună și să colaborați pentru a înțelege, dar **nu să copiați**

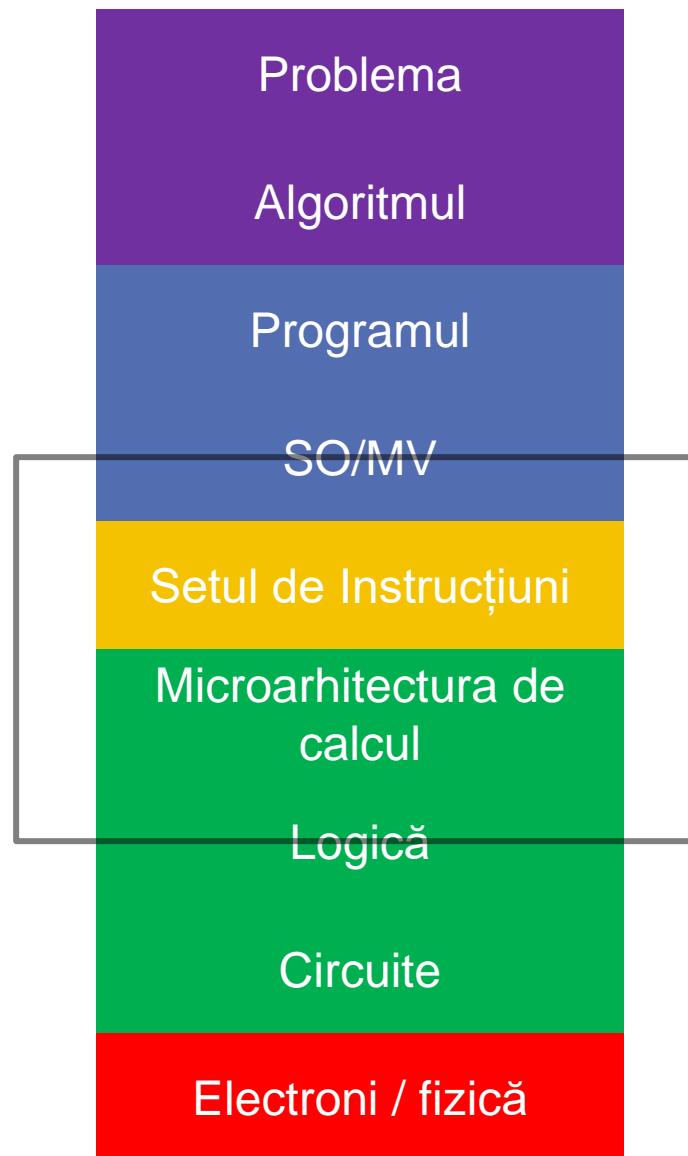
# **NU COPIAȚI/PLAGIAȚI**

- **pedeapsa pentru copiat nu este doar că veți pica materia**
- **veți fi raportați la comisia de etică și riscați exmatricularea**
- **fără copy/paste la seminar/laborator/teme/test/examen**
- **fără copiat de la colegi (riscați toți)**

# STRUCTURA CURSULUI

- **circuite digitale**
  - teoria informației și abstractizarea digitală
  - funcții și circuite logice
- **arhitecturi de calcul** ← materia acoperită la laborator începe de aici
  - seturi de instrucțiuni
  - limbajul assembly
  - compilatoare
  - pipelining
  - ierarhia memoriei
- **organizarea calculatoarelor**
  - unitatea de procesare centrală
  - performanța calculatoarelor
  - dispozitive periferice și întreruperi
  - calcul paralel
- **potențiale subiecte moderne la curs: RISC-V, WebAssembly, TockOS, hardware pentru machine learning (GPU, TPU, etc.), etc.**
- **laborator: programare în limbajul Assembly x86**

# POZIȚIONAREA CURSULUI



# OBIECTIVELE CURSULUI

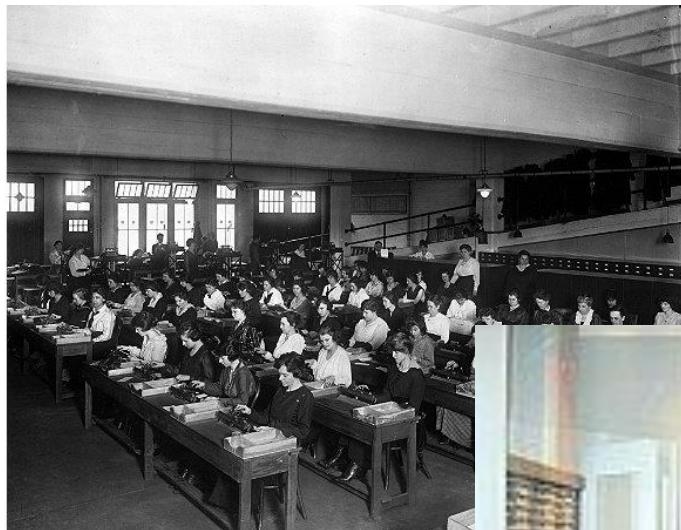
- să înțelegeți principile arhitecturii sistemelor de calcul
  - ce urmărim
  - ce limitări există
  - punerea în balanță a unor criterii de performanță contradictorii
- să înțelegeți ce există ca tehnologie acum
  - ce limitări există astăzi
  - ce execuță un computer pe care îl programezi voi
  - cum puteți optimiza execuția programelor
- să înțelegeți cum să folosiți ce ați învățat în viitor
  - design de hardware nou
- pentru a ne asigura că lucrurile merg bine, veți avea posibilitatea să oferiți feedback (anonim binențeles) la jumătatea cursului
- pe parcurs, dacă sunt probleme sau neclarități vă rog să mi le comunicați (e-mail, la clasă, fie direct sau anonim sau printr-un reprezentant, etc.)

# OBIECTIVELE CURSULUI

- **să înțelegeți principile arhitecturii sistemelor de calcul**
  - ce urmărim
  - ce limitări există
  - punerea în balanță a unor criterii de performanță contradictorii
- **să înțelegeți ce există ca tehnologie acum**
  - ce limitări există astăzi
  - ce execuță un computer pe care îl programezi voi
  - cum puteți optimiza execuția programelor
- **să înțelegeți cum să folosiți ce ați învățat în viitor**
  - design de hardware nou
- **pentru a ne asigura că lucrurile merg bine, veți avea posibilitatea să oferiți feedback (anonim binențeles) la jumătatea cursului**
- **pe parcurs, dacă sunt probleme sau neclarități vă rog să mi le comunicați (e-mail, la clasă, fie direct sau anonim sau printr-un reprezentant, etc.)**

**“The purpose of computing is insight, not numbers.” (Richard Hamming)**

# PROGRESUL TEHNOLOGIC FĂCUT



- înainte de al doilea razboi mondial, 1939, USA
- putere de calcul: 50 operații / secundă



- ENIAC
- 1947 – 1955, USA
- 1000 operații / secundă

- FUGAKU, 2020, Japonia
- core-uri: 7.630.848
- memorie: 5.087.232 GB
- 537.212 trilioane operații / secundă



<https://www.theatlantic.com/technology/archive/2013/10/computing-power-used-to-be-measured-in-kilo-girls/280633/>

<https://en.wikipedia.org/wiki/ENIAC>

<https://www.top500.org>

# CERINȚELE DE AZI (EXEMPLU)

- hardware pentru machine learning (învățare automată)
- DeepMind și OpenAI vorbesc despre două lucruri:
  - algoritmi noi pentru modelare și antrenare
  - “compute” (infrastructura hardware pe care rulează algoritmii)



- **AlphaZero: software bazat de ML**
  - Stockfish (software clasic bazat pe metode de căutare – în principal alpha/beta pruning)
    - 3500 ELO
  - AlphaZero vs Stockfish:
    - $+155 -6 = 839$
  - Magnus Carlsen (campionul mondial actual)
    - < 2900 ELO
  - AlphaZero (alb) vs Stockfish (negru)



# CERINȚELE DE AZI (EXEMPLU)

- hardware pentru machine learning (învățare automată)
- DeepMind și OpenAI vorbesc despre două lucruri:
  - algoritmi noi pentru modelare și antrenare
  - “compute” (infrastructura hardware pe care rulează algoritmii)



- **AlphaZero: software bazat de ML**
  - Stockfish (software clasic bazat pe metode de căutare – în principal alpha/beta pruning)
    - 3500 ELO
  - DeepMind spune că a antrenat acest algoritm 4 ore (învățat din self-play)
    - da, **4 ore** pe platforma lor de calcul
    - un calcul rapid, aproximativ, arată că pe laptop-ul meu aceeași procedură de antrenare ar dura **30 de ani**
    - **costul? aproximativ 20\$ milioane**
  - cum scădem costul?
    - **algoritmi mai eficienți**
    - **hardware special, dedicat**

# CERINȚELE DE AZI (EXEMPLU)

- ASC e importantă și pentru software-ul folosit de zi cu zi
- ce face următorul algoritm?

```
# varianta A
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

# CERINȚELE DE AZI (EXEMPLU)

- ASC e importantă și pentru software-ul folosit de zi cu zi
- ce face următorul algoritm?

```
# varianta A
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

- înmulțește două matrice
- care este complexitatea numerică?

# CERINȚELE DE AZI (EXEMPLU)

- ASC e importantă și pentru software-ul folosit de zi cu zi
- ce face următorul algoritm?

```
# varianta A
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

```
# varianta B
for (int j = 0; j < n; ++j)
    for (int i = 0; i < n; ++i)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

- înmulțește două matrice
- care este complexitatea numerică?
  - $O(n^3)$
  - $2n^3$  operații
- ce face varianta B?

# CERINȚELE DE AZI (EXEMPLU)

- ASC e importantă și pentru software-ul folosit de zi cu zi
- ce face următorul algoritm?

```
# varianta A
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

```
# varianta B
for (int j = 0; j < n; ++j)
    for (int i = 0; i < n; ++i)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

- înmulțește două matrice
- care este complexitatea numerică?
  - $O(n^3)$
  - $2n^3$  operații
- ce face varianta B?
- cei doi algoritmi sunt echivalenți, matematic

# CERINȚELE DE AZI (EXEMPLU)

- ASC e importantă și pentru software-ul folosit de zi cu zi
- ce face următorul algoritm?

```
# varianta A
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

```
# varianta B
for (int j = 0; j < n; ++j)
    for (int i = 0; i < n; ++i)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

- înmulțește două matrice
- care este complexitatea numerică?
  - $O(n^3)$
  - $2n^3$  operații
- ce face varianta B?
- cei doi algoritmi sunt echivalenți, matematic

care algoritm se execută mai rapid?

# CERINȚELE DE AZI (EXEMPLU)

- ASC e importantă și pentru software-ul folosit de zi cu zi
- ce face următorul algoritm?

```
# varianta A
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

```
# varianta B
for (int j = 0; j < n; ++j)
    for (int i = 0; i < n; ++i)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

- înmulțește două matrice
- care este complexitatea numerică?
  - $O(n^3)$
  - $2n^3$  operații
- ce face varianta B?
- cei doi algoritmi sunt echivalenți, matematic

pe sistemul meu de calcul varianta B este de 4 ori mai lentă decât varianta A  
cum este posibil aşa ceva?  
care este diferența dintre cele două variante?

# CERINȚELE DE AZI (EXEMPLU)

- ASC e importantă și pentru software-ul folosit de zi cu zi
- ce face următorul algoritm?

```
# varianta A
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

```
# varianta B
for (int j = 0; j < n; ++j)
    for (int i = 0; i < n; ++i)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

- înmulțește două matrice
- care este complexitatea numerică?
  - $O(n^3)$
  - $2n^3$  operații
- ce face varianta B?
- cei doi algoritmi sunt echivalenți, matematic

pe sistemul meu de calcul varianta B este de 4 ori mai lentă decât varianta A  
cum este posibil aşa ceva?  
care este diferența dintre cele două variante?  
ordinea în care se execută instrucțiunile este foarte importantă

# CERINȚELE DE AZI (EXEMPLU)

- ASC e importantă și pentru software-ul folosit de zi cu zi
- ce face următorul algoritm?

```
# varianta A
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

```
# varianta B
for (int j = 0; j < n; ++j)
    for (int i = 0; i < n; ++i)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

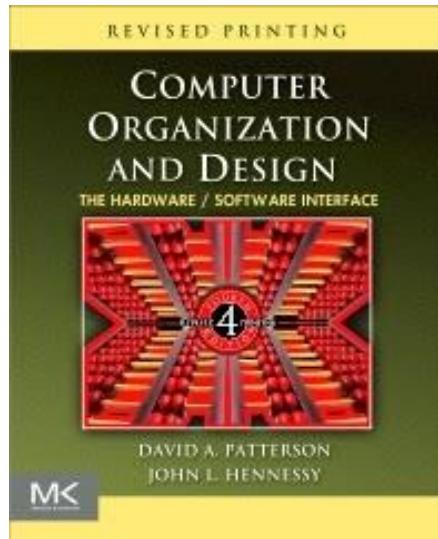
- înmulțește două matrice
- care este complexitatea numerică?
  - $O(n^3)$
  - $2n^3$  operații
- există un algoritm care calculează rezultatul  $C$  în  $O(n^{2.8074})$ 
  - din păcate, pe arhitecturile de calcul moderne, acest algoritm este mai lent decât metoda clasică (o formă de varianta A)

concluzii:

- complexitatea numerică este foarte importantă, dar nu este totul
- ce calculăm matematic și ce putem executa sunt două lucruri diferite
- câteodată progresul în lumea reală este surprinzător și complet neevident

# REFERINȚE BIBLIOGRAFICE GENERALE

- D. Patterson and J. Hennessy, Computer Organisation and Design (PH book)



- Chris Terman, 6.004 Computation Structures, MIT, 2017
- MIT Computer Architecture Group: <http://groups.csail.mit.edu/cag/>
- aceste slide-uri se bazează și pe alte cursuri de ASC la UB (R. Olimid și D. Dragulici)

La sfârșitul fiecarui curs veți avea o listă de referințe specială pentru conținutul aceluia curs

# UN DEMO PENTRU FINAL

- ce putem face dacă înțelegem arhitectura sistemelor de calcul / assembly / sisteme de operare / tehnologii web?
- Doom 3 rulat in browser (demo WebAssembly):  
<http://wasm.continuation-labs.com/d3demo/>



# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x01**

**EVOLUȚIA SISTEMELOR DE CALCUL ȘI  
SISTEMUL BINAR DE CALCUL**

Cristian Rusu

# CUPRINS

- **scurt istoric al sistemelor de calcul**
- **sistemul binar**
  - reprezentarea binară
  - reprezentarea în complement față de doi
  - funcții logice
- **referințe bibliografice**

# **SCURT ISTORIC AL SISTEMELOR DE CALCUL**

- **contribuții majore ale:**
  - Blaise Pascal
  - Gottfried Wilhelm von Leibniz
  - George Boole
  - Charles Babbage
  - Ada Lovelace
  - Konrad Zuse
  - Alan Turing
  - John von Neumann
  - Claude Shannon
  - după 1945 interestul în sisteme de calcul a crescut semnificativ iar progresul este dificil de atribuit doar unor indivizi

# **BLAISE PASCAL (1623 - 1662)**

- **în 1642, când încă nu avea 19 ani, crează Pascaline**
  - un calculator mecanic
  - capabil de adunări/scăderi (utilizat pentru calcul de taxe)
  - nu a fost o mașină practică
  - mai puțin de 50 au fost create
  - era utilizată pe post de “jucarie” pentru aristocrație
- **limbajul Pascal e numit în onoarea lui**



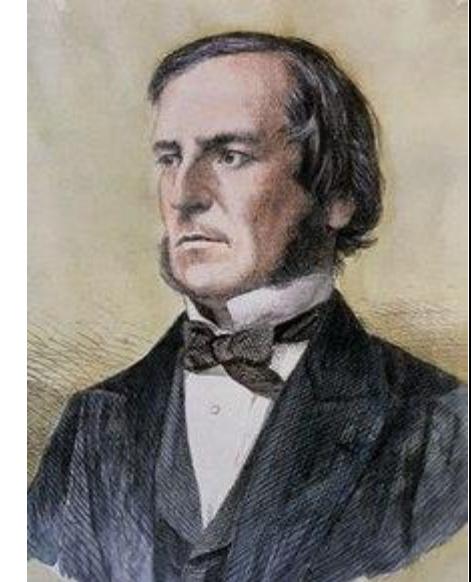
# GOTTFRIED WILHELM VON LEIBNIZ (1646 – 1716)

- **toate contribuțile lui sunt imposibil de enumerat**
- **două contribuții majore:**
  - studiază sistemul binar
  - extinde mașina lui Pascal, adăugând operațiile de înmulțire și împărțire – tot o mașină mecanică creată în 1673



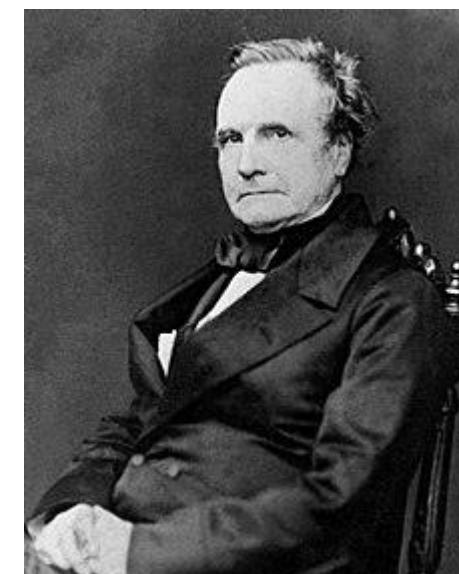
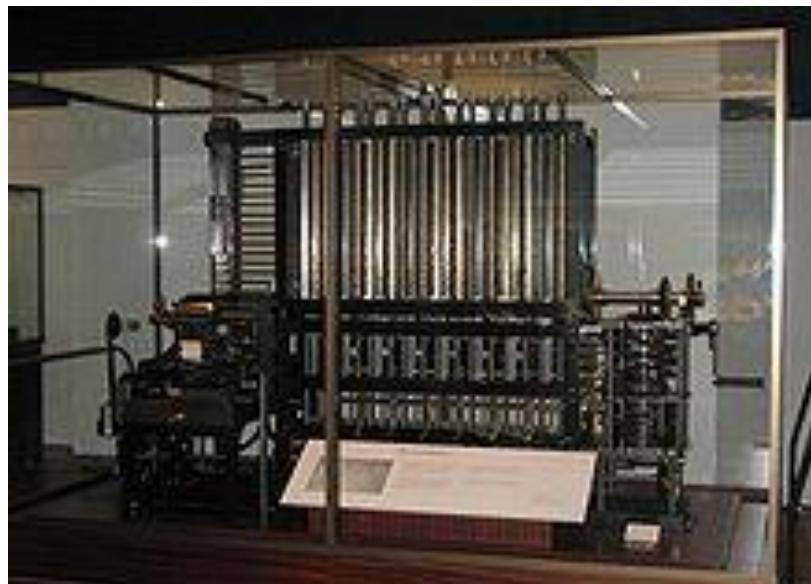
# GEORGE BOOLE (1815 – 1864)

- scrie “The Laws of Thought” (1854)
- introduce logica booleană și analizează operațiile de bază
  - negația
  - conjuncția
  - disjuncția
  - disjuncția exclusivă
- toate acestea stau la baza teoriei informației



# CHARLES BABBAGE (1791 – 1871)

- proiectează Mașina Differentială Nr. 2 (Difference Engine No. 2)
- doar teoretic, design-ul este realizat de abia în 1991
- totuși, este prima mașină de calcul (mecanică) programabilă
- prototipurile sale aveau deja peste 13 tone
- este considerat “tatăl calculatoarelor moderne”



# ADA LOVELACE (1815 – 1852)

- colaboratoare a lui Babbage
- scrie primul program, calculează numere Bernoulli
- nu existau limbaje de programare, dar ea a descris o serie de pași care să fie executăți de o mașină
- este considerată primul “programator”

Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.										Working Variables.						Result Variables.										
					Data				Working Variables.						Result Variables.										B <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>	B <sub>4</sub>	B <sub>5</sub>	B <sub>6</sub>	B <sub>7</sub>
					IV <sub>1</sub>	IV <sub>2</sub>	IV <sub>3</sub>	IV <sub>4</sub>	s <sub>V<sub>1</sub></sub>	s <sub>V<sub>2</sub></sub>	s <sub>V<sub>3</sub></sub>	s <sub>V<sub>4</sub></sub>	s <sub>V<sub>5</sub></sub>	s <sub>V<sub>6</sub></sub>	s <sub>V<sub>7</sub></sub>	s <sub>V<sub>8</sub></sub>	s <sub>V<sub>9</sub></sub>	s <sub>V<sub>10</sub></sub>	s <sub>V<sub>11</sub></sub>	s <sub>V<sub>12</sub></sub>	s <sub>V<sub>13</sub></sub>	s <sub>V<sub>14</sub></sub>	s <sub>V<sub>15</sub></sub>	s <sub>V<sub>16</sub></sub>	B <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>	B <sub>4</sub>	B <sub>5</sub>	B <sub>6</sub>	B <sub>7</sub>
1	$\times$	IV <sub>2</sub> × IV <sub>3</sub>	IV <sub>4</sub> , IV <sub>5</sub> , IV <sub>6</sub>	{IV <sub>2</sub> = IV <sub>3</sub> } {IV <sub>4</sub> = IV <sub>5</sub> }	2 n																										
2	-	IV <sub>4</sub> - IV <sub>5</sub>	IV <sub>2</sub>	{IV <sub>4</sub> = IV <sub>5</sub> }																											
3	+	IV <sub>4</sub> + IV <sub>5</sub>	IV <sub>2</sub>	{IV <sub>4</sub> = IV <sub>5</sub> }																											
4	+	IV <sub>4</sub> + IV <sub>5</sub>	IV <sub>11</sub>	{IV <sub>4</sub> = IV <sub>5</sub> }																											
5	+	IV <sub>13</sub> + IV <sub>12</sub>	IV <sub>11</sub>	{IV <sub>13</sub> = IV <sub>12</sub> }																											
6	-	IV <sub>13</sub> - IV <sub>12</sub>	IV <sub>12</sub>	{IV <sub>13</sub> = IV <sub>12</sub> }																											
7	-	IV <sub>9</sub> - IV <sub>10</sub>	IV <sub>10</sub>	{IV <sub>9</sub> = IV <sub>10</sub> }																											
8	+	IV <sub>7</sub> + IV <sub>8</sub>	IV <sub>7</sub>	{IV <sub>7</sub> = IV <sub>8</sub> }	2 + 0 = 2																										
9	+	IV <sub>8</sub> + IV <sub>7</sub>	IV <sub>11</sub>	{IV <sub>8</sub> = IV <sub>7</sub> }																											
10	×	IV <sub>12</sub> × IV <sub>13</sub>	IV <sub>12</sub>	{IV <sub>12</sub> = IV <sub>13</sub> }																											
11	+	IV <sub>12</sub> + IV <sub>13</sub>	IV <sub>12</sub>	{IV <sub>12</sub> = IV <sub>13</sub> }	1	2 n - 1																									
12	-	IV <sub>12</sub> - IV <sub>13</sub>	IV <sub>10</sub>	{IV <sub>12</sub> = IV <sub>13</sub> }																											
13	-	IV <sub>6</sub> - IV <sub>5</sub>	IV <sub>6</sub>	{IV <sub>6</sub> = IV <sub>5</sub> }	2 n - 1																										
14	+	IV <sub>4</sub> + IV <sub>2</sub>	IV <sub>7</sub>	{IV <sub>4</sub> = IV <sub>2</sub> }	2 + 1 = 3																										
15	-	IV <sub>7</sub> - IV <sub>4</sub>	IV <sub>8</sub>	{IV <sub>7</sub> = IV <sub>4</sub> }	2 n - 1																										
16	×	IV <sub>1</sub> × IV <sub>2</sub>	IV <sub>11</sub>	{IV <sub>1</sub> = IV <sub>2</sub> }																											
17	-	IV <sub>6</sub> - IV <sub>5</sub>	IV <sub>6</sub>	{IV <sub>6</sub> = IV <sub>5</sub> }	2 n - 2																										
18	+	IV <sub>1</sub> + IV <sub>2</sub>	IV <sub>7</sub>	{IV <sub>1</sub> = IV <sub>2</sub> }	3 + 1 = 4																										
19	-	IV <sub>9</sub> - IV <sub>8</sub>	IV <sub>9</sub>	{IV <sub>9</sub> = IV <sub>8</sub> }	2 n - 2																										
20	×	IV <sub>9</sub> × IV <sub>10</sub>	IV <sub>11</sub>	{IV <sub>9</sub> = IV <sub>10</sub> }	2 n - 2 n - 1 2 n - 2																										
21	×	IV <sub>12</sub> × IV <sub>13</sub>	IV <sub>12</sub>	{IV <sub>12</sub> = IV <sub>13</sub> }																											
22	+	IV <sub>12</sub> + IV <sub>13</sub>	IV <sub>12</sub>	{IV <sub>12</sub> = IV <sub>13</sub> }																											
23	-	IV <sub>12</sub> - IV <sub>13</sub>	IV <sub>10</sub>	{IV <sub>12</sub> = IV <sub>13</sub> }	n - 3 (- 1)																										
24	+	IV <sub>13</sub> + IV <sub>12</sub>	Y <sub>24</sub>	{IV <sub>13</sub> = IV <sub>12</sub> }																											
25	+	IV <sub>1</sub> + IV <sub>2</sub>	Y <sub>2</sub>	{IV <sub>1</sub> = IV <sub>2</sub> }																											

Here follows a repetition of Operations thirteen to twenty-three.



# KONRAD ZUSE (1910 – 1995)

- introduce o serie de calculatoare: Z1, Z2, Z3 si Z4
- primele prototipuri in 1940-1941
- folosește sistemul binar
- instrucțiunile sunt stocate pe o bandă
- introduce reprezentarea și calculul în virgulă mobilă
- face aproape totul în izolare (1936-1945)



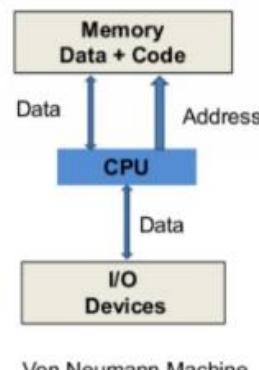
# **ALAN TURING (1912 – 1954)**

- **celebru pentru publicul larg pentru contribuția lui în spargerea rapidă a mesajelor Enigma utilizând mașina “The Bombe”**
  - practic, mașina făcea un brute-force search pentru a reduce numărul de posibilități în decriptarea mesajelor
- **introduce Mașina Turing**
  - un model teoretic pentru a implementa orice algoritm
  - conceptul de Turing-complete
    - intuiția: un sistem care poate recunoaște și analiza seturi de reguli pentru manipularea datelor (o cantitate infinită, teoretic)
- **introduce Testul Turing**
  - The imitation game: “The original question, ‘Can machines think?’ I believe to be too meaningless to deserve discussion” A. Turing

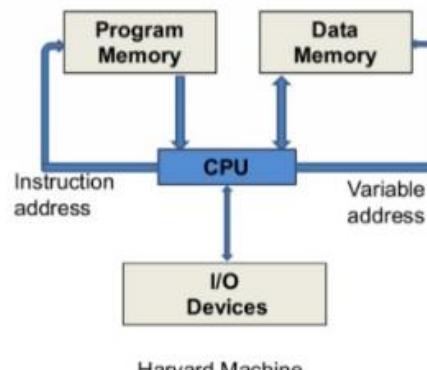


# JOHN VON NEUMANN (1903 – 1957)

- considerat unul dintre cei mai buni matematicieni ai ultimului secol, aduce contribuții în numeroase domenii
- ajută la crearea primului calculator electronic ENIAC (Electronic Numerical Integrator And Computer), 1939-1944
- îmbunătățește ENIAC ajutând la crearea EDVAC (Electronic Discrete Variable Automatic Computer), sistemul este binar și are programe stocate
- introduce arhitectura von Neumann



Von Neumann Machine

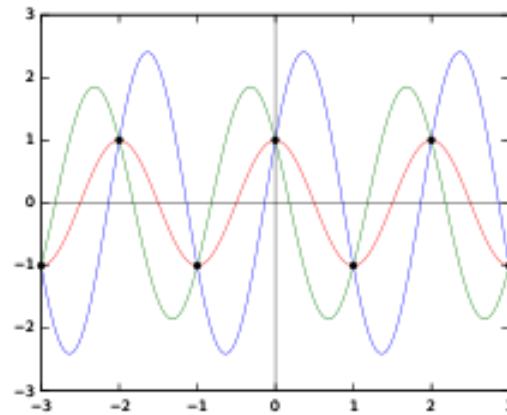


Harvard Machine



# CLAUDE SHANNON (1916 – 2001)

- considerat “părintele teoriei informației”
- trei contribuții exceptionale:
  - demonstrează faptul că probleme de logică Booleană pot fi rezolvate cu circuite electronice
  - teorema de eșantionare Shannon-Nyquist



- inventează teoria informației
  - cursul următor discutăm detaliat



# **IDEILE MARI**

- de la mecanic la electric
- de la o mașină care face un singur lucru automat, la o mașină care este programabilă
- design modular
- teorie despre ce este posibil pe aceste mașini
- dorință de a face lucrurile optim, la limită și fără risipă

# **POST SHANNON ...**

- după al doilea razboi mondial, cercetarea în domeniul calculatoare începe un ritm exponential
- sunt multe, mici inventii și discoperiri tehnologice pe parcurs
- nu avem cum să le acoperim pe toate
- actorii importanți în domeniu au devenit grupuri profesionale (ex: IEEE, ACM, etc.) și state (ex: State Unite, programe de cercetare DARPA, etc.)
- la baza acestui progres stau niște concepte de matematică fundamentale, începem cu sistemul binar

# SISTEMUL BINAR

- este baza sistemelor moderne de calcul
- orice număr (întreg sau, general, real) poate fi reprezentat printr-un număr (potential infinit) de biți
- bit = *binary digit*
- ne aflăm în sistemul de numărare cu **baza  $B = 2$**
- avem disponibile doar două cifre: **0 și 1**

# SISTEMUL BINAR

- un număr natural reprezentat în baza  $B = 2$

bit $b_i$ :	...	0	1	1	1	1	0	0	0	1
$2^i$ :	...	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

- $x = \sum_{i=0}^{N-1} b_i 2^i$
- **N** este numărul de biți pe care îl folosim în reprezentare
- În exemplul de mai sus:
  - care este numărul reprezentat?
  - pe câți biți este reprezentat acest număr?

# SISTEMUL BINAR

- un număr natural reprezentat în baza  $B = 2$

bit $b_i$ :	...	0	1	1	1	1	0	0	0	1
$2^i$ :	...	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

- $x = \sum_{i=0}^{N-1} b_i 2^i$
- **N** este numărul de biți pe care îl folosim în reprezentare
- În exemplul de mai sus:
  - $0 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 241$
  - Mai sus avem  $N = 9$ , dar defapt avem nevoie de  $N = 8$

# SISTEMUL BINAR

- intuiția noastră este în baza  $B = 10$
- este folositor să abstractizăm și să considerăm baza generală  $B$
- în baza  $B$  avem:
  - cifre de la 0 la  $B-1$  (restul se numesc numere)
  - reprezentarea este  $x = \sum_{i=0}^{N-1} b_i B^i$
  - bitul  $b_0$  se numește **Least Significant Bit (LSB)** iar bitul  $b_{N-1}$  se numește **Most Significant Bit (MSB)**
  - cum reprezentăm un număr din baza 10 în baza  $B$ ?
    - împărțiri repetitive cu  $B$  și păstrăm restul

# SISTEMUL BINAR

- un exemplu explicit:  $(4215)_{10} = (1000001110111)_2$

2	4215		
2	2107	— 1	← LSB
2	1053	— 1	
2	526	— 1	
2	263	— 0	
2	131	— 1	
2	65	— 1	
2	32	— 1	
2	16	— 0	
2	8	— 0	
2	4	— 0	
2	2	— 0	
2	1	— 0	
	0	— 1	← MSB

# SISTEMUL BINAR

- conversia între sisteme de numărare este foarte folositoare
- ce se întâmplă în baza  $B = 10$ ?
  - ce se întâmplă dacă vrem să trecem din baza  $B_{old} = 10$  în noua bază  $B_{new} = 100$ ?
    - câte cifre sunt în baza  $B_{new} = 100$ ?

# SISTEMUL BINAR

- conversia între sisteme de numărare este foarte folositoare
- ce se întâmplă în baza  $B = 10$ ?
  - ce se întâmplă dacă vrem să trecem din baza  $B_{old} = 10$  în noua bază  $B_{new} = 100$ ?
    - câte cifre sunt în baza  $B_{new} = 100$ ? **100**
    - care sunt cifrele în baza  $B_{new} = 100$ ?

# SISTEMUL BINAR

- conversia între sisteme de numărare este foarte folositoare
- ce se întâmplă în baza  $B = 10$ ?
  - ce se întâmplă dacă vrem să trecem din baza  $B_{old} = 10$  în noua bază  $B_{new} = 100$ ?
    - câte cifre sunt în baza  $B_{new} = 100$ ? **100**
    - care sunt cifrele în baza  $B_{new} = 100$ ? **de la cifra “0” la cifra “99”**
    - primesc un număr în baza 10, cum îl transform în baza 100?
    - ex:  $(4837103)_{10} = (?????)_{100}$

# SISTEMUL BINAR

- conversia între sisteme de numărare este foarte folositoare
- ce se întâmplă în baza  $B = 10$ ?
  - ce se întâmplă dacă vrem să trecem din baza  $B_{old} = 10$  în noua bază  $B_{new} = 100$ ?
    - câte cifre sunt în baza  $B_{new} = 100$ ? **100**
    - care sunt cifrele în baza  $B_{new} = 100$ ? **de la cifra “0” la cifra “99”**
    - primesc un număr în baza 10, cum îl transform în baza 100?
    - ex:  $(4837103)_{10} = (“4” “83” “71” “3”)_{100}$
    - cum am obținut rezultatul? doar am grupat cifre consecutive, câte două – de ce două?

# SISTEMUL BINAR

- conversia între sisteme de numărare este foarte folositoare
- ce se întâmplă în baza  $B = 10$ ?
  - ce se întâmplă dacă vrem să trecem din baza  $B_{old} = 10$  în noua bază  $B_{new} = 100$ ?
    - câte cifre sunt în baza  $B_{new} = 100$ ? **100**
    - care sunt cifrele în baza  $B_{new} = 100$ ? **de la cifra “0” la cifra “99”**
    - primesc un număr în baza 10, cum îl transform în baza 100?
    - ex:  $(4837103)_{10} = (“4” “83” “71” “3”)_{100}$
    - cum am obținut rezultatul? doar am grupat cifre consecutive, câte două – de ce două?

**Regula generală:** când trecem din baza  $B$  în baza  $B^p$  trebuie doar să grupăm noul număr în cate  $p$  cifre

# SISTEMUL BINAR

- cineva spune: “Am cheltuit 1.000.000 de euro. O cifră enormă!!!” este corect?

21:57 56%

m.hotnews.ro/stire/2433680 [38] :

€ 4,1438 miercuri 07 oct 2020

**HotNews.ro** STIRI NON-STOP Versiune Mobile

Home Ultimele Economie Sport Actual

**Coronavirus Franța: Aproape 19.000 noi cazuri în ultimele 24 de ore, o cifră record**

de R.M. Miercuri, 7 octombrie 2020, 21:08

- a A+ Intra în cont

f t w m +



Autoritățile sanitare franceze au anunțat miercuri 18.746 noi cazuri de COVID-19 confirmate în ultimele 24 de ore, un nou record și aproape dublu față de bilanțul de marți, de 10.489 cazuri, potrivit agenției Reuters.

# SISTEMUL BINAR

- cineva spune: “Am cheltuit 1.000.000 de euro. O cifră enormă!!!”
- 1.000.000 e număr, nu cifră
- când poate să fie 1.000.000 cifră?
  - doar dacă cel care vorbește se referă la numere într-o baza numerică  $B \geq 1.000.001$
  - și atunci, ar trebui să spună “1.000.000”
- pentru ultima dată
  - în baza  $B = 10$ , cifrele sunt de la “0” la “9”
  - restul sunt numere
  - conceptul se generalizează pentru orice  $B$

# SISTEMUL BINAR

- un exemplu explicit:  $(1110111000001)_2$

- în baza 4:
- în baza 8:
- în baza 16 (hexazecimal):

$0_{\text{hex}}$	=	$0_{\text{dec}}$	=	$0_{\text{oct}}$	0	0	0	0
$1_{\text{hex}}$	=	$1_{\text{dec}}$	=	$1_{\text{oct}}$	0	0	0	1
$2_{\text{hex}}$	=	$2_{\text{dec}}$	=	$2_{\text{oct}}$	0	0	1	0
$3_{\text{hex}}$	=	$3_{\text{dec}}$	=	$3_{\text{oct}}$	0	0	1	1
$4_{\text{hex}}$	=	$4_{\text{dec}}$	=	$4_{\text{oct}}$	0	1	0	0
$5_{\text{hex}}$	=	$5_{\text{dec}}$	=	$5_{\text{oct}}$	0	1	0	1
$6_{\text{hex}}$	=	$6_{\text{dec}}$	=	$6_{\text{oct}}$	0	1	1	0
$7_{\text{hex}}$	=	$7_{\text{dec}}$	=	$7_{\text{oct}}$	0	1	1	1
$8_{\text{hex}}$	=	$8_{\text{dec}}$	=	$10_{\text{oct}}$	1	0	0	0
$9_{\text{hex}}$	=	$9_{\text{dec}}$	=	$11_{\text{oct}}$	1	0	0	1
$A_{\text{hex}}$	=	$10_{\text{dec}}$	=	$12_{\text{oct}}$	1	0	1	0
$B_{\text{hex}}$	=	$11_{\text{dec}}$	=	$13_{\text{oct}}$	1	0	1	1
$C_{\text{hex}}$	=	$12_{\text{dec}}$	=	$14_{\text{oct}}$	1	1	0	0
$D_{\text{hex}}$	=	$13_{\text{dec}}$	=	$15_{\text{oct}}$	1	1	0	1
$E_{\text{hex}}$	=	$14_{\text{dec}}$	=	$16_{\text{oct}}$	1	1	1	0
$F_{\text{hex}}$	=	$15_{\text{dec}}$	=	$17_{\text{oct}}$	1	1	1	1

# SISTEMUL BINAR

- un exemplu explicit:  $(1110111000001)_2$

- în baza 4: ("1" "11" "01" "11" "00" "00" "01")<sub>4</sub> = (1313001)<sub>4</sub>
- **în baza 8:**
- **în baza 16 (hexazecimal):**

$0_{\text{hex}}$	=	$0_{\text{dec}}$	=	$0_{\text{oct}}$	0	0	0	0
$1_{\text{hex}}$	=	$1_{\text{dec}}$	=	$1_{\text{oct}}$	0	0	0	1
$2_{\text{hex}}$	=	$2_{\text{dec}}$	=	$2_{\text{oct}}$	0	0	1	0
$3_{\text{hex}}$	=	$3_{\text{dec}}$	=	$3_{\text{oct}}$	0	0	1	1
$4_{\text{hex}}$	=	$4_{\text{dec}}$	=	$4_{\text{oct}}$	0	1	0	0
$5_{\text{hex}}$	=	$5_{\text{dec}}$	=	$5_{\text{oct}}$	0	1	0	1
$6_{\text{hex}}$	=	$6_{\text{dec}}$	=	$6_{\text{oct}}$	0	1	1	0
$7_{\text{hex}}$	=	$7_{\text{dec}}$	=	$7_{\text{oct}}$	0	1	1	1
$8_{\text{hex}}$	=	$8_{\text{dec}}$	=	$10_{\text{oct}}$	1	0	0	0
$9_{\text{hex}}$	=	$9_{\text{dec}}$	=	$11_{\text{oct}}$	1	0	0	1
$A_{\text{hex}}$	=	$10_{\text{dec}}$	=	$12_{\text{oct}}$	1	0	1	0
$B_{\text{hex}}$	=	$11_{\text{dec}}$	=	$13_{\text{oct}}$	1	0	1	1
$C_{\text{hex}}$	=	$12_{\text{dec}}$	=	$14_{\text{oct}}$	1	1	0	0
$D_{\text{hex}}$	=	$13_{\text{dec}}$	=	$15_{\text{oct}}$	1	1	0	1
$E_{\text{hex}}$	=	$14_{\text{dec}}$	=	$16_{\text{oct}}$	1	1	1	0
$F_{\text{hex}}$	=	$15_{\text{dec}}$	=	$17_{\text{oct}}$	1	1	1	1

# SISTEMUL BINAR

- un exemplu explicit:  $(1110111000001)_2$ 
  - în baza 4: ("1" "11" "01" "11" "00" "00" "01")<sub>4</sub> = (1313001)<sub>4</sub>
  - în baza 8: ("1" "110" "111" "000" "001")<sub>8</sub> = (16701)<sub>8</sub>
  - în baza 16 (hexazecimal):

<b>0<sub>hex</sub></b>	=	0 <sub>dec</sub>	=	0 <sub>oct</sub>	0	0	0	0
<b>1<sub>hex</sub></b>	=	1 <sub>dec</sub>	=	1 <sub>oct</sub>	0	0	0	1
<b>2<sub>hex</sub></b>	=	2 <sub>dec</sub>	=	2 <sub>oct</sub>	0	0	1	0
<b>3<sub>hex</sub></b>	=	3 <sub>dec</sub>	=	3 <sub>oct</sub>	0	0	1	1
<b>4<sub>hex</sub></b>	=	4 <sub>dec</sub>	=	4 <sub>oct</sub>	0	1	0	0
<b>5<sub>hex</sub></b>	=	5 <sub>dec</sub>	=	5 <sub>oct</sub>	0	1	0	1
<b>6<sub>hex</sub></b>	=	6 <sub>dec</sub>	=	6 <sub>oct</sub>	0	1	1	0
<b>7<sub>hex</sub></b>	=	7 <sub>dec</sub>	=	7 <sub>oct</sub>	0	1	1	1
<b>8<sub>hex</sub></b>	=	8 <sub>dec</sub>	=	10 <sub>oct</sub>	1	0	0	0
<b>9<sub>hex</sub></b>	=	9 <sub>dec</sub>	=	11 <sub>oct</sub>	1	0	0	1
<b>A<sub>hex</sub></b>	=	10 <sub>dec</sub>	=	12 <sub>oct</sub>	1	0	1	0
<b>B<sub>hex</sub></b>	=	11 <sub>dec</sub>	=	13 <sub>oct</sub>	1	0	1	1
<b>C<sub>hex</sub></b>	=	12 <sub>dec</sub>	=	14 <sub>oct</sub>	1	1	0	0
<b>D<sub>hex</sub></b>	=	13 <sub>dec</sub>	=	15 <sub>oct</sub>	1	1	0	1
<b>E<sub>hex</sub></b>	=	14 <sub>dec</sub>	=	16 <sub>oct</sub>	1	1	1	0
<b>F<sub>hex</sub></b>	=	15 <sub>dec</sub>	=	17 <sub>oct</sub>	1	1	1	1

# SISTEMUL BINAR

- un exemplu explicit:  $(1110111000001)_2$ 
  - în baza 4: ("1" "11" "01" "11" "00" "00" "01")<sub>4</sub> = (1313001)<sub>4</sub>
  - în baza 8: ("1" "110" "111" "000" "001")<sub>8</sub> = (16701)<sub>8</sub>
  - în baza 16 (hexazecimal): ("1" "1101" "1100" "0001")<sub>16</sub> = (1DC1)<sub>16</sub>

$0_{\text{hex}}$	=	$0_{\text{dec}}$	=	$0_{\text{oct}}$	0	0	0	0
$1_{\text{hex}}$	=	$1_{\text{dec}}$	=	$1_{\text{oct}}$	0	0	0	1
$2_{\text{hex}}$	=	$2_{\text{dec}}$	=	$2_{\text{oct}}$	0	0	1	0
$3_{\text{hex}}$	=	$3_{\text{dec}}$	=	$3_{\text{oct}}$	0	0	1	1
$4_{\text{hex}}$	=	$4_{\text{dec}}$	=	$4_{\text{oct}}$	0	1	0	0
$5_{\text{hex}}$	=	$5_{\text{dec}}$	=	$5_{\text{oct}}$	0	1	0	1
$6_{\text{hex}}$	=	$6_{\text{dec}}$	=	$6_{\text{oct}}$	0	1	1	0
$7_{\text{hex}}$	=	$7_{\text{dec}}$	=	$7_{\text{oct}}$	0	1	1	1
$8_{\text{hex}}$	=	$8_{\text{dec}}$	=	$10_{\text{oct}}$	1	0	0	0
$9_{\text{hex}}$	=	$9_{\text{dec}}$	=	$11_{\text{oct}}$	1	0	0	1
$A_{\text{hex}}$	=	$10_{\text{dec}}$	=	$12_{\text{oct}}$	1	0	1	0
$B_{\text{hex}}$	=	$11_{\text{dec}}$	=	$13_{\text{oct}}$	1	0	1	1
$C_{\text{hex}}$	=	$12_{\text{dec}}$	=	$14_{\text{oct}}$	1	1	0	0
$D_{\text{hex}}$	=	$13_{\text{dec}}$	=	$15_{\text{oct}}$	1	1	0	1
$E_{\text{hex}}$	=	$14_{\text{dec}}$	=	$16_{\text{oct}}$	1	1	1	0
$F_{\text{hex}}$	=	$15_{\text{dec}}$	=	$17_{\text{oct}}$	1	1	1	1

intr-un slide anterior am spus despre "99" ca e cifra in baza 100, pentru ca nu am litere pana la 99

pentru "99" putem continua pe lista ASCII extinsa (pornind de la F care este "15") pana ajungem la "99": š

# SISTEMUL BINAR

- **numere întregi negative**
  - până acum am văzut doar numere naturale
  - ce ați face voi că să reprezentați numere negative?
  - care este prima (cea mai simplă) idee?

# SISTEMUL BINAR

- **numere întregi negative**
  - până acum am văzut doar numere naturale
  - ce ați face voi că să reprezentați numere negative?
  - care este prima (cea mai simplă) idee?
    - pai trebuie să salvăm semnul numărului
    - cât spațiu ocupă asta? 1 bit
    - deci 1 bit pentru semn, restul pentru valoarea absolută
      - 1 101 ar fi -5
      - 0 101 ar fi 5
    - cum arată “zero” reprezentat asa?

# SISTEMUL BINAR

- **numere întregi negative**
  - până acum am văzut doar numere naturale
  - ce ați face voi că să reprezentați numere negative?
  - care este prima (cea mai simplă) idee?
    - pai trebuie să salvăm semnul numărului
    - cât spațiu ocupă asta? 1 bit
    - deci 1 bit pentru semn, restul pentru valoarea absolută
      - 1 101 ar fi -5
      - 0 101 ar fi 5
    - cum arată “zero” reprezentat aşa?

# SISTEMUL BINAR

- **numere întregi negative**

- până acum am văzut doar numere naturale
- ce ați face voi că să reprezentați numere negative?
- care este prima (cea mai simplă) idee?
  - pai trebuie să salvăm semnul numărului
  - cât spațiu ocupă asta? 1 bit
  - deci 1 bit pentru semn, restul pentru valoarea absolută
    - 1 101 ar fi -5
    - 0 101 ar fi 5
  - cum arată “zero” reprezentat aşa?
    - 1 000 ar fi -0
    - 0 000 ar fi 0
    - este redundant
  - și mai este o problemă: avem nevoie de circuite speciale pentru a face operații cu aceste numere (trebuie verificat primul bit și în funcție de asta trebuie decise operațiile)

# SISTEMUL BINAR

- numere întregi negative

bit $b_i$ :	1	1	1	1	0	0	0	1
$2^i$ :	$-2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

- ce se întâmplă? MSB este negativ

- $$x = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$$

- în exemplul de mai sus:
  - care este numărul reprezentat?
  - pe cati biți este reprezentat acest număr?

# SISTEMUL BINAR

- numere întregi negative

bit $b_i$ :	1	1	1	1	0	0	0	1
$2^i$ :	$-2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

- ce se întâmplă? MSB este negativ

- $$x = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$$

- în exemplul de mai sus:

- $-1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = -15$
  - 8 biți

- acesta este sistemul de reprezentare în complement față de doi

# SISTEMUL BINAR

- numere întregi negative

bit $b_i$ :	1	1	1	1	0	0	0	1
-------------	---	---	---	---	---	---	---	---

$2^i:$      $-2^7$      $2^6$      $2^5$      $2^4$      $2^3$      $2^2$      $2^1$      $2^0$

$$\bullet \quad x = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$$

- reprezentarea se numește *în complement față de doi*

- numerele în intervalul  $-2^{N-1}$  pana la  $2^{N-1} - 1$
- se “pierde” un bit pentru semn, e optim
- MSB este semnul, restul bițiilor sunt valoarea
- ca să putem folosi numere înscrise în operații aritmetice avem nevoie să facem niște transformări

*vedeți tabelul din dreapta, numerele pozitive sunt la fel ca înainte (dar pe 3 biți doar), cele negative arată la fel doar că primul bit este “1” (jumătatea de sus a tabelului are MSB “0”, jumătatea de jos este identică, dar cu MSB “1”)*

*ce se întâmplă dacă adunăm 1 la 7?*

complement față de doi	zecimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

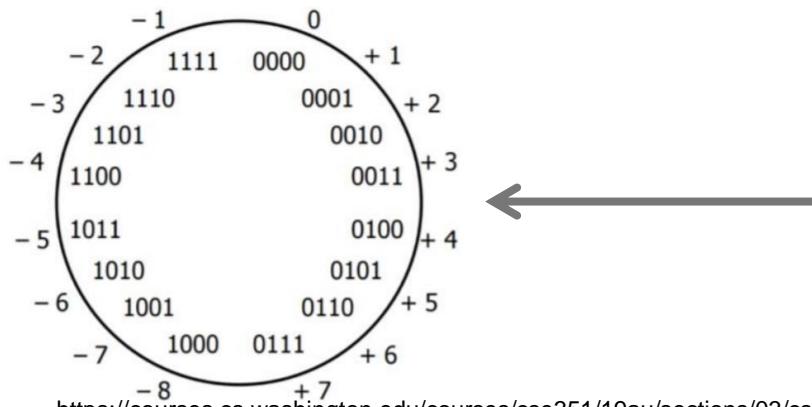
# SISTEMUL BINAR

- numere întregi negative

bit $b_i$ :	1	1	1	1	0	0	0	1
$2^i$ :	$-2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

$$\bullet \quad x = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$$

- reprezentarea se numește *în complement față de doi*
  - numerele în intervalul  $-2^{N-1}$  pana la  $2^{N-1} - 1$
  - se “pierde” un bit pentru semn, e optim
  - MSB este semnul, restul bițiilor sunt valoarea
  - ca să putem folosi numere înscrise în operații aritmetice avem nevoie să facem niște transformări



complement față de doi	zecimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

# SISTEMUL BINAR

- numere întregi negative

bit $b_i$ :	1	1	1	1	0	0	0	1
$2^i$ :	$-2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

- $x = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$

- cum reprezentăm un număr negativ zecimal x? (ex: x = -30)

- scriem  $|x|$  în binar

- setăm MSB și

inversăm restul biților

- adunăm unu

	0	0	1	1	1	1	0
1	1	1	0	0	0	0	1
1	1	1	0	0	0	1	0

- deci  $(-30)_{10} = (11100010)_2$

# SISTEMUL BINAR

- numere întregi negative

bit $b_i$ :	1	1	1	1	0	0	0	1
$2^i$ :	$-2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

- $x = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$

- cum reprezentăm un număr binar x? (ex: x = 1011 1010)

- scriem x în binar

1	0	1	1	1	0	1	0
---	---	---	---	---	---	---	---

- MSB = 1, deci x este negativ (dacă MSB = 0 atunci x e pozitiv)

- inversăm bitii

0	1	0	0	0	1	0	1
0	1	0	0	0	1	1	0

- adăugăm unu

- deci  $(10111010)_2 = (-70)_{10}$  (negativ,  $64 + 4 + 2 = 70$ )

# SISTEMUL BINAR

- operații aritmetice, numere naturale exemplu

1	0	1	1	0	0	1	0	178
0	0	1	1	0	1	1	0	54
+	1	1	1	0	1	0	0	232

# **SISTEMUL BINAR**

- de ce folosim acest sistem în complement față de 2?
- pentru că algoritmul de adunare este la fel ca pentru numere naturale, nu trebuie să schimbăm nimic
- circuitele anterioare care realizează adunarea naturală pot fi folosite și acum

# SISTEMUL BINAR

- operații aritmetice, numere întregi exemple
  - numărul negativ este mai mic (magnitudine) decât cel pozitiv

1	1	0	1	1	0	0	1	-39	
0	1	0	1	1	1	0	0	92	
+	0	0	1	1	0	1	0	1	53

- cum am obținut reprezentarea pentru -39?

	0	1	0	0	1	1	1	39
1	1	0	1	1	0	0	0	inversarea de biți
1	1	0	1	1	0	0	1	plus unu

# SISTEMUL BINAR

- operații aritmetice, numere întregi exemple
  - numărul negativ este mai mare (magnitudine) decât cel pozitiv

1	1	0	1	1	0	0	1	-39
0	0	0	1	0	1	0	1	21

$$+ \quad \begin{array}{cccccccc} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{array} \quad -18$$

- de unde am obținut -18?

1	1	1	0	1	1	1	0	rezultatul, primul bit e 1
0	0	0	1	0	0	0	1	inversarea de biți
0	0	0	1	0	0	1	0	plus unu

# SISTEMUL BINAR

- operații aritmetice, numere întregi exemple
  - ambele numere sunt negative

1	1	0	1	1	0	0	1	-39
1	1	1	0	1	1	1	0	-18
+	1	1	0	0	0	1	1	-57

- de unde am obținut -57?

1	1	0	0	0	1	1	1	rezultatul, primul bit e 1
0	0	1	1	1	0	0	0	inversarea de biti
0	0	1	1	1	0	0	1	plus unu

# SISTEMUL BINAR

- operații aritmetice, numere întregi exemple
  - overflow la limită

0	1	1	1	1	1	1	1	127
0	0	0	0	0	0	0	1	1
+	1	0	0	0	0	0	0	-128

- de unde am obținut -128?

1	0	0	0	0	0	0	0	rezultatul, primul bit e 1
0	1	1	1	1	1	1	1	inversarea de biți
1	0	0	0	0	0	0	0	plus unu: 128

-128 are aceeași reprezentare

# SISTEMUL BINAR

- operații aritmetice, numere întregi exemple
  - ambele numere pozitive, mari

0	1	1	1	0	1	1	1	119	
0	1	0	0	1	0	1	1	75	
+	1	1	0	0	0	0	1	0	-62

- de unde am obținut -62?

1	1	0	0	0	0	1	0	rezultatul, primul bit e 1
0	0	1	1	1	1	0	1	inversarea de biți
0	1	1	1	1	1	1	0	plus unu

- $119 + 75 = 194$  (nu încap pe 7 biți)

# SISTEMUL BINAR

- operații aritmetice, numere întregi exemple
  - ambele numere pozitive, mari

0	1	1	1	0	1	1	1	119	
0	1	0	0	1	0	1	1	75	
+	1	1	0	0	0	0	1	0	-62

- intuiția, de unde am obținut -62?
  - $119 + 75 = 194$  (nu încap pe 7 biți)
  - maximum e 127, deci avem  $194 - 127 = 67$  "extra"
  - overflow începe după 127, după 127 este -128 (folosim un extra)
  - deci 66 extra rămași pornesc de la -128
  - deci avem  $-128 + 66 = -62$

# SISTEMUL BINAR

- operații aritmetice, numere întregi exemple
  - overflow la limită

0	1	1	1	1	1	1	1	127
0	0	0	0	0	0	0	1	1
+	1	0	0	0	0	0	0	-128

- intuiția, de unde am obținut -128?
  - $127 + 1 = 128$  (nu încap pe 7 biți)
  - maximum e 127, deci avem  $128 - 127 = 1$  "extra"
  - overflow începe după 127, după 127 este -128 (folosim un extra)
  - deci 0 extra ramași pornesc de la -128
  - deci avem  $-128 + 0 = -128$

# SISTEMUL BINAR

- operații aritmetice, numere întregi exemple

- ambele numere negative, mari

1	0	0	0	1	0	0	1	-119
1	0	1	1	0	1	0	1	-75
+	0	0	1	1	1	1	1	62

- intuiția, de unde am obținut 62?
  - $-119 - 75 = -194$  (nu încap pe 7 biți)
  - minimul e -128, deci avem  $+194 - 128 = 66$  “extra”
  - underflow începe înainte de -128, înainte de -128 este 127 (folosim un extra)
  - deci 65 extra rămași pornesc de la 127
  - deci avem  $127 - 65 = 62$

# SISTEMUL BINAR

- extinderea numărului de biți pentru reprezentare
  - să presupunem că avem numărul 01 11 11 reprezentat pe 6 biți
  - ni se spune că numărul este natural
  - ni se spune că putem folosi încă 2 biți pentru reprezentare
    - noua reprezentare este:

# SISTEMUL BINAR

- extinderea numărului de biți pentru reprezentare
  - să presupunem că avem numărul 01 11 11 reprezentat pe 6 biți
  - ni se spune că numărul este natural
  - ni se spune că putem folosi încă 2 biți pentru reprezentare
    - noua reprezentare este: 0001 1111
- ce se întâmplă acum dacă avem numărul 10 00 11 reprezentat pe 6 biți (dar știm că suntem în complement față de doi)
- ni se spune din nou că putem folosi încă 2 biți pentru reprezentare
  - noua reprezentare este: 0010 0011 ?

# SISTEMUL BINAR

- extinderea numărului de biți pentru reprezentare
  - să presupunem că avem numărul 01 11 11 reprezentat pe 6 biți
  - ni se spune că numărul este natural
  - ni se spune că putem folosi încă 2 biți pentru reprezentare
    - noua reprezentare este: 0001 1111
  - ce se întâmplă acum dacă avem numărul 10 00 11 reprezentat pe 6 biți (dar știm că suntem în complement față de doi)
  - ni se spune din nou că putem folosi încă 2 biți pentru reprezentare
    - noua reprezentare este: 0010 0011 ?
    - numărul acesta nici măcar nu este negativ (MSB este 0)
    - deci nu putem extinde cu “zero”
    - cu ce extindem?

# SISTEMUL BINAR

- extinderea numărului de biți pentru reprezentare
  - să presupunem că avem numărul 01 11 11 reprezentat pe 6 biți
  - ni se spune că numărul este natural
  - ni se spune că putem folosi încă 2 biți pentru reprezentare
    - noua reprezentare este: 0001 1111
  - ce se întâmplă acum dacă avem numărul 10 00 11 reprezentat pe 6 biți (dar știm că suntem în complement față de doi)
  - ni se spune din nou că putem folosi încă 2 biți pentru reprezentare
    - noua reprezentare este: 0010 0011 ?
    - numărul acesta nici măcar nu este negativ (MSB este 0)
    - deci nu putem extinde cu “zero”
    - cu ce extindem? cu “unu”
    - noua reprezentare este: 1110 0011

# SISTEMUL BINAR

- **încă un exemplu:**
- **extinderea numărului de biți pentru reprezentare**
  - să presupunem că avem numărul 10111 reprezentat pe 5 biți
  - trecem în baza  $B = 8$ 
    - numărul în baza  $B = 8$  este  $10\ 111 = 27$
    - dar, dacă vedem  $(27)_8$  atunci am putea crede ca în binar avem 010111
    - dar 010111 este pe 6 biți și este pozitiv
  - **ideea:** când trecem din binar în baza  $B = 8$  (și știm că aici implicit avem 6 biți de reprezentare) atunci extindem reprezentarea
  - deci, pornim cu 110 111
  - iar în baza  $B = 8$  atunci avem  $110\ 111 = 67$
  - problema este generată de faptul ca 3 nu îl împarte exact pe 5
  - ambele variante sunt corecte:  $(27)_8$  dar știi că sunt 5 biți sau  $(67)_8$  și crezi că sunt 6 biți (implicit când vezi două cifre în baza  $B = 8$  crezi că sunt 6 biți)

# SISTEMUL BINAR

- logica binară (**0 = False, 1 = True**)
- operații logice:
  - NOT (negația)
  - AND (conjuncția)
  - OR (disjuncția)
  - XOR (disjuncția exclusivă)

X	NOT X
0	1
1	0

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

- pentru numere reprezentate binar operația logică se face pentru fiecare bit în parte (pentru numere pe  $N$  biți, sunt  $N$  operații)

# SISTEMUL BINAR

- logica binară, exemplu

1	0	1	1	0	0	1	0	178
0	0	1	1	0	1	1	0	54

AND: 

0	0	1	1	0	0	1	0	50
---	---	---	---	---	---	---	---	----

OR: 

1	0	1	1	0	1	1	0	182
---	---	---	---	---	---	---	---	-----

XOR: 

1	0	0	0	0	1	0	0	132
---	---	---	---	---	---	---	---	-----

- aparent, valorile zecimale nu au interpretare clară
- totuși, putem spune ceva: OR încurajează apariția de biți “1”, AND o descurajează, iar la XOR ... depinde
- totuși, vom vedea că logica binară (în combinații interesante) ne poate spune multe și despre numere zecimale

# CE AM FĂCUT ASTĂZI

- am discutat despre istoria sistemelor de calcul
- am acoperit elementele de bază ale sistemului binar
  - reprezentarea binară (numere naturale)
  - calcule cu baze  $B$  diferite
  - reprezentarea complement față de doi (numere întregi)
  - operații binare elementare
- la seminar, vom vedea cum facem operații în sistemul binar și cum folosim complementul față de doi

# **DATA VIITOARE ...**

- introducere în teoria informației
- cum măsurăm cantitatea de informație
- entropia lui Shannon
- continuăm să studiem sistemul binar
- ... mai târziu: înmulțirea și împărțirea
- ... și mai târziu: floating point

# LECTURĂ SUPLIMENTARĂ

- **PH book**
  - 2.4 Signed and Unsigned Numbers
  - 2.17 Real Stuff: x86 Instructions
  - 3.2 Addition and Subtraction
- **Thomas Finley course notes,**  
<https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html>
- **mașina Turing și conceptul de Turing-complete:**
  - Turing Machines Explained – Computerphile,  
<https://youtube.com/watch?v=dNRDvLACg5Q>
  - Turing Complete – Computerphile,  
<https://www.youtube.com/watch?v=RPQD7-AOjMI>

# **LECTURĂ SUPLIMENTARĂ (NU INTRA IN EXAMEN)**

- dacă sunteți interesați de istoria sistemelor de calcul vă sugerez următoarele prezentări:
  - Computer Pioneers: Pioneer Computers Part 1 (1935 – 1945),  
<https://www.youtube.com/watch?v=qundvme1Tik>
  - Computer Pioneers: Pioneer Computers Part 2 (1946 – 1950),  
<https://www.youtube.com/watch?v=wsirYCAocZk>
  - BBC History of Computers,  
<https://www.youtube.com/playlist?list=PL1331A4548513EA81>
  - <https://www.gresham.ac.uk/lectures-and-events/a-very-brief-history-of-computing-1948-2015>
  - The Grand Narrative of the History of Computing,  
<https://www.youtube.com/watch?v=njwQgz63rls>



# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x02**

**INTRODUCERE ÎN TEORIA INFORMAȚIEI**

Cristian Rusu

# **DATA TRECUTĂ**

- am văzut cum transformăm dintr-o bază numerică în alta
- reprezentarea în complement față de doi
- am văzut operații de adunare și logice în binar
- dar de unde vin acești biți pe care facem operații?
- azi, vom studia conceptul de informație propriu-zis ...

# CUPRINS

- **teoria informației**
  - ce este informația
  - cum putem măsura cantitatea de informație
  - codarea datelor
  - detectarea și corectarea erorilor
- **referințe bibliografice**

# TEORIA INFORMAȚIEI

- ce este “informația”?
  - niște date care afectează (în general, reduc) incertitudinea pe care o avem despre un eveniment/fenomen/etc.
- **un exemplu: se dă un pachet de cărți de joc cu 52 de cărți, care eveniment aduce cea mai mare cantitate de informație?**
  - extragem din pachet o carte care este inimă roșie
  - extragem din pachet o carte care este un rege
  - în care situație avem informație mai exactă despre carte?

# TEORIA INFORMAȚIEI

- ce este “informația”?
  - niște date care afectează (în general, reduc) incertitudinea pe care o avem despre un eveniment/fenomen/etc.
- **un exemplu: se dă un pachet de cărți de joc cu 52 de cărți, care eveniment aduce cea mai mare cantitate de informație?**
  - extragem din pachet o carte care este inimă roșie (13/52)
  - **extragem din pachet o carte care este un rege (4/52)**
- **de ce? probabilitatea de a extrage un rege din pachet este mult mai mică, deci dacă extragem această carte primim mai multă informație (evenimentele rare produc multă informație)**
- **alt mod de a vedea problema: cărți de inimă roșie sunt 13, regi sunt doar 4 deci e mai ușor să ghicim ce carte a fost extrasă**

# TEORIA INFORMAȚIEI

- cum măsurăm cantitatea de informație?
- introducem o variabilă aleatoare X:
  - această variabilă poate lua  $N$  valori distincte ( $x_1, x_2, \dots, x_N$ )
  - fiecare valoare distinctă apare cu probabilitate ( $p_1, p_2, \dots, p_N$ )
- câtă informație primim dacă observăm că variabila X a luat valoarea  $x_i$ ?

$$I(x_i) = \log_2 \left( \frac{1}{p_i} \right)$$

- cu cât  $p_i$  este mai mic cu atât cantitatea de informație este mai mare
- în exemplul anterior:  $I(\text{rege}) = \log_2 \left( \frac{1}{\frac{4}{52}} \right) = \log_2 \left( \frac{52}{4} \right) = 3.7$  biti

# TEORIA INFORMAȚIEI

- **cazurile anterioare:**

$$I(\text{rege}) = \log_2 \left( \frac{1}{\frac{4}{52}} \right) = \log_2 \left( \frac{52}{4} \right) = 3.7 \text{ biti}$$

$$I(\text{inima rosie}) = \log_2 \left( \frac{1}{\frac{13}{52}} \right) = \log_2 \left( \frac{52}{13} \right) = 2 \text{ biti}$$

- **câtă informație avem dacă știm exact cartea selectată din pachet?**

$$I(\text{o carte particulara}) = \log_2 \left( \frac{1}{\frac{1}{52}} \right) = \log_2 \left( \frac{52}{1} \right) = 5.7 \text{ biti}$$

- **aruncați o privire pe valorile de mai sus:  $5.7 = 3.7 + 2$** 
  - **e o concidență?**

# TEORIA INFORMAȚIEI

- unele experimente nu ne oferă informația completă
- informația se poate acumula: se pot realiza mai multe experimente (putem pune mai multe întrebări)
- informația nu este creată sau distrusă, este constantă
- am folosit câteva concepte din teoria probabilităților
  - care este probabilitatea ca un eveniment să se întâmple?
  - $P(\text{eveniment A}) = \frac{\text{număr situații în care se întâmplă A}}{\text{număr total situații}}$
  - $P(\text{eveniment A și eveniment B}) = P(\text{eveniment A}) \times P(\text{eveniment B})$ 
    - doar dacă cele două evenimente sunt independente
    - independența evenimentelor se presupune în multe situații pentru a simplifica rezultatele

# TEORIA INFORMAȚIEI

- să presupem că avem  $N$  evenimente, la fel de probabile
- rulăm un experiment (imperfect din punct de vedere informational) care ne spune că doar  $M$  evenimente au fost posibile din totalul de  $N$
- ex:  $N = 52$  (pentru că avem 52 de cărți posibile),  $M = 13$  (pentru că aflăm că am extras o carte de inimă roșie, dar nu știm care exact e cartea)

- câtă informație primim?

$$I = \log_2 \left( \frac{1}{M \frac{1}{N}} \right) = \log_2 \left( \frac{N}{M} \right)$$

- ce se întâmplă dacă:

- $M = 1$
- $N = M$
- $M > N$

# TEORIA INFORMAȚIEI

- să presupem că avem  $N$  evenimente, la fel de probabile
- rulăm un experiment (imperfect din punct de vedere informational) care ne spune că doar  $M$  evenimente au fost posibile din totalul de  $N$
- ex:  $N = 52$  (pentru că avem 52 de cărți posibile),  $M = 13$  (pentru că aflăm că am extras o carte de inimă roșie, dar nu știm care exact e cartea)
- câtă informație primim?

$$I = \log_2 \left( \frac{1}{M \frac{1}{N}} \right) = \log_2 \left( \frac{N}{M} \right)$$

- ce se întâmplă dacă:
  - $M = 1$  am redus incertitudinea la o singură variantă, perfect
  - $N = M$  nu am rezolvat nimic, incertitudinea este aceeași
  - $M > N$  mai rău, incertitudinea este mai mare

# TEORIA INFORMAȚIEI

- alte exemple
  - aruncăm un ban (cap/pajură)
    - $N = ?, M = ?, I = ?$

# TEORIA INFORMAȚIEI

- alte exemple
  - aruncăm un ban (cap/pajură)
    - $N = 2, M = 1, I = \log_2 (2/1) = 1$  bit
  - extragem o carte și vedem că e înimă roșie
    - $N = ?, M = ?, I = ?$

# TEORIA INFORMAȚIEI

- alte exemple
  - aruncăm un ban (cap/pajură)
    - $N = 2, M = 1, I = \log_2 (2/1) = 1$  bit
  - extragem o carte și vedem că e înimă roșie
    - $N = 52, M = 13, I = \log_2 (52/13) = 2$  biți
  - aruncăm două zaruri
    - $N = ?, M = ?, I = ?$

# TEORIA INFORMAȚIEI

- alte exemple
  - aruncăm un ban (cap/pajură)
    - $N = 2, M = 1, I = \log_2 (2/1) = 1$  bit
  - extragem o carte și vedem că e înimă roșie
    - $N = 52, M = 13, I = \log_2 (52/13) = 2$  biți
  - aruncăm două zaruri
    - $N = 36, M = 1, I = \log_2 (36/1) = 5.17$  biți
  - în filmele “Batman”, există un rău-făcător care are o monedă cu “cap” pe ambele părți
    - $N = ?, M = ?, I = ?$

# TEORIA INFORMAȚIEI

- alte exemple
  - aruncăm un ban (cap/pajură)
    - $N = 2, M = 1, I = \log_2 (2/1) = 1$  bit
  - extragem o carte și vedem că e înimă roșie
    - $N = 52, M = 13, I = \log_2 (52/13) = 2$  biți
  - aruncăm două zaruri
    - $N = 36, M = 1, I = \log_2 (36/1) = 5.17$  biți
  - în filmele “Batman”, există un rău-făcător care are o monedă cu “cap” pe ambele părți
    - $N = 1, M = 1, I = \log_2 (1/1) = 0$  biți

# ENTROPIA

- **entropia**
  - valoarea medie de informației primită despre o variabilă X

$$H(X) = E(I(X)) = \sum_{i=1}^N p_i \log_2 \frac{1}{p_i} = \sum_{i=1}^N -p_i \log_2 p_i$$

- $H(X)$  se numește entropia lui X
- $I(X)$  este informația despre X
- $E$  este “expected value”, operația care calculează valoarea medie
- exemplu:  $X = \{A, B, C, D\}$  cu probabilități  $\{1/3, 1/2, 1/12, ?\}$

# ENTROPIA

- **entropia**
  - valoarea medie de informației primită despre o variabilă X

$$H(X) = E(I(X)) = \sum_{i=1}^N p_i \log_2 \frac{1}{p_i} = \sum_{i=1}^N -p_i \log_2 p_i$$

- $H(X)$  se numește entropia lui X
- $I(X)$  este informația despre X
- $E$  este “expected value”, operația care calculează valoarea medie
- exemplu:  $X = \{A, B, C, D\}$  cu probabilități  $\{1/3, 1/2, 1/12, 1/12\}$ 
  - care este entropia?
  -

# ENTROPIA

- **entropia**

- valoarea medie de informației primită despre o variabilă X

$$H(X) = E(I(X)) = \sum_{i=1}^N p_i \log_2 \frac{1}{p_i} = \sum_{i=1}^N -p_i \log_2 p_i$$

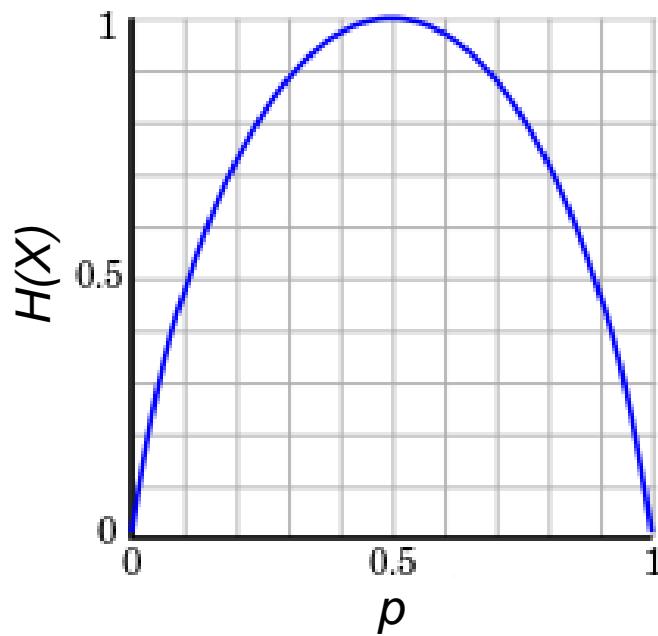
- $H(X)$  se numește entropia lui X
- $I(X)$  este informația despre X
- $E$  este “expected value”, operația care calculează valoarea medie
- exemplu:  $X = \{A, B, C, D\}$  cu probabilități  $\{1/3, 1/2, 1/12, 1/12\}$

- $H(X) = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{12} \log_2 \frac{1}{12} - \frac{1}{12} \log_2 \frac{1}{12} = 1.626$  biti
- variabila X are 4 opțiuni, deci în mod normal am avea nevoie de 2 biți să memorăm toate posibilitățile, dar (pentru că probabilitățile nu sunt egale) putem să codăm mai bine de 2 biți,

# ENTROPIA

$$H(X) = E(I(X)) = \sum_{i=1}^N p_i \log_2 \frac{1}{p_i} = \sum_{i=1}^N -p_i \log_2 p_i$$

- considerăm o monedă pe care o putem arunca
- probabilitatea de a vedea cap este  $p$  (iar pentru pajură este  $1-p$ )



# ENTROPIA

- exemplu:  $X = \{A, B, C, D\}$  cu probabilități  $\{1/3, 1/2, 1/12, 1/12\}$
- entropia lui  $X$  este 1.626
- ce se întâmplă?
  - pot memora variabila  $X$  folosind câte 2 biți (codarea este: 00, 01, 10, 11) pentru fiecare eveniment posibil
    - este în regulă, dar ineficient
  - de ce este ineficient? pentru că entropia este 1.626 deci ne spune că putem fi mai eficienți, adică codarea de mai sus nu este cel mai eficient mod în care putem coda informația
    - în loc de 2 biți per eveniment putem să avem doar 1.626 biți
  - entropia ne spune că nu putem coda variabila de mai sus sub 1.626 biți per eveniment fără să pierdem informație
    - de exemplu, am putea coda  $X$  cu un singur bit (0 sau 1) dar atunci putem distinge doar între două evenimente, nu patru
      - deci nu putem “decoda” ce s-a întâmplat
- entropia este limita de compresie posibilă

# CODAREA DATELOR

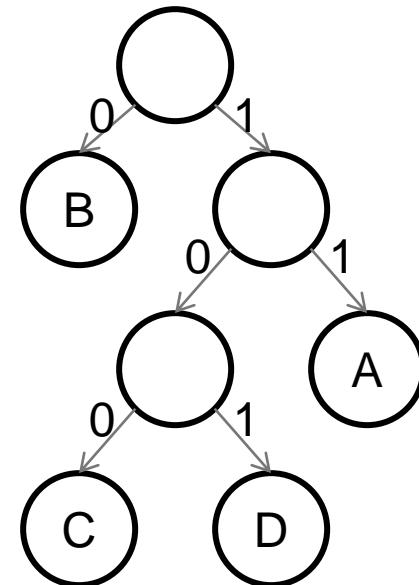
- cum atingem acel 1.626 biți pentru cele patru evenimente?
- folosim o codare diferită de cea standard
  - codarea standard A = 00, B = 01, C = 10, D = 11
  - cu această codare avem ABBC = 00 01 01 10
  - decodarea este directă: luăm câte 2 biți și fiecare e un eveniment
- o altă codare (mai eficientă):
  - dimensiune variabilă a codului: A = 01, B = 1, C = 000, D = 001
  - acum avem ABBC = 01 1 1 000
  - acum sunt 7 biți, față de 8 înainte (deci e mai bine)
  - decodarea trebuie să fie unică! trebuie să ne putem întoarce
- o altă codare (și mai eficientă, dar incorectă):
  - dimensiune variabilă a codului: A = 1, B = 0, C = 10, D = 11
  - acum avem ABBC = 1 0 0 10
  - 5 biți, și mai bine
  - problema? decodarea: dacă primim 10010 cum îl decodăm?

# CODAREA DATELOR

- cum atingem acel 1.626 biți pentru cele patru evenimente?
- folosim o codare diferită de cea standard
  - codarea standard A = 00, B = 01, C = 10, D = 11
  - cu această codare avem ABBC = 00 01 01 10
  - decodarea este directă: luăm câte 2 biți și fiecare e un eveniment
- o altă codare (mai eficientă):
  - dimensiune variabilă a codului: A = 01, B = 1, C = 000, D = 001
  - acum avem ABBC = 01 1 1 000
  - acum sunt 7 biți, față de 8 înainte (deci e mai bine)
  - decodarea trebuie să fie unică! trebuie să ne putem întoarce
- o altă codare (și mai eficientă, dar incorectă):
  - dimensiune variabilă a codului: A = 1, B = 0, C = 10, D = 11
  - acum avem ABBC = 1 0 0 10
  - 5 biți, și mai bine
  - problema? decodarea: dacă primim 10010 cum îl decodăm?
    - ABBC sau CBC sau ...

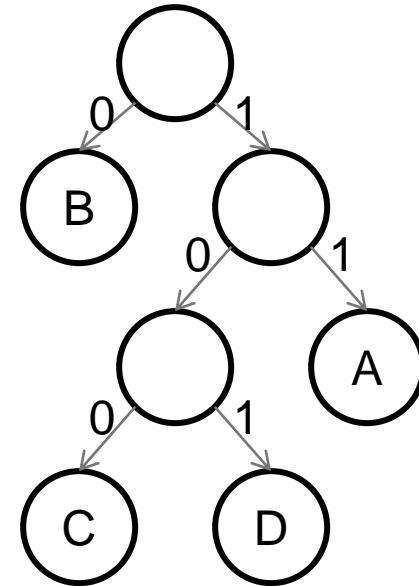
# CODAREA DATELOR

- cum putem crea o codare eficientă și unică?
  - un arbore binar
    - frunzele sunt codurile
    - stânga/dreapta e decis de 0/1
    - codarea este:
      - $B = 0$
      - $A = 11$
      - $C = 100$
      - $D = 101$
    - asta garantează codare eficientă și decodare unică
  - cum generăm codarea eficientă?
    - algoritmul Huffman
    - input: probabilitatea fiecărui eveniment  $\{1/3, \frac{1}{2}, 1/12, 1/12\}$
    - output: codurile care se citesc de pe un arbore binar (mai sus)
    - cheia: unele evenimente/simboluri apar mai des decât altele, deci acestea primesc o codare mai scurtă
    - dacă toate evenimentele sunt equiprobabile, atunci nu putem face nimic



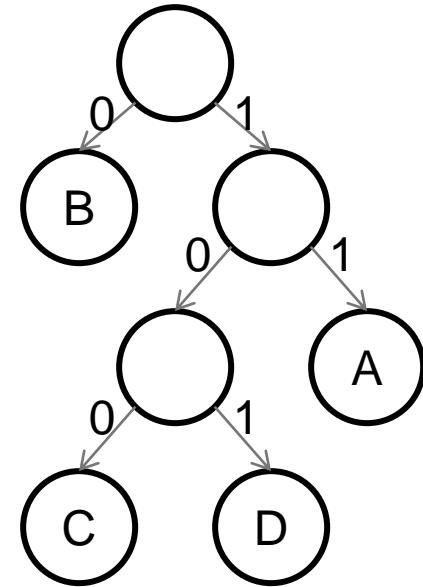
# CODAREA DATELOR

- cum putem crea o codare eficientă și unică?
  - un arbore binar
    - frunzele sunt codurile
    - stânga/dreapta e decis de 0/1
    - codarea este:
      - $B = 0$
      - $A = 11$
      - $C = 100$
      - $D = 101$
    - asta garantează codare eficientă și decodare unică
  - exercițiu: **decodați 00100111010011**



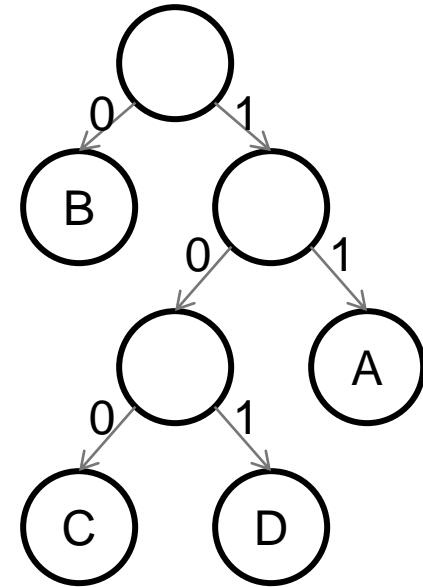
# CODAREA DATELOR

- cum putem crea o codare eficientă și unică?
  - un arbore binar
    - frunzele sunt codurile
    - stânga/dreapta e decis de 0/1
    - codarea este:
      - $B = 0$
      - $A = 11$
      - $C = 100$
      - $D = 101$
    - asta garantează codare eficientă și decodare unică
  - exercițiu: decodați 0 0 100 11 101 0 0 11
    - soluția: BBCADBBA



# CODAREA DATELOR

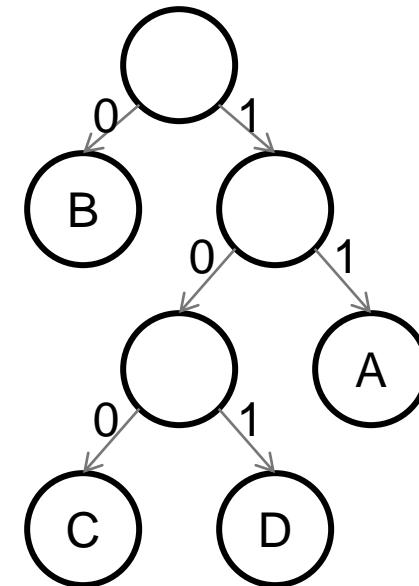
- cum putem crea o codare eficientă și unică?
  - un arbore binar
    - frunzele sunt codurile
    - stânga/dreapta e decis de 0/1
    - codarea este:
      - $B = 0$
      - $A = 11$
      - $C = 100$
      - $D = 101$
    - asta garantează codare eficientă și decodare unică
  - exercițiu: cum calculăm eficiența acestei codări? dimensiunea în medie a unui mesaj este? probabilitățile sunt  $\{1/3, 1/2, 1/12, 1/12\}$



# CODAREA DATELOR

- cum putem crea o codare eficientă și unică?

- un arbore binar
  - frunzele sunt codurile
  - stânga/dreapta e decis de 0/1
  - codarea este:
    - $B = 0$
    - $A = 11$
    - $C = 100$
    - $D = 101$
  - asta garantează codare eficientă și decodare unică



- exercițiu: cum calculăm eficiența acestei codări? dimensiunea în medie a unui mesaj este? probabilitățile sunt  $\{1/3, 1/2, 1/12, 1/12\}$ 
  - $2 \times 1/3 + 1 \times 1/2 + 3 \times 1/12 + 3 \times 1/12 = 1.667$  biți
  - comparat cu 1.626 biți care e optim

# CODAREA DATELOR

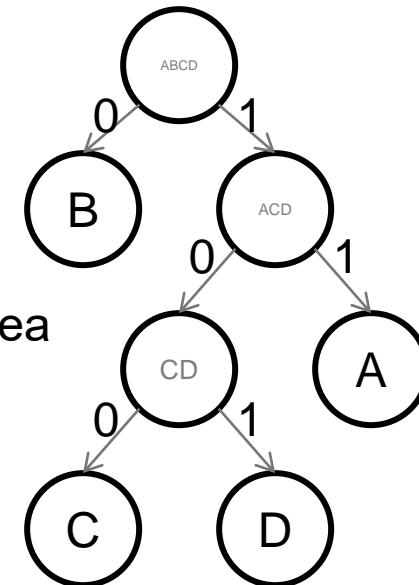
- **deja am văzut noi numere codate în sistemul binar**
  - codarea/memorarea numerelor naturale și întregi
  - ce fel de codare a fost aceasta? dimensiune fixă
  - indiferente de ce număr memorăm, folosim  $N$  biți
  - care sunt avantajele codării cu dimensiune fixă?
  - care sunt avantajele codării cu dimensiune variabilă?

# CODAREA DATELOR

- **deja am văzut noi numere codate în sistemul binar**
  - codarea/memorarea numerelor naturale și întregi
  - ce fel de codare a fost aceasta? dimensiune fixă
  - indiferente de ce număr memorăm, folosim  $N$  biți
  - care sunt avantajele codării cu dimensiune fixă?
    - e simplu, știm că fiecare simbol are același număr de biți (deci știm de cât spațiu avem nevoie, etc.)
    - putem accesa direct al  $i$ -lea simbol din sir (ABBA etc.)
    - implementarea în circuite electronice se face la fel pentru că toate elementele au același număr de biți (32 sau 64 de biți)
    - este optimă dacă simbolurile au probabilități egale
  - care sunt avantajele codării cu dimensiune variabilă?
    - codare eficientă (spațiu de stocare)

# CODAREA DATELOR

- algoritmul Huffman
  - input: probabilitatea fiecărui eveniment  $\{1/3, \frac{1}{2}, 1/12, 1/12\}$
- algoritm:
  - luați evenimentele cele mai improbabile
    - C și D, și le punem în arbore
    - creăm un nou eveniment CD, probabilitatea de apariție a acestuia este  $1/6$  (suma C și D)
    - noile evenimente sunt  $\{A, B, CD\}$  cu probabilități  $\{1/3, \frac{1}{2}, 1/6\}$
  - din nou evenimentele cele mai improbabile
    - A și CD, A merge pe cealaltă frunză
    - noile evenimente sunt  $\{B, ACD\}$  cu probabilități  $\{\frac{1}{2}, \frac{1}{2}\}$
  - din nou evenimentele cele mai improbabile
    - acum sunt doar două,  $\{B, ACD\}$ , B merge pe cealaltă frunză



# CODAREA DATELOR

- algoritmul Huffman este optim dacă considerăm un singur simbol pe rând
- dar putem uni simboluri, adică putem face același arbore binar pentru toate combinațiile de câte două simboluri
  - AA, AB, AC, AD, BA, ..., CA, CB, CC, ..., DA, DB, DC, DD
- câteva întrebări:
  - câte simboluri avem acum? **16**
  - care este probabilitatea următoarelor evenimente?
    - AA **1/9**
    - DD **1/144**
    - DB **1/24**
    - AD **1/36**
  - cum calculăm entropia acum? o sumă de 16 termeni
    - noua lungime medie Huffman, cu câte două simboluri? 1.646 biți
    - mai bine decât înainte (Huffman cu un singur simbol, 1.667 biți)
    - mai aproape de entropia de 1.626 biți (optim)
    - deci, dacă asociem mai multe simboluri convergem către 1.626 biți

# CODAREA DATELOR

- **comprimăm un text (avem un zip, de exemplu)**
  - putem comprima: text, imagini, chiar și executabile
- **ce ne oprește să mai comprimăm o dată? (să comprimăm zip-ul)**

# CODAREA DATELOR

- **comprimăm un text (avem un zip, de exemplu)**
  - putem comprima: text, imagini, chiar și executabile
- **ce ne oprește să mai comprimăm o dată? (să comprimăm zip-ul)**
- **conținutul zip-ului arată complet aleator**
  - ceva comprimat perfect arată ca zgomot
  - doar zgomotul nu poate fi comprimat
  - dacă pierdem ceva din zgomot nu mai putem recupera

# DETECTAREA / CORECTAREA ERORILOR

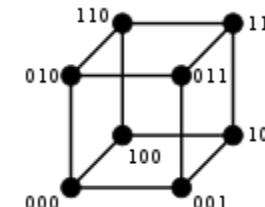
- detectarea erorilor
  - ce se întâmplă dacă memorăm un sir binar dar se întamplă ceva eroare: un 0 devine 1 sau un 1 devine 0?
  - de exemplu:
    - inițial avem: 0110 0101
    - datele sunt corupte și avem: 1100 0001
  - primul pas:
    - trebuie să definim o distanță între sirul corect și cel corrupt
    - distanța Hamming între două siruri binare: câți biți sunt diferenți (biți de pe aceleași poziții în prezentarea binară)
    - sirurile trebuie să aibă aceeași lungime
    - distanța Hamming mai sus: 3

# DETECTAREA / CORECTAREA ERORILOR

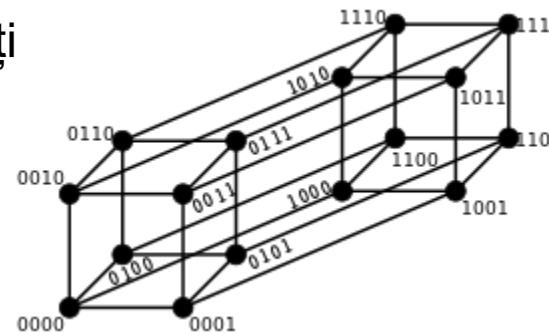
- detectarea erorilor

- primul pas:

- trebuie să definim o distanță între sirul corect și cel corrupt
    - distanța Hamming între două siruri binare: câți biți sunt diferiți (biți de pe aceleași poziții în prezentarea binară)
    - sirurile trebuie să aibă aceeași lungime
    - distanța Hamming în exemplul anterior: 3
  - Hamming pentru siruri de 3 biți,



- Hamming pentru siruri de 4 biți,



# DETECTAREA / CORECTAREA ERORILOR

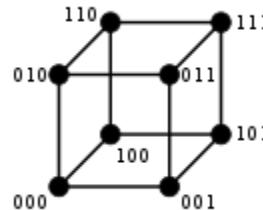
- detectarea erorilor
  - exemplul anterior:
    - inițial avem: 0110 0101
    - datele sunt corupte și avem: 1100 0001
  - problema: 0110 0101 și 1100 0001 sunt siruri valide
  - adică, nu ne putem da seamă că o eroare s-a produs
  - ideea: nu toate sirurile binare vor fi posibile: în felul acesta, dacă apare un sir binar care nu este permis (care nu există) atunci știm că o eroare s-a produs undeva
  - ideea cea mai simplă: adăugăm un bit de paritate simbolurilor
    - 0 devine 00
    - 1 devine 11
    - doar aceste două noi simboluri sunt valide

# DETECTAREA / CORECTAREA ERORILOR

- detectarea erorilor
  - ideea cea mai simplă: adăugăm un bit de paritate simbolurilor
    - 0 devine 00
    - 1 devine 11
    - doar aceste două noi simboluri sunt valide
  - dacă primim 01 sau 10, ştim că o eroare s-a produs
  - dacă primim 00 sau 11, ştim că totul e OK
  - ce am făcut? am adăugat redundanță: adică putem pierde ceva și tot ne putem da seama ce simbol avem
  - **ce pierdem?** suntem de două ori mai ineficienți (în loc de un bit trebuie acum să stocăm doi biți)
  - putem detecta erori multiple? nu. dacă sunt două erori atunci 00 se poate transforma în 11 din cauza erorilor

# DETECTAREA / CORECTAREA ERORILOR

- corecția erorilor
  - distanță Hamming destul de mare poate să ducă și la corectarea erorilor (nu doar detectarea lor)
  - să considerăm siruri de 3 biți



- singurele coduri valide sunt “000” care e “0” și “111” care e “1”
  - dacă primim “001” sau “010” sau “100” putem suspecta că e “0”
  - dacă primim “110” sau “101” sau “011” putem suspecta că e “1”
- o distanță Hamming de  $2E+1$  poate corecta  $E$  erori
  - care e intuiția?

# DETECTAREA / CORECTAREA ERORILOR

- corecția erorilor
  - distanță Hamming destul de mare poate să ducă și la corectarea erorilor (nu doar detectarea lor)
  - să considerăm siruri de 3 biți
- singurele coduri valide sunt “000” care e “0” și “111” care e “1”
  - dacă primim “001” sau “010” sau “100” putem suspecta că e “0”
  - dacă primim “110” sau “101” sau “011” putem suspecta că e “1”
- o distanță Hamming de  $2E+1$  poate corecta  $E$  erori
  - care e intuiția? majority vote
- în general, dacă vrem să detectăm  $E$  erori avem nevoie de o distanță Hamming între coduri de  $E + 1$ : adică “0” poate să fie “000” iar “1” poate să fie “111” – suntem de 3 ori mai ineficienți acum ca stocare, dar putem detecta mai 2 erori și putem corecta o eroare

# TEORIA INFORMAȚIEI

- un exercițiu ...

# TEORIA INFORMAȚIEI

Eur\*pa a în\*eg\*st\*\*t cel mai m\*\*e n\*m\*r să\*\*ăm\*nal de c\*\*uri de \*\*ro\*\*virus de până ac\*\*, iar Or\*\*\*\*\*ia M\*\*\*ială a Sa\*\*tății a averti\*\*\* că bil\*\*tu\*ile zi\*\*ic\* ale de\*es\*\*or ar putea aju\*\*e în ap\*\*lie 2021 să fie de 4-5 ori mai m\*\*i decât în pri\*\*\*ra acestui an. S\*ptes\*\*\*zece țări, între care și R\*\*\*\*ia, din to\*\*lul celor 27 de s\*\*\*e me\*\*\*e UE p\*\*\* M\*\*\*\* B\*\*\*\*\*e \*\*\*t \*\*\*\*ate cu r\*\*u pe n\*\*a h\*\*\*\* eană de \*\*\*\* ep\*\*\*\*\*.

# TEORIA INFORMAȚIEI

Europa a înregistrat cel mai mare număr săptămânal de cazuri de coronavirus de până acum, iar Organizația Mondială a Sănătății a avertizat că bilanțurile zilnice ale deceselor ar putea ajunge în aprilie 2021 să fie de 4-5 ori mai mari decât în primăvara acestui an. Șaptesprezece țări, între care și România, din totalul celor 27 de state membre UE plus Marea Britanie sunt marcate cu roșu pe noua hartă europeană de risc epidemic.

# TEORIA INFORMAȚIEI

- un alt exercițiu ...

# TEORIA INFORMAȚIEI

I cnduo't bvleiee taht I culod aulacly uesdtannrd waht I was rdnaieg. Unisg the icndeblire pweor of the hmuau mnid, aocdcrnig to rseecrah at Cmabrigde Uinervtisy, it dseno't mttaer in waht oderr the Iterets in a wrod are, the olny irpoamtnt tihng is taht the frsit and lsat ltteer be in the rghit pclae. The rset can be a taotl mses and you can stil raed it whoutit a pboerlm. Tihs is bucseae the huamn mnid deos not raed ervey ltteer by istlef, but the wrod as a wlohe. Aaznmig, huh? Yaeh and I awlyas tghhuot sleinpg was ipmorant! See if yuor fdreins can raed tihs too.

# **TEORIA INFORMAȚIEI**

- de ce am făcut aceste exerciții?
- care este “morala”?

# TEORIA INFORMAȚIEI

- de ce am făcut aceste exerciții?
- care este “morala”?
- **limba română și engleză sunt redundante**
  - În plus, mașina de decodare pe care o avem (creierul) este destul de performantă în astfel de situații
- **putem extrapola și putem spune că toate limbile lumii sunt redundante**
- **altfel, nu am putea comunica din cauza zgomotului (în cazul acesta, zgomot este la propriu)**

# **CE AM FĂCUT ASTĂZI**

- am discutat despre informație
- am discutat despre entropia lui Shannon
- am văzut cum codăm/decodăm informația
- detectarea și corecția erorilor

# **DATA VIITOARE ...**

- abstractizarea digitală
- începem să discutăm despre circuite electronice
- tranzistorul
- cum sunt implementate operații de pe un sistem de calcul cu circuite electronice
- circuite de bază

# LECTURĂ SUPLIMENTARĂ

- David MacKay, **Information Theory, Inference, and Learning Algorithms**, 2003
  - I Data Compression
  - cele 3 capitole introductive
- **The Universe is Hostile to Computers,**  
[https://www.youtube.com/watch?v=AaZ\\_RSt0KP8](https://www.youtube.com/watch?v=AaZ_RSt0KP8)
- **video-urile lui MacKay (primele sunt cele relevante pentru noi)**
  - <https://www.youtube.com/playlist?list=PLruBu5BI5n4aFpG32iMbdWoRVAA-Vcs06>

# **LECTURĂ SUPLIMENTARĂ (NU INTRA ÎN EXAMEN)**

- Sean Carroll, The Biggest Ideas in the Universe | 20. Entropy and Information, <https://www.youtube.com/watch?v=rBPPOI5Ule0>
- Computerphile, playlist despre entropie și informație, [https://www.youtube.com/playlist?list=PLzH6n4zXuckpKAj1\\_88VS-8Z6yn9zX\\_P6](https://www.youtube.com/playlist?list=PLzH6n4zXuckpKAj1_88VS-8Z6yn9zX_P6)
- 3Blue1Brown, Hamming codes, how to overcome noise, <https://www.youtube.com/watch?v=X8jsijhIIA>
- 3Blue1Brown, Hamming codes part 2, the elegance of it all, [https://www.youtube.com/watch?v=b3NxrxZOu\\_CE](https://www.youtube.com/watch?v=b3NxrxZOu_CE)
- Computerphile, Reed-Solomon Encoding, <https://www.youtube.com/watch?v=fBRMaEAFLE0>



# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x03**

**ABSTRACTIZAREA DIGITALĂ ȘI  
CIRCUITE COMBINAȚIONALE**

Cristian Rusu

# **DATA TRECUTĂ**

- am discutat despre conceptul de informație
- am văzut cum măsurăm informația
- am calculat entropia lui Shannon
- am văzut algoritmul Huffman pentru codarea variabilă a datelor
- am folosit distanța Hamming între două siruri de biți
- azi, vom vedea cum implementăm acești biți în circuite

# CUPRINS

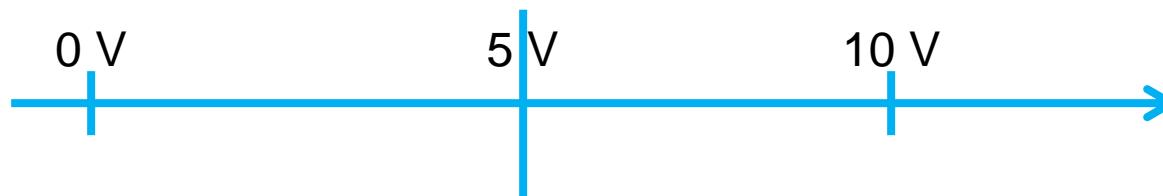
- abstractizarea digitală
- circuite digitale
- tranzistorul
- circuite combinaționale
- referințe bibliografice

# ABSTRACTIZAREA DIGITALĂ

- **avem nevoie de o reprezentare fizică a bițiilor**
- **avem nevoie de o metodă de stocare**
- **ce caracteristici am dori?**
  - să se poată stoca mulți biți
  - să fie ieftin să stocăm per bit, pentru că avem mulți
  - să fie o reprezentare stabilă (să nu dispară sau să se degradeze în timp)
  - să îi putem manipula ușor
- **soluția: folosim proprietăți electrice**
  - de obicei voltaj
  - dar voltajul are valori continue (majoritatea efectelor în natură sunt continue) iar noi vrem binar
  - avem nevoie de o metodă de cuantizare

# ABSTRACTIZAREA DIGITALĂ

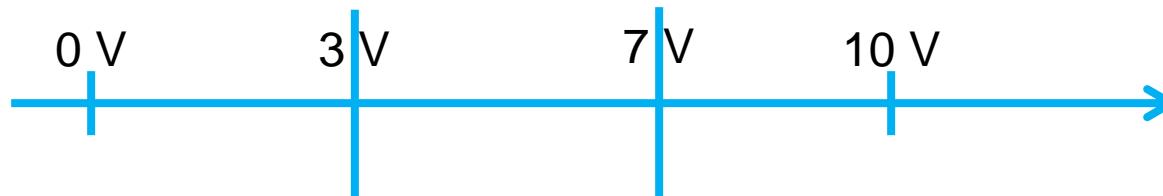
- de la continuu la digital



- cea mai simplă ideea: avem un voltaj maxim care poate să fie atins: deci de la 0V la 5V codăm "0" iar de la 5V la 10V avem "1"
- ce dificultăți avem în această situație?
  - e dificil să înțelegem ce se întamplă în jurul lui 5V

# ABSTRACTIZAREA DIGITALĂ

- de la continuu la digital

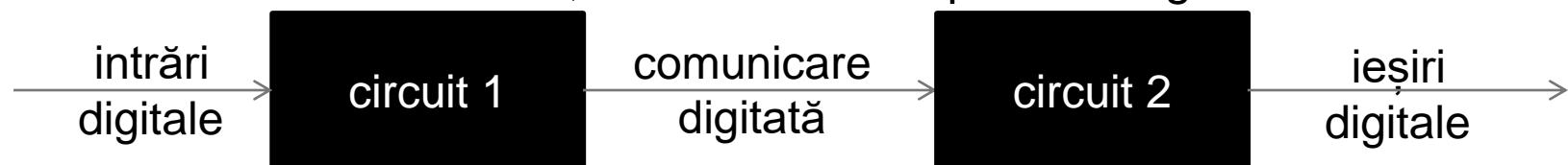


- o soluție puțin mai sofisticată: avem două limite
- de data asta: “0” este între 0V și 3V iar “1” este între 7V și 10V
- intervalul între 3V și 7V este un “no man’s land”
  - nu putem decide voltajul
  - așteptăm stabilizarea la o valoare  $< 3V$  sau  $> 7V$

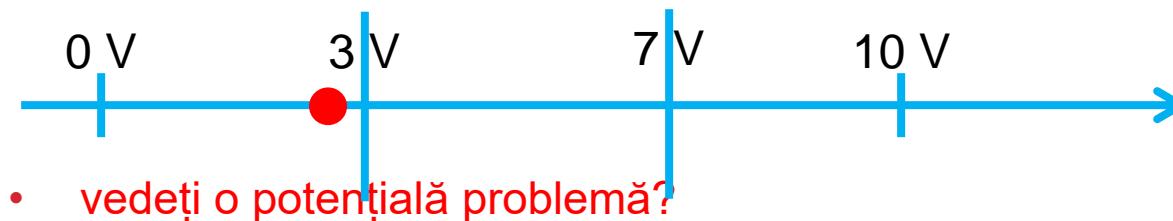
**Atenție:** sistemul nu este perfect, la 3V suntem “0” dar un zgomot de doar 4V din acest punct ne poate duce la 7V, deci “1”

# ABSTRACTIZAREA DIGITALĂ

- de la continuu la digital
  - în cele mai multe cazuri, vom conecta dispozitive digitale între ele



- de exemplu:
  - primul circuit scoate un “0”, dar la limita superioară



**Atenție:** sistemul nu este perfect, la 3V suntem “0” dar un zgomot de doar 4V din acest punct ne poate duce la 7V, deci “1”

# ABSTRACTIZAREA DIGITALĂ

- de la continuu la digital

- în cele mai multe cazuri, vom conecta dispozitive digitale între ele

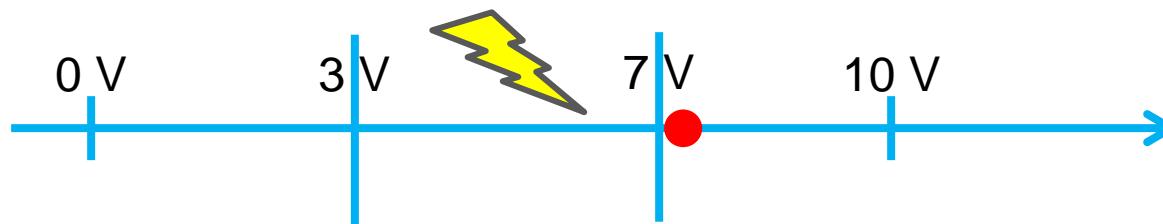


- de exemplu:

- primul circuit scoate un "0", dar la limita superioară



- semnalul este trimis către al doilea circuit, dar apare zgomot pe fir



**Atenție:** sistemul nu este perfect, la 3V suntem "0" dar un zgomot de doar 4V din acest punct ne poate duce la 7V, deci "1"

**Soluția?**

# ABSTRACTIZAREA DIGITALĂ

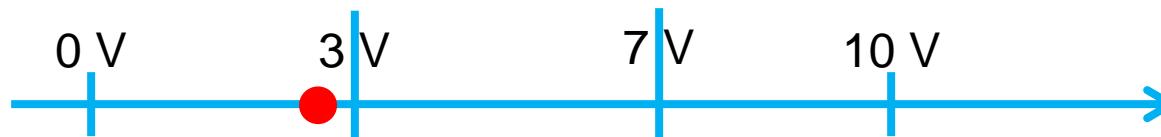
- de la continuu la digital

- în cele mai multe cazuri, vom conecta dispozitive digitale între ele

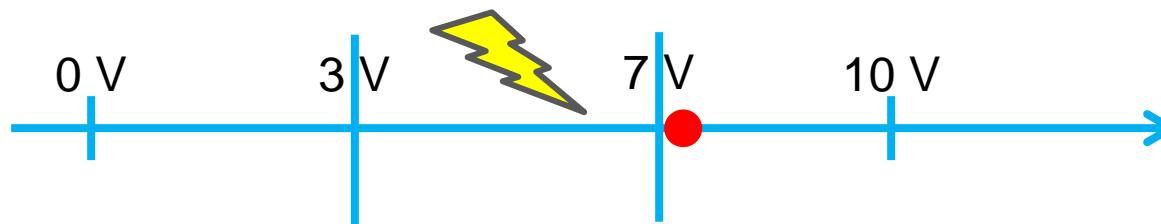


- de exemplu:

- primul circuit scoate un "0", dar la limita superioară



- semnalul este trimis către al doilea circuit, dar apare zgomot pe fir



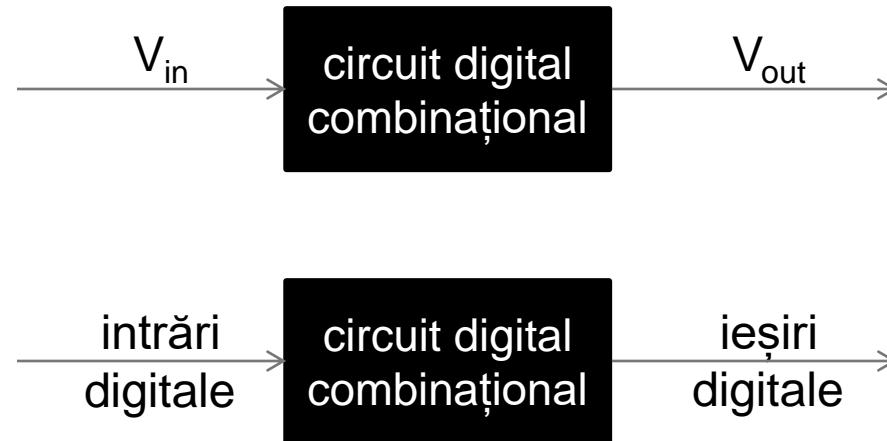
**Atenție:** sistemul nu este perfect, la 3V suntem "0" dar un zgomot de doar 4V din acest punct ne poate duce la 7V, deci "1"

**Soluția?** pentru ieșiri vom impune limite mai stricte (sub 2V, peste 8V)

ideal, am vrea ca limitele să fie cât mai înguste: 0V este "0" iar 10V este "1"

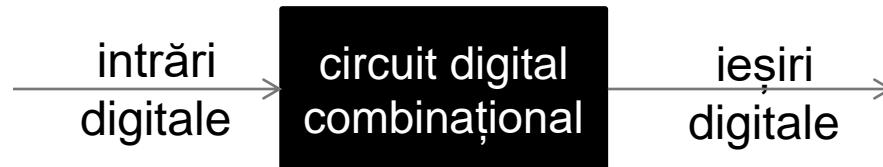
# CIRCUITE DIGITALE

- circuit digital combinational



# CIRCUITE DIGITALE

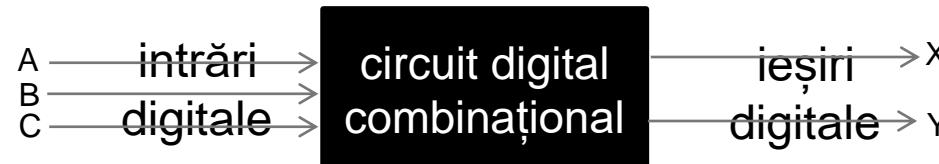
- circuit digital combinațional



- intrări digitale, pot fi multe
- ieșiri digitale, pot fi multe
- nu are stări interne
  - pui un semnal digital constant la intrare și ai un alt semnal digital constant la ieșire
  - dar nu poate “memora” nimic
  - nu are o “stare internă” (memorie)
- avem un **temp de propagare ( $t_p$ )**: timpul maxim necesar pentru a produce la ieșire semnale digitale corecte și valide din momentul în care la intrare s-au specificat semnale digitale corecte și valide
- de ce se numesc circuite combinaționale?
  - pentru că ieșire este o combinație (o funcție logică care combină) toate (sau o parte) a intrărilor

# CIRCUITE DIGITALE

- circuit digital combinațional



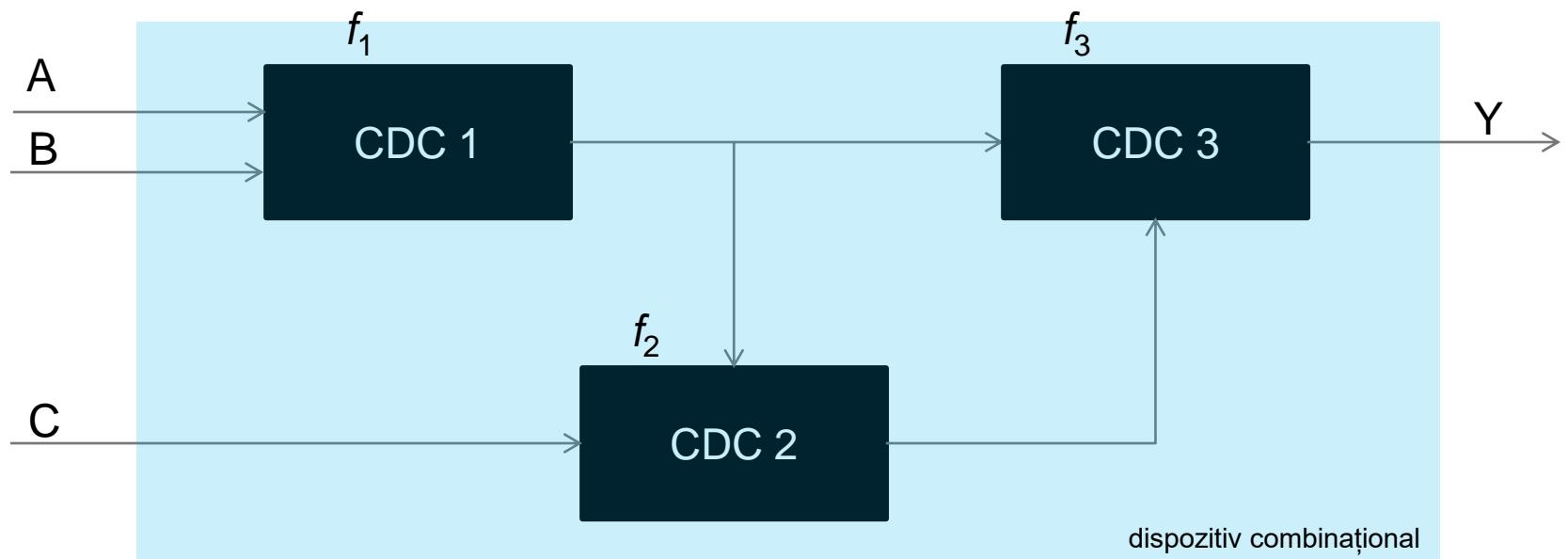
- de ce se numesc circuite combinaționale?
  - pentru că ieșire este o combinație (o funcție logică care combină) toate (sau o parte) a intrărilor
  - deci, pentru fiecare intrare, trebuie să știm care e ieșirea

A	B	C	X	Y
0	0	0	0	1
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	0

tabel de adevăr

# CIRCUITE DIGITALE

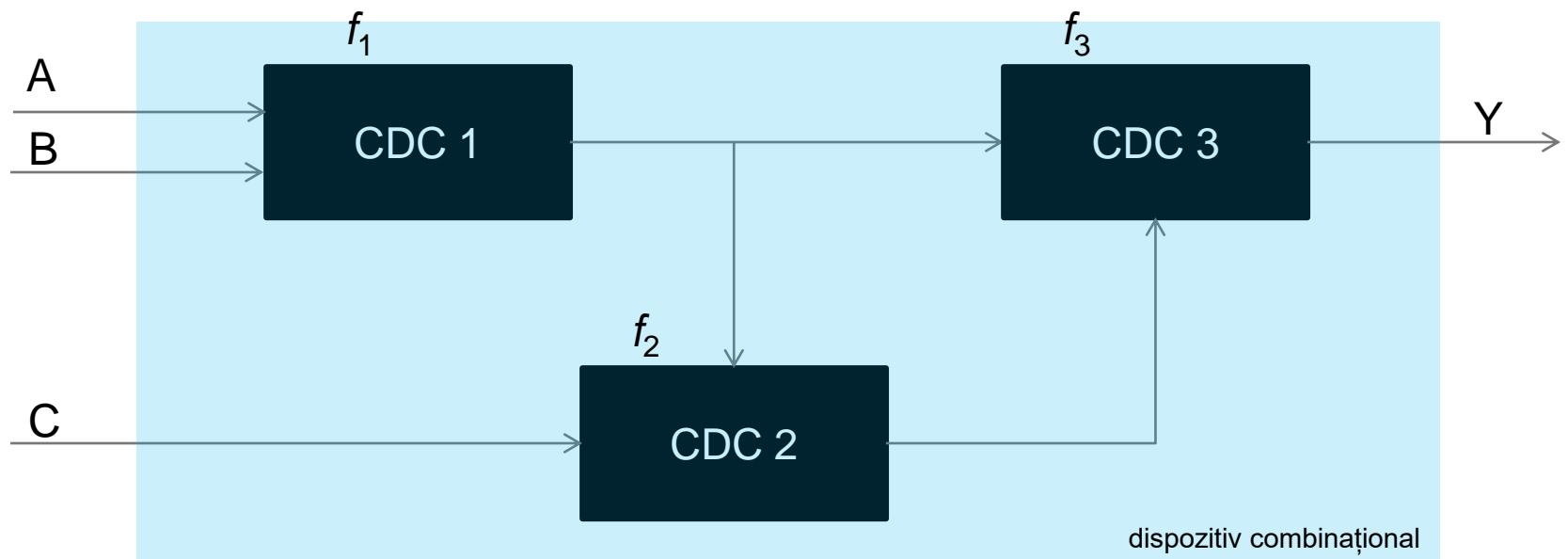
- **dispozitiv combinațional**
  - fiecare element este un circuit combinațional
  - fiecare intrare este conectată la exact o ieșire sau la o constantă
  - nu există niciun ciclu în graful direcțional al dispozitivului



- care este funcția dispozitivului?

# CIRCUITE DIGITALE

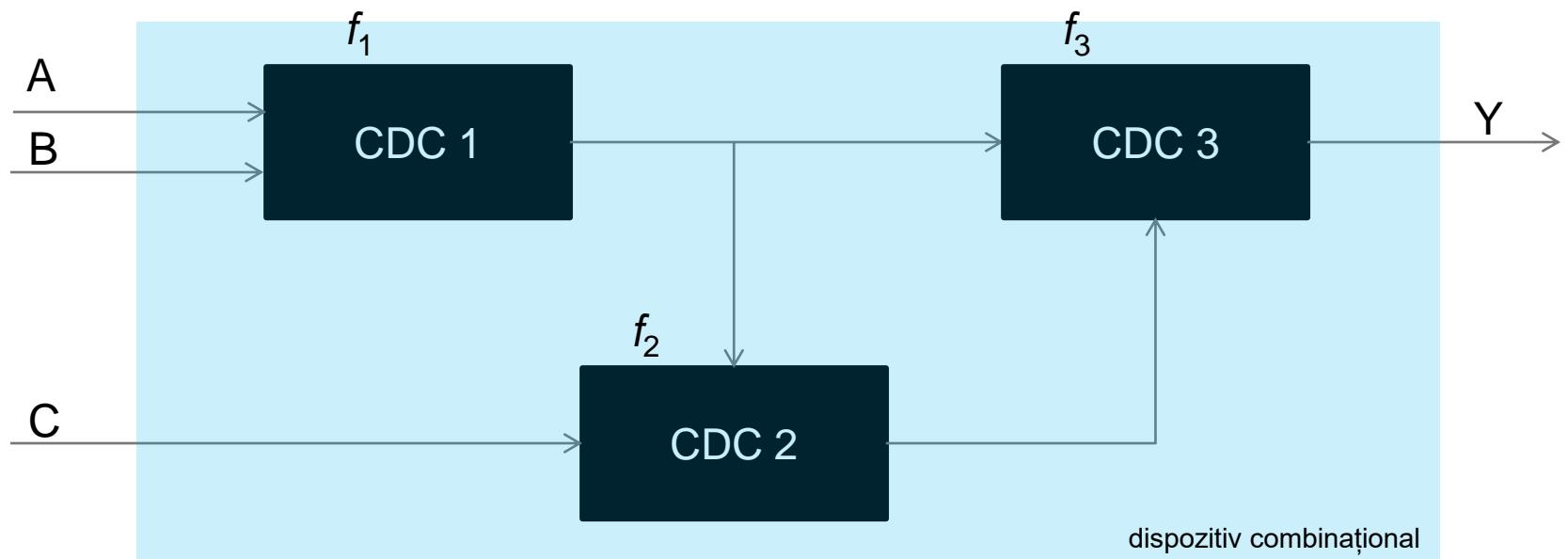
- **dispozitiv combinațional**
  - fiecare element este un circuit combinațional
  - fiecare intrare este conectată la exact o ieșire sau la o constantă
  - nu există niciun ciclu în graful direcțional al dispozitivului



- care este funcția dispozitivului?  $Y = f_3(f_1(A, B), f_2(f_1(A, B), C))$

# CIRCUITE DIGITALE

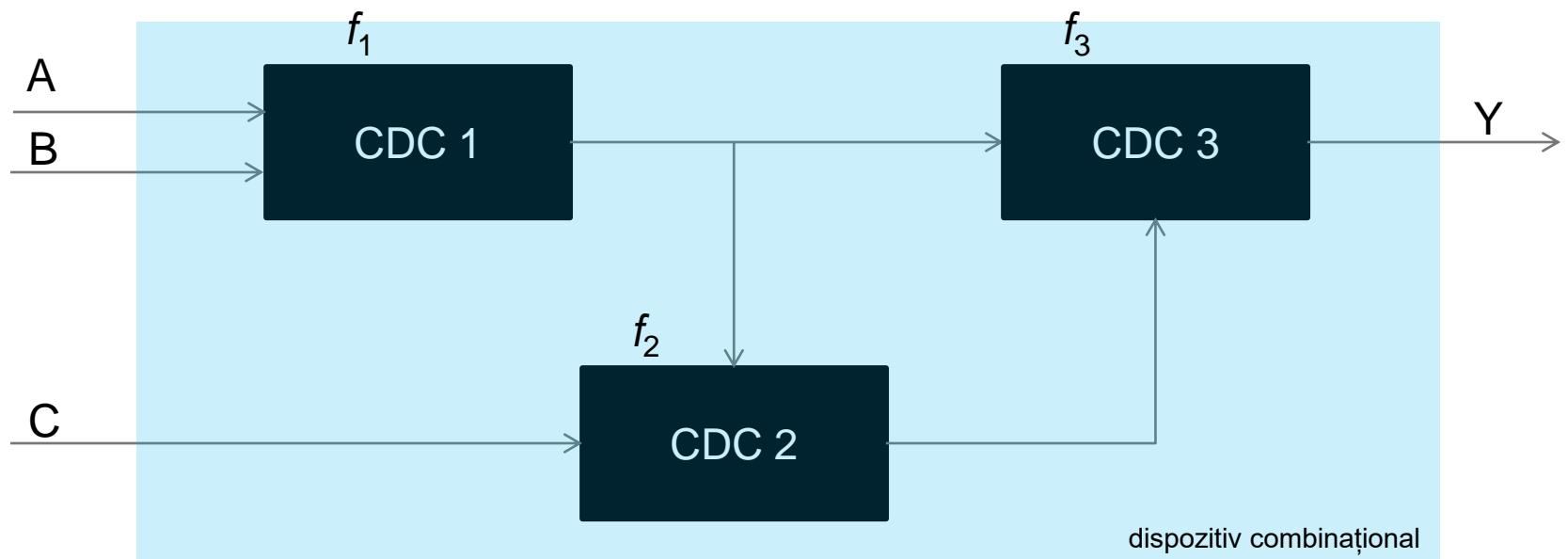
- **dispozitiv combinațional**
  - fiecare element este un circuit combinațional
  - fiecare intrare este conectată la exact o ieșire sau la o constantă
  - nu există niciun ciclu în graful direcțional al dispozitivului



- timpul total de propagare?

# CIRCUITE DIGITALE

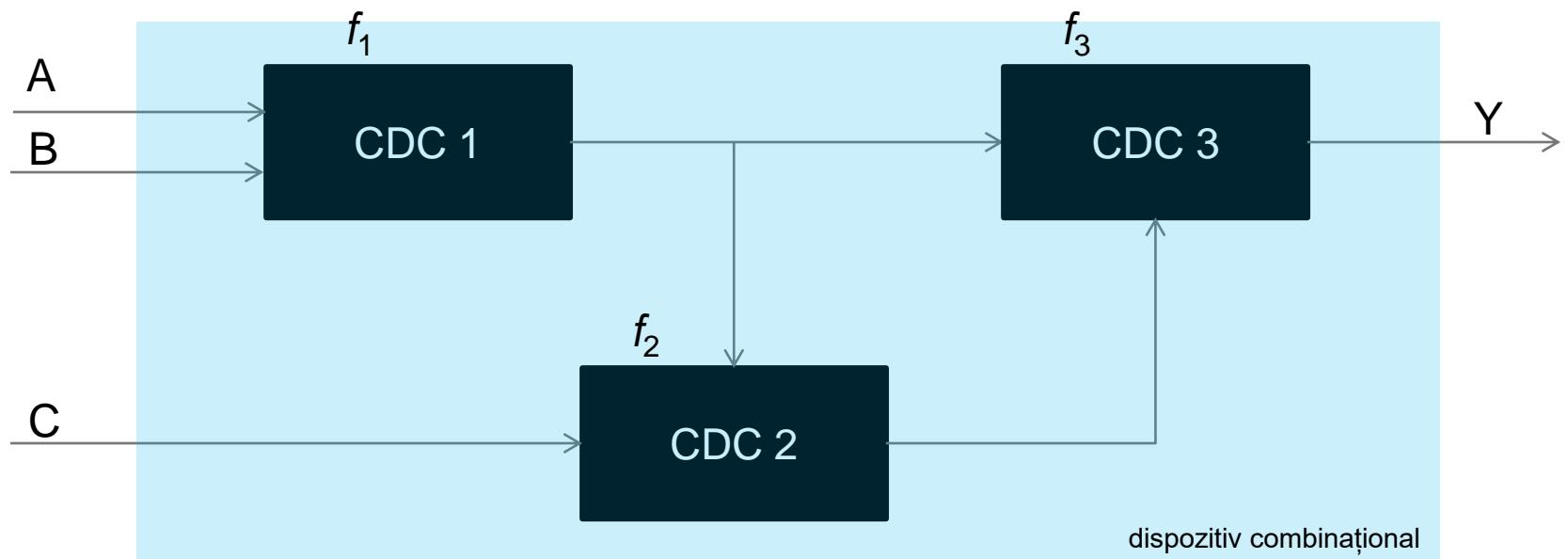
- **dispozitiv combinațional**
  - fiecare element este un circuit combinațional
  - fiecare intrare este conectată la exact o ieșire sau la o constantă
  - nu există niciun ciclu în graful direcțional al dispozitivului



- timpul total de propagare?  $t_{p,\text{total}} = t_{p,1} + t_{p,2} + t_{p,3}$  (longest path)

# CIRCUITE DIGITALE

- **dispozitiv combinațional**
  - fiecare element este un circuit combinațional
  - fiecare intrare este conectată la exact o ieșire sau la o constantă
  - nu există niciun ciclu în graful direcțional al dispozitivului

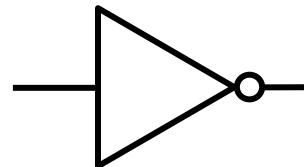


- timpul total de propagare?  $t_{p,\text{total}} = t_{p,1} + t_{p,2} + t_{p,3}$  (longest path)  
timpul maxim după care avem o ieșire validă dacă avem intrări valide

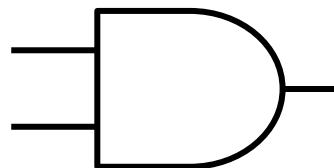
# CIRCUITE DIGITALE

- circuit digital combinational
  - exemple fondamentale

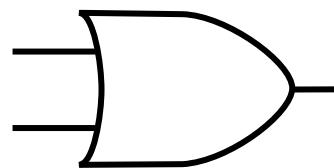
NOT



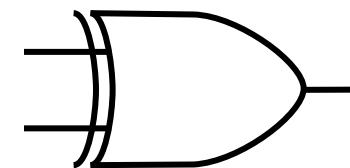
AND



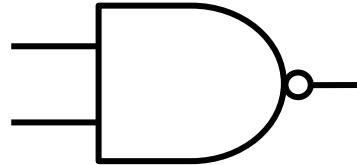
OR



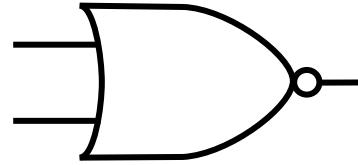
XOR



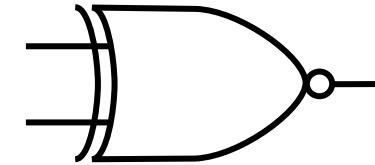
NAND



NOR

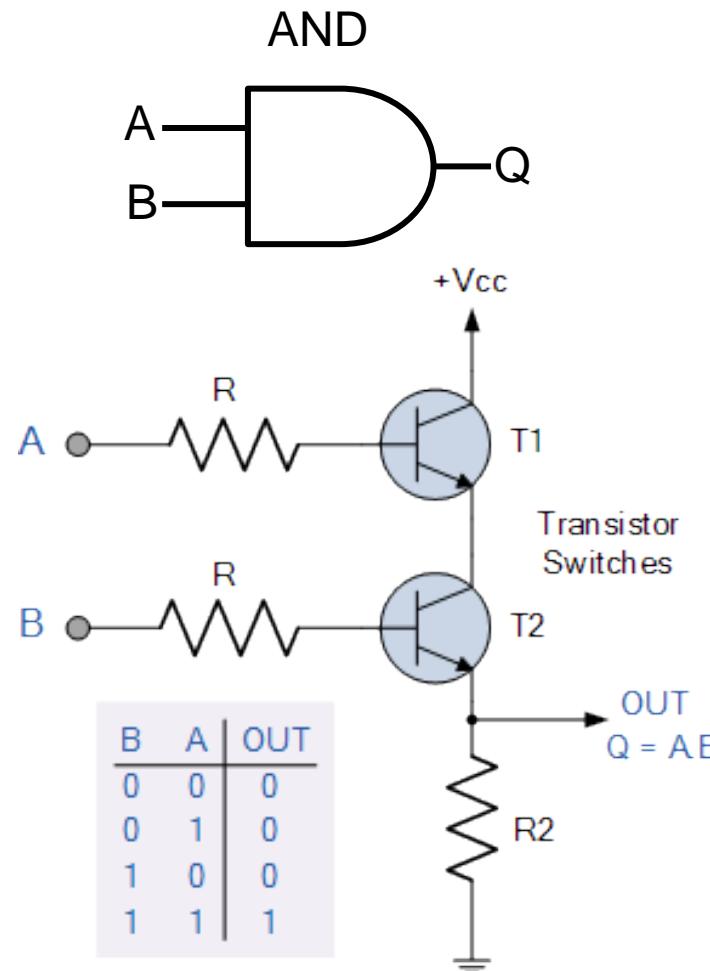


XNOR



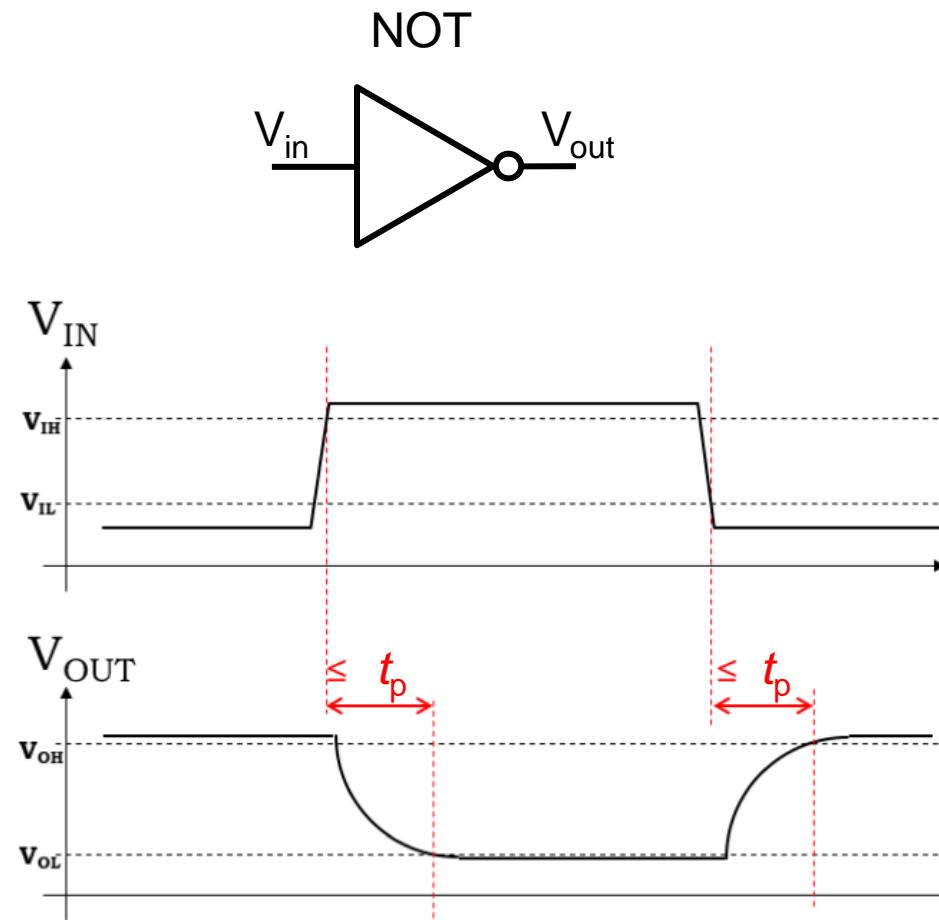
# CIRCUITE DIGITALE

- circuit digital combinational
  - exemple fundamentale
  - la baza tuturor se află circuite electronice bazate de tranzistor



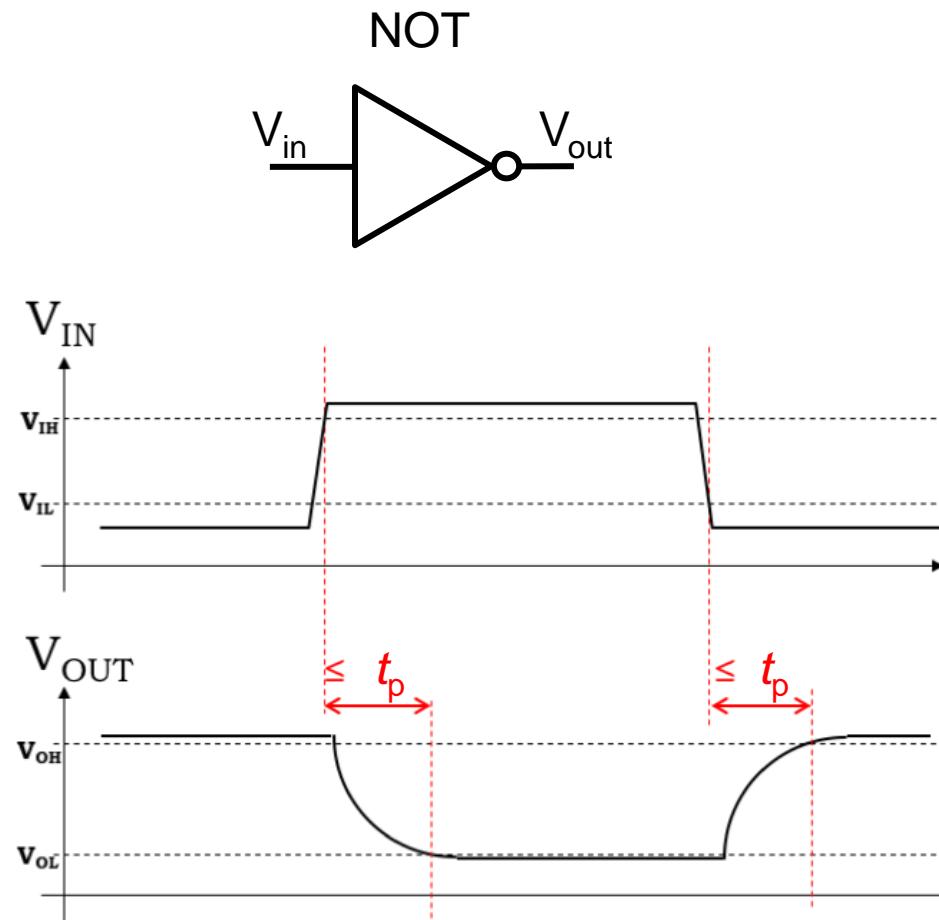
# CIRCUITE DIGITALE

- circuit digital combinational
  - exemple fundamentale: să nu uităm că totul este analogic



# CIRCUITE DIGITALE

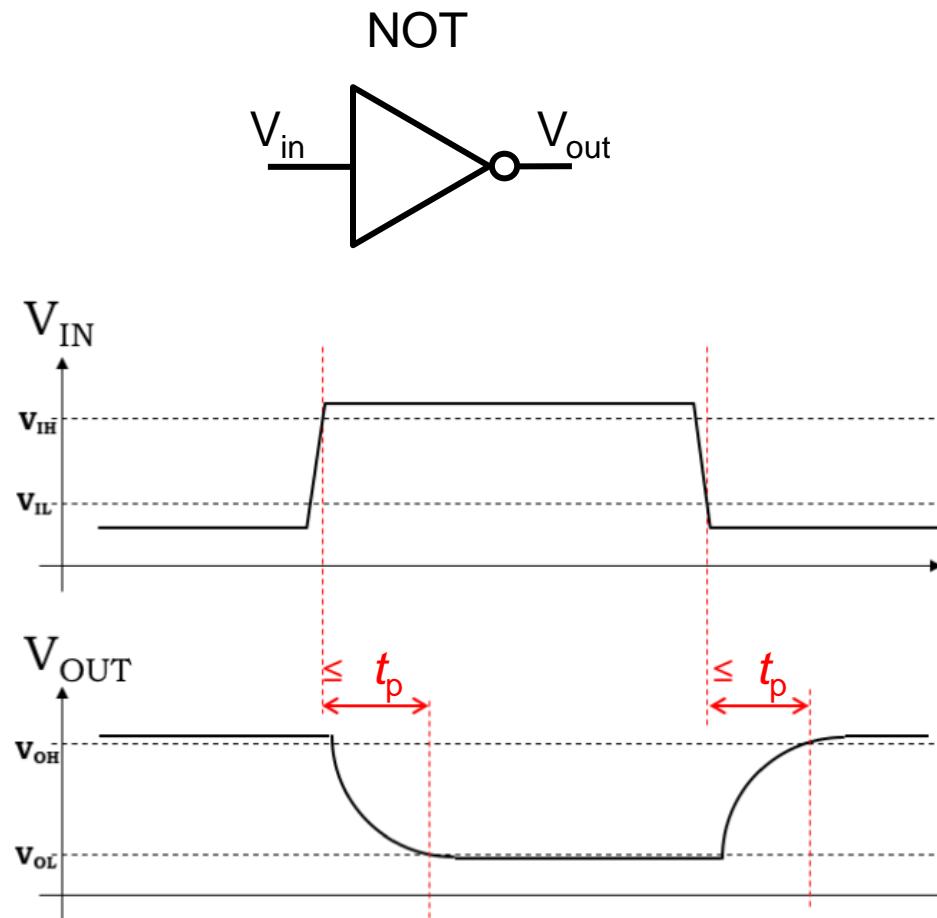
- circuit digital combinațional
  - exemple fundamentale: să nu uităm că totul este analogic



de ce este important acest  $t_p$  pentru arhitectura calculatoarelor?

# CIRCUITE DIGITALE

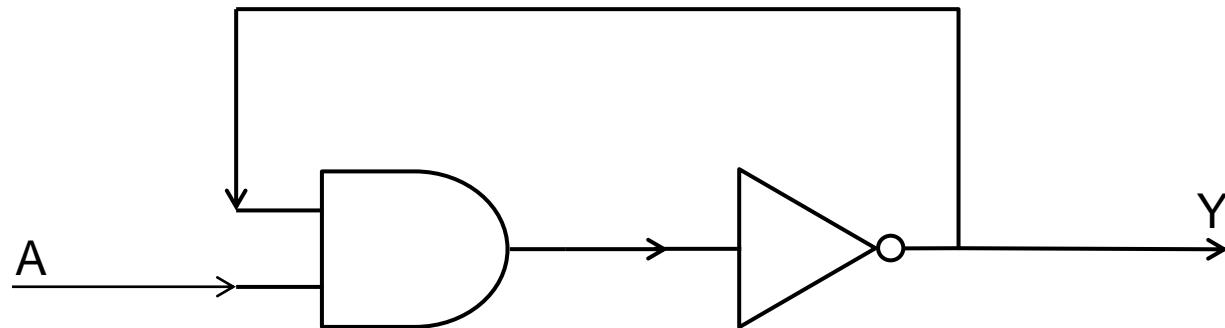
- circuit digital combinațional
  - exemple fundamentale: să nu uităm că totul este analogic



un computer care funcționează la 1GHz trimite comenzi o dată la 1ns

# CIRCUITE DIGITALE

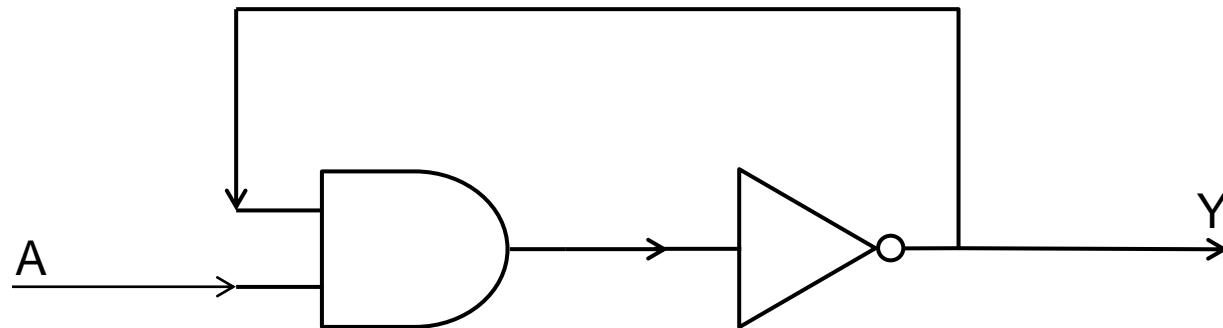
- circuit digital combinațional
  - ce se întâmplă dacă avem un ciclu în circuit?



- ce se întâmplă dacă  $A = 0$ ?
  - ce se întâmplă dacă  $A = 1$ ?
- (în ambele cazuri considerăm cealalaltă intrare în AND ca 0)

# CIRCUITE DIGITALE

- circuit digital combinațional
  - ce se întâmplă dacă avem un ciclu în circuit?



- ce se întâmplă dacă  $A = 0$ ?
  - ce se întâmplă dacă  $A = 1$ ?
- (în ambele cazuri considerăm cealalaltă intrare în AND ca 0)
- totuși, vom folosi cicluri pentru a crea stări în circuite digitale

# ABSTRACTIZAREA DIGITALĂ

- două întrebări importante:
  - de ce folosim semnale digitale în loc de analogice?
  - de ce folosim sistemul binar? ar fi mai avantajos să folosim hex?

# ABSTRACTIZAREA DIGITALĂ

- două întrebări importante:
  - de ce folosim semnale digitale în loc de analogice?
    - din cauza zgomotului
    - într-un sistem analogic zgomotul se acumulează
    - într-un sistem digital, avem corecțiile de zgomot (avem margini)
  - de ce folosim sistemul binar? ar fi mai avantajos să folosim hex?
    - da, ar fi mai avantajos să folosim hex (e de 4 ori mai avantajos)
    - problema este că în loc de două stări ar trebui acum să avem 16
    - asta înseamnă că trebuie să distingem 16 nivele de voltaj în prezența zgomotului (adică cu tot cu margini de zgomot)
    - probabil 16 nivele e prea mult ... dar probabil 4 nivele ar fi fezabil
    - dacă am avea 4 nivele (adică baza  $B = 4$ ) am fi de două ori mai eficienți

# CIRCUITE DIGITALE

- hex în media
  - The Martian Hexadecimal Scene,  
<https://www.youtube.com/watch?v=k-GH3mbvUro>
  - care e problema?
  - de ce folosește hex?
  - cum traduce din hex în litere?
  - ce face la sfârșit? (aparent scrie hex direct ca să programeze)

# CIRCUITE DIGITALE

- hex în media

- The Martian Hexadecimal Scene,  
<https://www.youtube.com/watch?v=k-GH3mbvUro>

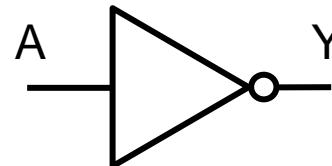
- care e problema? 27 de caractere sunt prea multe
- de ce folosește hex? pentru că în loc de 27 are doar 16 caractere
- cum traduce din hex în litere? [tabela ASCII](#)
- la sfârșit, scrie [cod mașină](#) (și noi vom face asta, puțin)

## ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	,	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

# PORȚILE LOGICE DE BAZĂ

- NOT

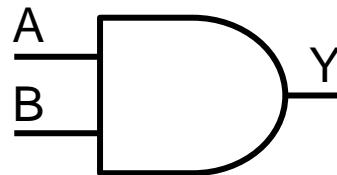


A	Y
0	1
1	0

- notația:
  - NOT A (explicit operația)
  - $\bar{A}$  (**A bar, A complement**)
  - $\neg A$  (notația din logică)
  - $\sim A$  (not A)
  - - A (minus A)
  - $A'$  (A prime, A complement)
  - $!A$  (bang A)

# PORȚILE LOGICE DE BAZĂ

- AND



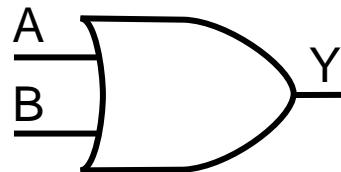
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

- notația:

- A AND B (explicit operația)
- $A \times B$  (înmulțire)
- $A \cdot B$  (înmulțire)
- $A * B$  (înmulțire)
- $A . B$  (înmulțire)
- **AB (înmulțire)**
- $A \wedge B$  (notația din logică)
- $A \& B$  (operația pe biți, C)
- $A \&\& B$  (operația logică, C)

# PORȚILE LOGICE DE BAZĂ

- OR

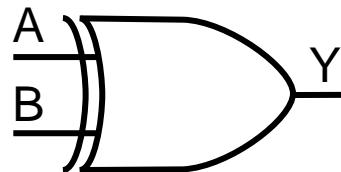


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

- notația:
  - A OR B (explicit operația)
  - **A + B (adunare)**
  - A v B (notația din logică)
  - A | B (operația pe biți, C)
  - A || B (operația logică, C)

# PORȚILE LOGICE DE BAZĂ

- XOR

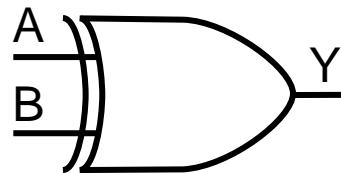


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

- notăția:
  - $A \text{ XOR } B$  (explicit operația)
  - $A \oplus B$  (**adunare XOR**)
  - $A \wedge B$  (operația pe biți, C)
  - $A \wedge\wedge B$  (există aşa ceva ???)

# PORȚILE LOGICE DE BAZĂ

- XOR



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

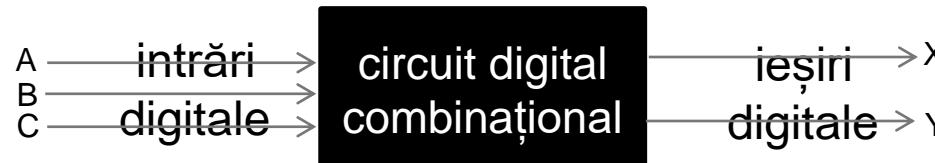
- notația:
  - $A \text{ XOR } B$  (explicit operația)
  - $A \oplus B$  (**adunare XOR**)
  - $A \wedge B$  (operația pe biți, C)
  - $A \neq B$  (operația logică, C: sau  $A \leftrightarrow B$ , sau  $A \sim B$ , sau ...)

# **ALGEBRĂ BOOLEANĂ**

- aveți un curs de Logică în acest semestru
- deci, din acest moment presupun că logica/algebra booleană este cunoscută de voi
- nu vom repeta materia de la logică, dar dacă este ceva foarte important vom trece rapid în revistă conceptele

# CIRCUIT DIGITAL COMBINAȚIONAL

- circuit digital combinațional



A	B	C	X	Y
0	0	0	0	1
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	0

- o expresie booleană care conține regulile din tabel?

- $X = \overline{ABC} + \overline{ABC} + \overline{ABC} + ABC$
- $Y = \dots$  (execuțiu pentru voi)

forma normală, suma de produse

# CIRCUIT DIGITAL COMBINAȚIONAL

- tabel de adevăr

A	B	C	X	Y
0	0	0	0	1
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	0

- expresia booleană echivalentă
  - $X = \overline{ABC} + \overline{AB}\bar{C} + A\overline{B}\bar{C} + ABC$
- **concluzia:** NOT, AND și OR sunt universale (pot implementa orice circuit combinațional)
- de ce lipsește XOR?

# CIRCUIT DIGITAL COMBINAȚIONAL

- tabel de adevăr

A	B	C	X	Y
0	0	0	0	1
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	0

- expresia booleană echivalentă
  - $X = \overline{ABC} + \overline{AB}\bar{C} + A\overline{B}\bar{C} + ABC$
- **concluzia:** NOT, AND și OR sunt universale (pot implementa orice circuit combinațional)
- de ce lipsește XOR?  $A \oplus B = \overline{AB} + A\overline{B}$

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

# CIRCUIT DIGITAL COMBINAȚIONAL

- tabel de adevăr

A	B	C	X	Y
0	0	0	0	1
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	0

- expresia booleană echivalentă
  - $X = \overline{ABC} + \overline{AB}\bar{C} + A\overline{B}\bar{C} + ABC$
- **o potențială problemă:** dacă avem N intrări, expresia pentru X depinde de un număr (potențial) mare de sume de produse
- **care e numărul maxim de termeni în sumele din X?**

# CIRCUIT DIGITAL COMBINAȚIONAL

- tabel de adevăr

A	B	C	X	Y
0	0	0	0	1
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	0

- expresia booleană echivalentă
  - $X = \overline{ABC} + \overline{AB}\bar{C} + A\overline{B}\bar{C} + ABC$
- **o potențială problemă:** dacă avem N intrări, expresia pentru X depinde de un număr (potențial) mare de sume de produse
- **care e numărul maxim de termeni în sumele din X?  $\frac{1}{2}2^N$**

# CIRCUIT DIGITAL COMBINAȚIONAL

- tabel de adevăr

A	B	C	X	Y
0	0	0	0	1
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	0

- expresia booleană echivalentă
  - $X = \overline{ABC} + \overline{AB}\bar{C} + A\overline{B}\bar{C} + ABC$
- **o potențială problemă:** dacă avem N intrări, expresia pentru X depinde de un număr (potențial) mare de sume de produse
- **soluția generală:** vrem reprezentări minime

# CIRCUIT DIGITAL COMBINAȚIONAL

- tabel de adevăr

A	B	C	X	Y
0	0	0	0	1
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	0

- expresia booleană echivalentă
  - $X = \overline{ABC} + \overline{AB}\bar{C} + A\overline{B}\bar{C} + ABC$
- reprezentarea minimă a lui X?

# CIRCUIT DIGITAL COMBINATORIAL

- tabel de adevăr

A	B	C	X	Y
0	0	0	0	1
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	0

- expresia booleană echivalentă

- $$X = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$$

=

# CIRCUIT DIGITAL COMBINATORIAL

- tabel de adevăr

A	B	C	X	Y
0	0	0	0	1
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	0

- expresia booleană echivalentă

- $$\begin{aligned} X &= \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC \\ &= \overline{A} (\overline{B}C + B\overline{C}) + A (\overline{B}\overline{C} + BC) \\ &= \end{aligned}$$

# CIRCUIT DIGITAL COMBINATORIAL

- tabel de adevăr

A	B	C	X	Y
0	0	0	0	1
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	0

- expresia booleană echivalentă

- $$\begin{aligned} X &= \overline{ABC} + \overline{AB}\bar{C} + A\overline{B}\bar{C} + ABC \\ &= \overline{A} (\overline{BC} + B\bar{C}) + A (\overline{B}\bar{C} + BC) \\ &= \overline{A} (B \oplus C) + A \overline{(B \oplus C)} \end{aligned}$$

# CIRCUIT DIGITAL COMBINATORIAL

- tabel de adevăr

A	B	C	X	Y
0	0	0	0	1
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	0

- expresia booleană echivalentă

$$\begin{aligned} X &= \overline{ABC} + \overline{AB}\bar{C} + A\overline{B}\bar{C} + ABC \\ &= \overline{A} (\overline{BC} + B\bar{C}) + A (\overline{B}\bar{C} + BC) \\ &= \overline{A} (B \oplus C) + A \overline{(B \oplus C)} \\ &= A \oplus B \oplus C \end{aligned}$$

# **CE AM FĂCUT ASTĂZI**

- abstractizarea digitală
- discuție binary vs hex
- circuite digitale
- circuite combinaționale
- am văzut un singur exemplu de circuit pe bază de tranzistor

# **DATA VIITOARE ...**

- mai mult despre circuite de bază
- logică booleană
- sinteză logică
- circuite mai complexe
- circuite secvențiale

# LECTURĂ SUPLIMENTARĂ

- PH book
  - Appendix B
- Computerphile, Why Use Binary?,  
<https://www.youtube.com/watch?v=thrX3SBEpL8>
- Real Engineering, Transistors - The Invention That Changed The World, <https://www.youtube.com/watch?v=OwS9aTE2Go4>
- Ben Eater, Making logic gates from transistors,  
<https://www.youtube.com/watch?v=sTu3LwpF6XI>
- DrPhysicsA, An Introduction to Logic Gates,  
<https://www.youtube.com/watch?v=95kv5BF2Z9E>



# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x04**

**CIRCUITE COMBINAȚIONALE ȘI  
SECVENTIALE**

Cristian Rusu

# **DATA TRECUTĂ**

- abstractizarea digitală
- circuite digitale
- circuite combinaționale
- am văzut un singur exemplu de circuit pe bază de tranzistor
- discuție binary vs hex

# CUPRINS

- simplificări logice
- circuit de adunare
- exemple de circuite combinaționale și secvențiale
- referințe bibliografice

# CIRCUIT DIGITAL COMBINATORIAL

- tabel de adevăr

A	B	C	X
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	1

- $X = ?$

# CIRCUIT DIGITAL COMBINATORIAL

- tabel de adevăr

A	B	C	X
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	1

- $X = A\bar{B}\bar{C} + AB\bar{C} + \bar{A}BC + ABC$

=

# CIRCUIT DIGITAL COMBINATORIAL

- tabel de adevăr

A	B	C	X
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	1

- $$\begin{aligned} X &= A\bar{B}\bar{C} + AB\bar{C} + \bar{A}BC + ABC \\ &= A\bar{C}(\bar{B} + B) + (\bar{A} + A)BC \\ &= \end{aligned}$$

# CIRCUIT DIGITAL COMBINATORIAL

- tabel de adevăr

A	B	C	X
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	1

- $$\begin{aligned} X &= A\bar{B}\bar{C} + AB\bar{C} + \bar{A}BC + ABC \\ &= A\bar{C}(\bar{B} + B) + (\bar{A} + A)BC \\ &= A\bar{C} + BC \end{aligned}$$

# CIRCUIT DIGITAL COMBINATORIAL

- tabel de adevăr alternativ: reducerea cu wildcard

A	B	C	X
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	1



A	B	C	X
0	?	0	0
1	?	0	1
?	0	1	0
?	1	1	1

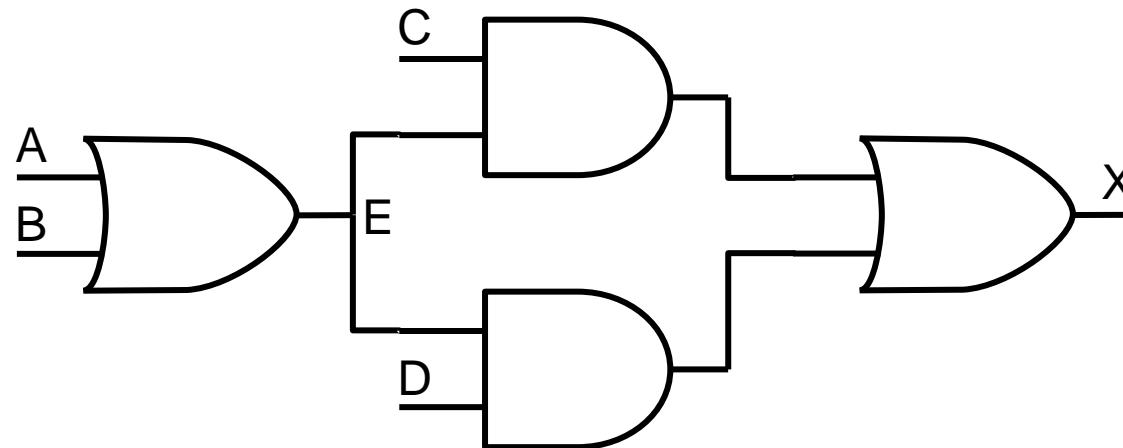
- $$X = A\bar{C} + BC$$

# CIRCUIT DIGITAL COMBINATORIAL

- considerăm acum
  - $X = AC + BC + AD + BD$   
=

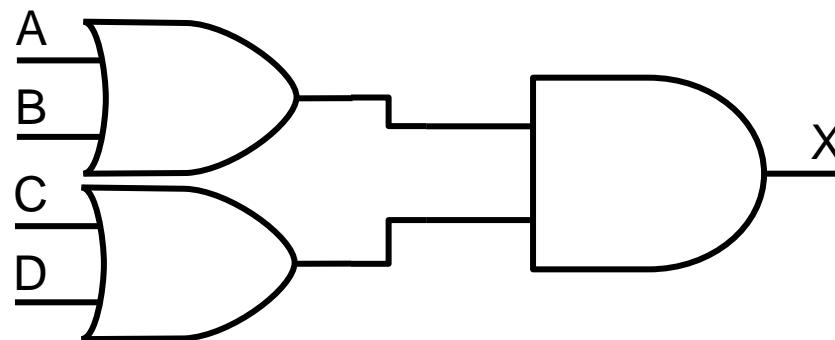
# CIRCUIT DIGITAL COMBINATORIAL

- considerăm acum
  - $$\begin{aligned} X &= AC + BC + AD + BD \\ &= (A + B)C + (A + B)D \\ &= EC + ED \end{aligned}$$

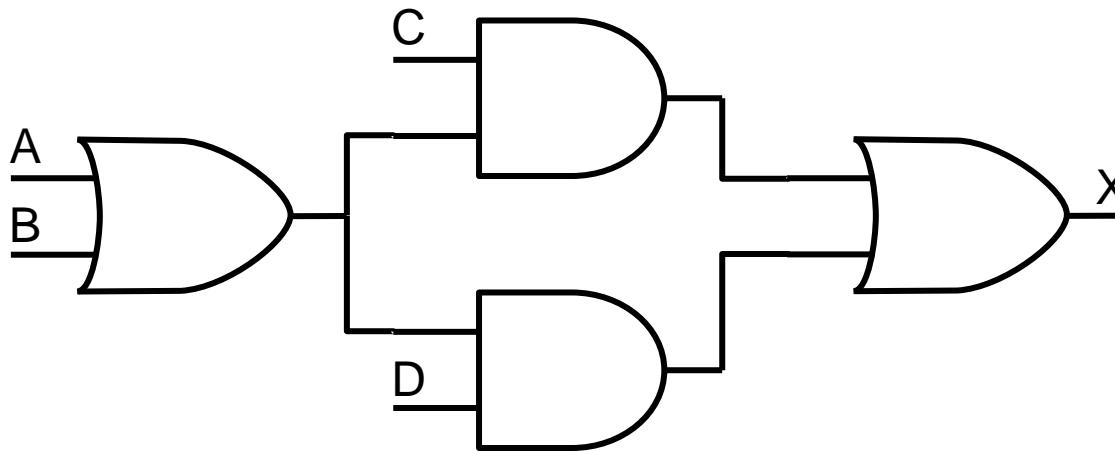


# CIRCUIT DIGITAL COMBINATORIAL

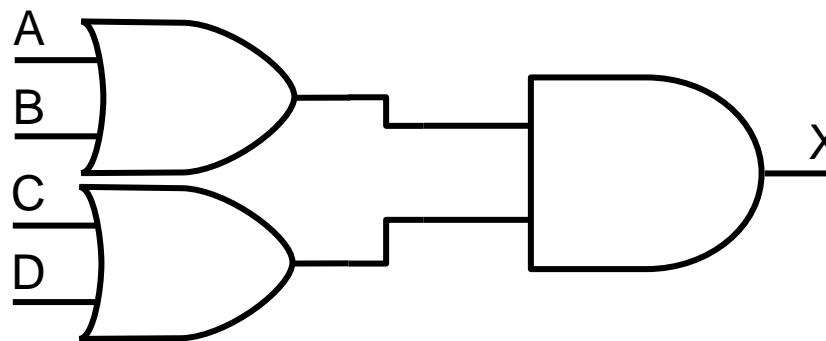
- considerăm acum
  - $$\begin{aligned} X &= AC + BC + AD + BD \\ &= (A + B)C + (A + B)D \\ &= (A + B)(C + D) \end{aligned}$$



# CIRCUIT DIGITAL COMBINAȚIONAL



- **versus**



În general, e bine să mergem până la capătul calculelor de grupare  
dar, rețineti că ne interesează și timpul de propagare

număr porti vs. stagii de calcul vs. tipul de porti ...

# CIRCUIT DIGITAL COMBINAȚIONAL

- nu toate circuitele sunt la fel de eficiente

poartă	întârziere (ps)	suprafață ( $\mu\text{m}^2$ )
AND-2	50	25
NAND-2	30	15
OR-2	55	26
NOR-2	35	16
AND-4	90	40
NAND-4	70	30
OR-4	100	42
NOR-4	80	32

- ce observați?

# CIRCUIT DIGITAL COMBINAȚIONAL

- nu toate circuitele sunt la fel de eficiente

poartă	întârziere (ps)	suprafață ( $\mu\text{m}^2$ )
AND-2	50	25
NAND-2	30	15
OR-2	55	26
NOR-2	35	16
AND-4	90	40
NAND-4	70	30
OR-4	100	42
NOR-4	80	32

- ce observați?

- $N^*$ -? sunt mai rapide și mai mici ca suprafață decât  $*^-$ ?
- $*^-2$  sunt mai rapide și mai mici ca suprafață decât  $*^-4$

**motivul:** unele tehnologii sunt bazate pe tranzistoare CMOS, iar circuitele analogice sunt mai eficiente pe logică negată (mai puține componente electrice)

# CIRCUITE MAI COMPLEXE

- **concluzia importantă:**
- **tot ce facem pe un sistem de calcul trebuie să se reducă la circuite care sunt porti logice**
- **altceva nu există la acest nivel**

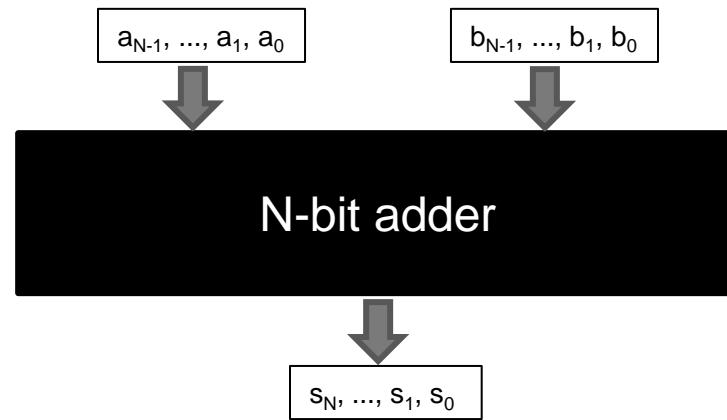
# CIRCUITE MAI COMPLEXE

- circuit de adunare
  - se dau  $a$  și  $b$  pe  $N$  biți
  - vrem să calculăm  $s = a + b$ 
    - pe câți biți este  $s$ ?
    -

# CIRCUITE MAI COMPLEXE

- **circuit de adunare**

- se dau  $a$  și  $b$  pe  $N$  biți
- vrem să calculăm  $s = a + b$ 
  - pe câți biți este  $s$ ?
  - $N+1$  biți

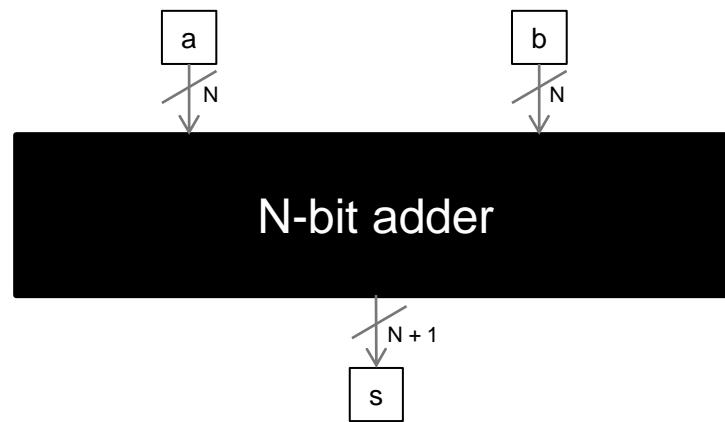


- câte intrări avem?
- câte ieșiri?
- cât de mare va fi circuitul combinațional?

# CIRCUITE MAI COMPLEXE

- circuit de adunare

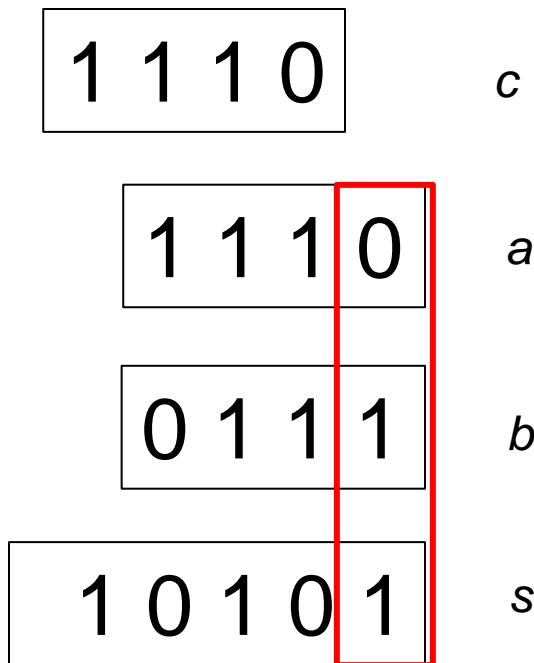
- se dau  $a$  și  $b$  pe  $N$  biți
- vrem să calculăm  $s = a + b$ 
  - pe câți biți este  $s$ ?
  - $N+1$  biți



- câte intrări avem?  $2N$
- câte ieșiri?  $N + 1$
- cât de mare va fi circuitul combinațional?  $(N+1)2^{2N-1}$

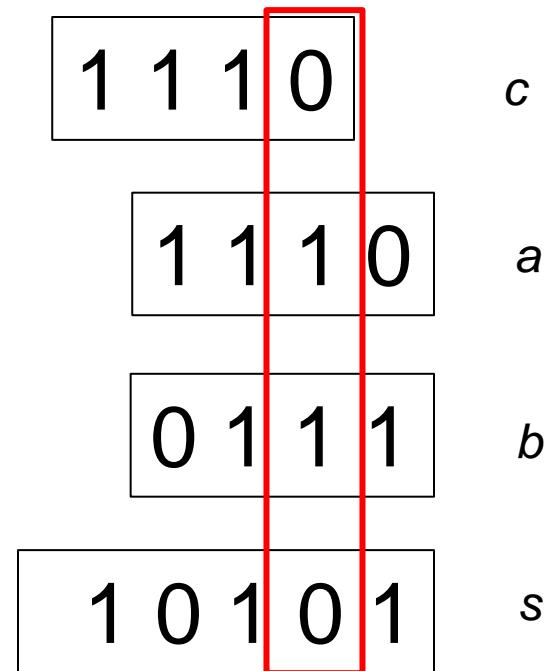
# CIRCUITE MAI COMPLEXE

- circuit de adunare
  - cum am făcut până acum adunarea binară?
    - $s = a + b$
    - exemplu:



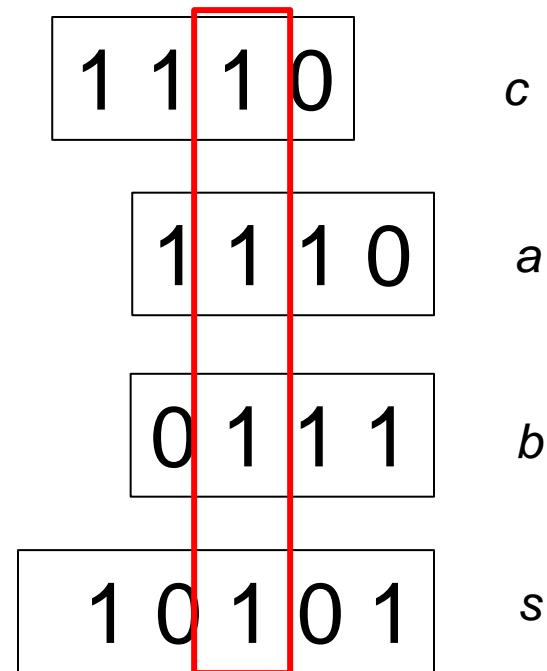
# CIRCUITE MAI COMPLEXE

- circuit de adunare
  - cum am făcut până acum adunarea binară?
    - $s = a + b$
    - exemplu:



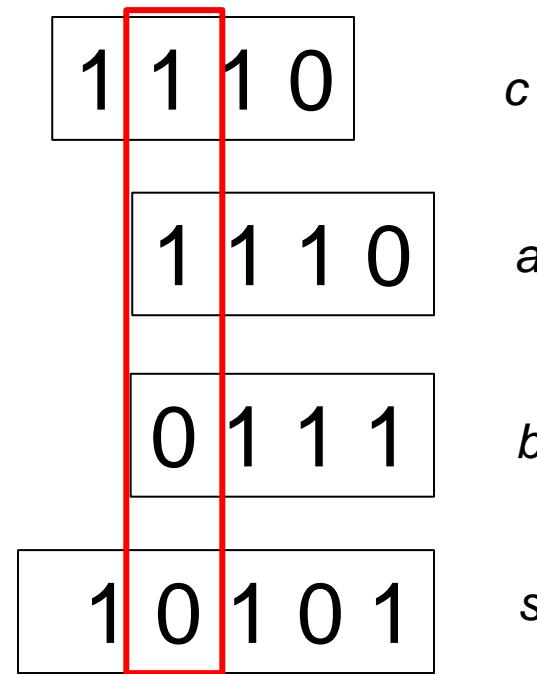
# CIRCUITE MAI COMPLEXE

- circuit de adunare
  - cum am făcut până acum adunarea binară?
    - $s = a + b$
    - exemplu:



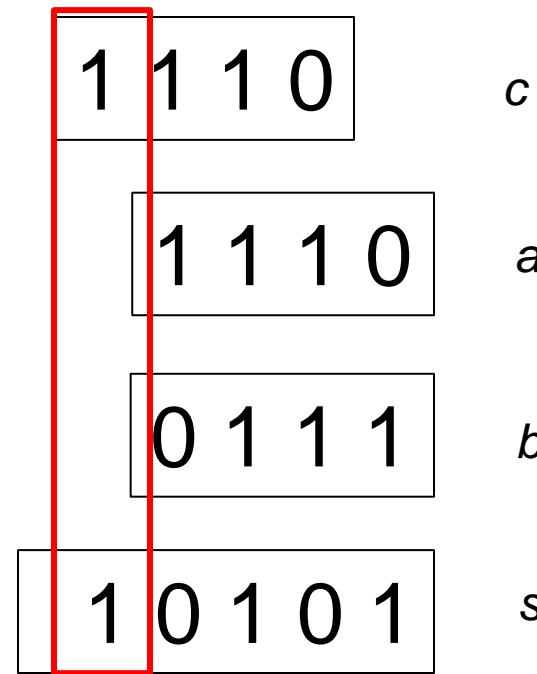
# CIRCUITE MAI COMPLEXE

- circuit de adunare
  - cum am făcut până acum adunarea binară?
    - $s = a + b$
    - exemplu:



# CIRCUITE MAI COMPLEXE

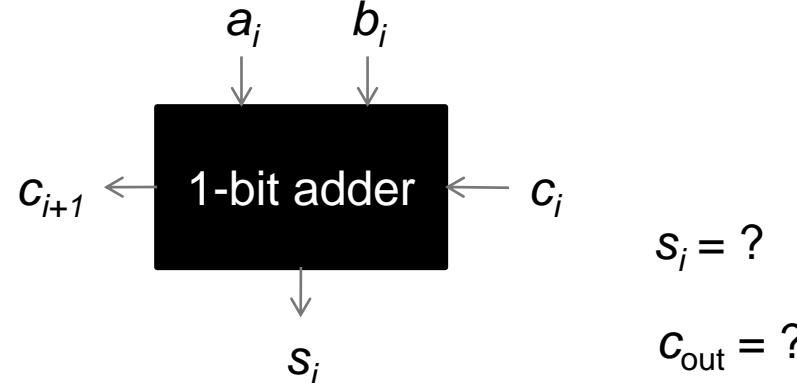
- circuit de adunare
  - cum am făcut până acum adunarea binară?
    - $s = a + b$
    - exemplu:



# CIRCUITE MAI COMPLEXE

- circuit de adunare

- ideea: definim un circuit bloc fundamental pe care bazăm totul

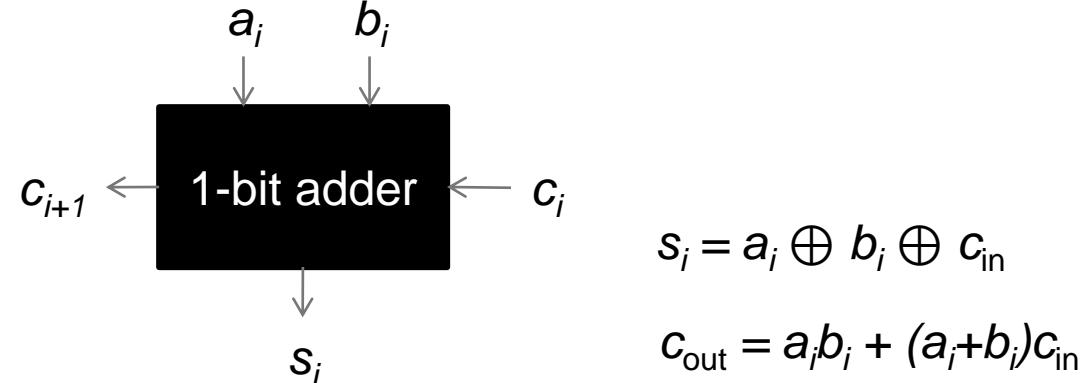


$a_i$	$b_i$	$c_{in}$	$s_i$	$c_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

# CIRCUITE MAI COMPLEXE

- circuit de adunare

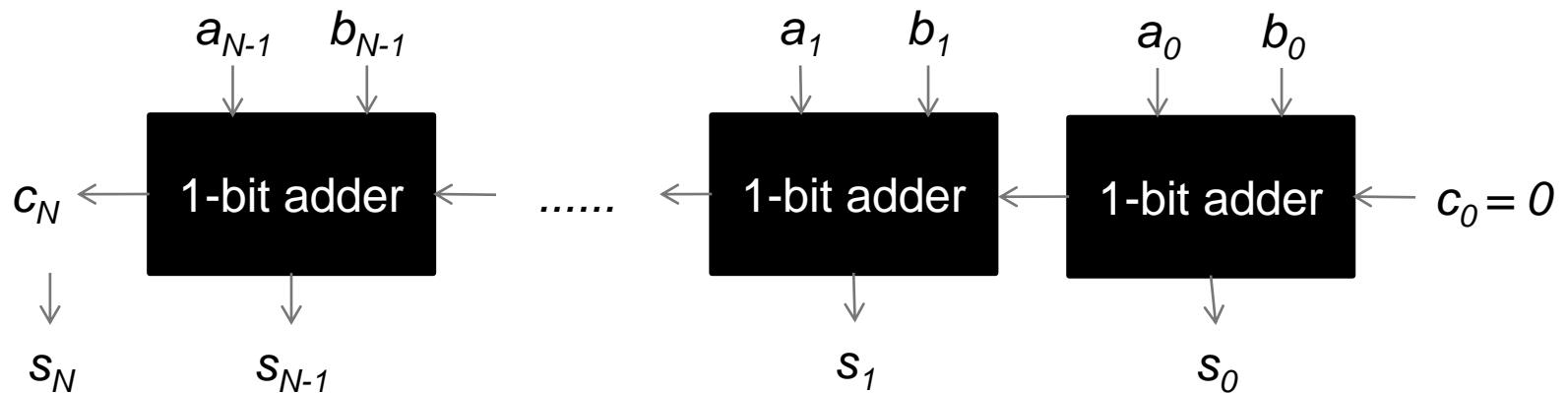
- ideea: definim un circuit bloc fundamental pe care bazăm totul



a <sub>i</sub>	b <sub>i</sub>	c <sub>in</sub>	s <sub>i</sub>	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

# CIRCUITE MAI COMPLEXE

- circuit de adunare
  - ideea: definim un circuit bloc fundamental pe care bazăm totul
  - ideea: folosim circuitul fundamental în cascadă



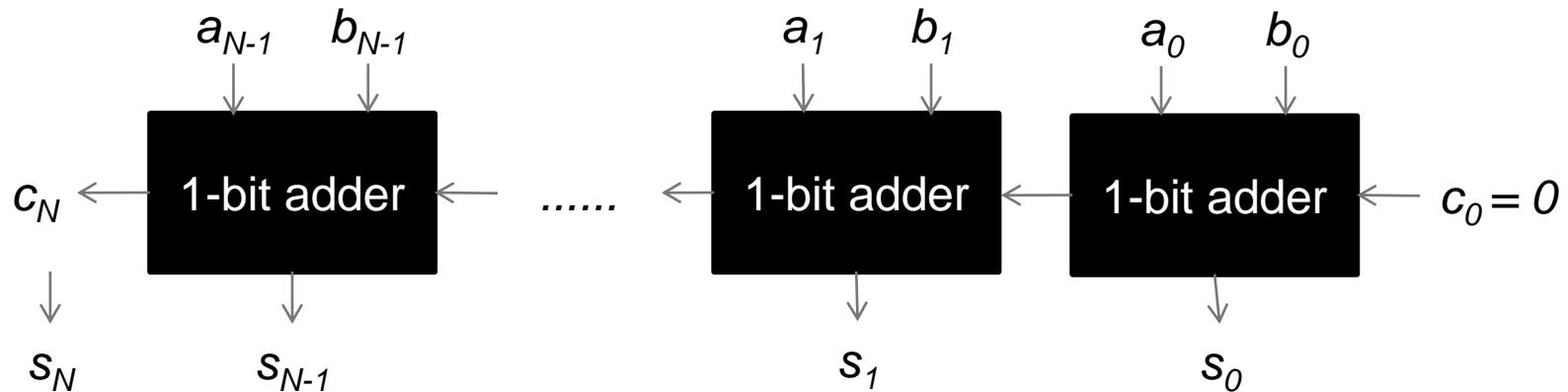
$$s_i = a_i \oplus b_i \oplus c_{\text{in}}$$

$$c_{\text{out}} = a_i b_i + (a_i + b_i) c_{\text{in}}$$

# CIRCUITE MAI COMPLEXE

- **circuit de adunare**

- **ideea:** definim un circuit bloc fundamental pe care bazăm totul
- **ideea:** folosim circuitul fundamental în cascadă



$$s_i = a_i \oplus b_i \oplus c_{\text{in}}$$

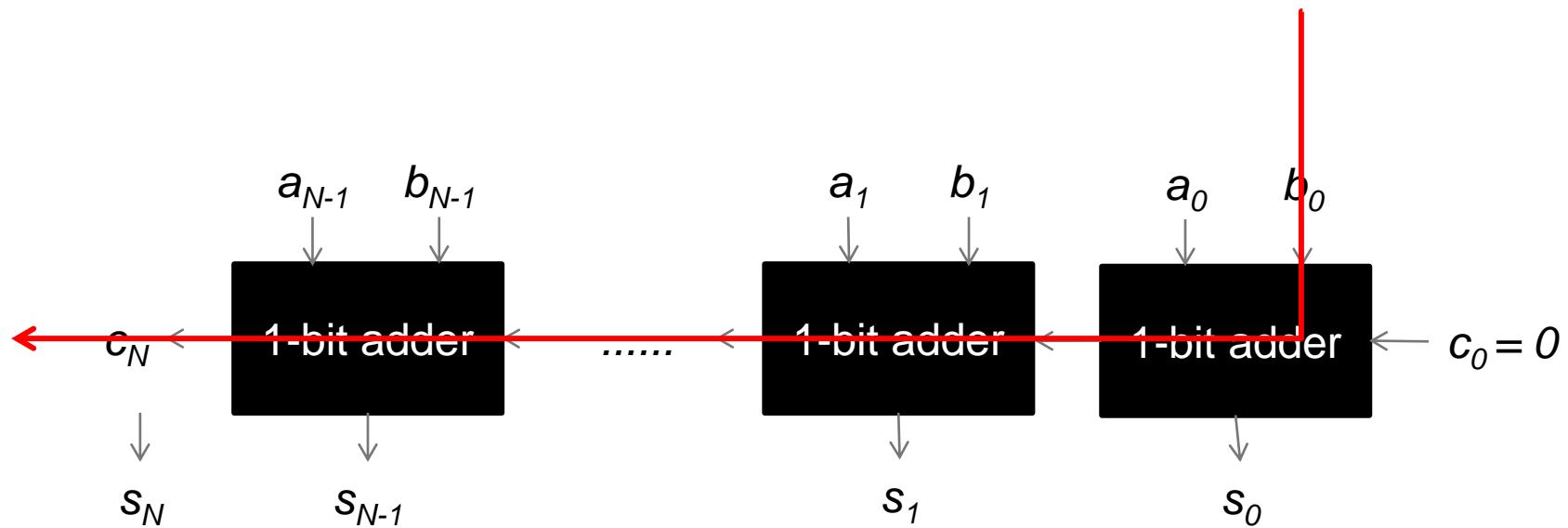
- data trecută am vorbit de  $t_P$
- cât este timpul de propagare în acest caz?

$$c_{\text{out}} = a_i b_i + (a_i + b_i) c_{\text{in}}$$

# CIRCUITE MAI COMPLEXE

- circuit de adunare

- ideea: definim un circuit bloc fundamental pe care bazăm totul
- ideea: folosim circuitul fundamental în cascadă



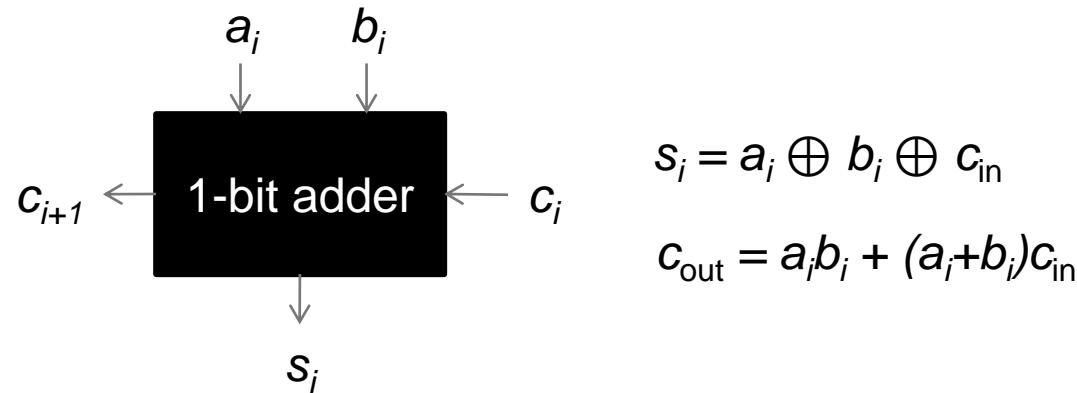
$$s_i = a_i \oplus b_i \oplus c_{\text{in}}$$

- data trecută am vorbit de  $t_P$
- cât este timpul de propagare în acest caz?
- $N t_{P, \text{ 1-bit adder}}$

$$c_{\text{out}} = a_i b_i + (a_i + b_i) c_{\text{in}}$$

# CIRCUITE MAI COMPLEXE

- circuit de adunare
  - care este problema fundamentală în circuitul de mai sus?
    - trebuie să așteptăm să putem calcula acei biți de carry

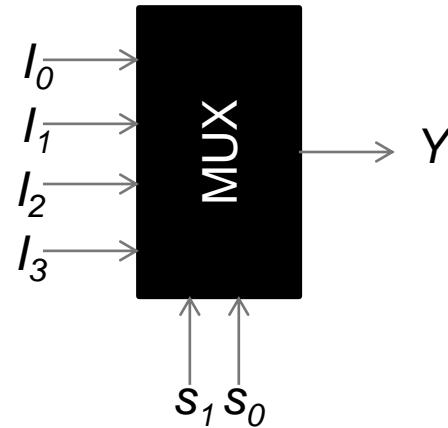


- notăm  $g_i = a_i b_i$  și  $p_i = a_i + b_i$ 
  - dacă  $g_i = 1$  atunci  $c_{\text{out}} = 1$
  - dacă  $g_i = 0$  și  $p_i = 0$  atunci  $c_{\text{out}} = 0$
  - dacă  $g_i = 0$  și  $p_i = 1$  atunci  $c_{\text{out}} = c_{\text{in}}$

# CIRCUITE MAI COMPLEXE

- **multiplexare**

- un circuit care selectează un semnal digital de la intrare pe baza unui semnal de activare  $s$



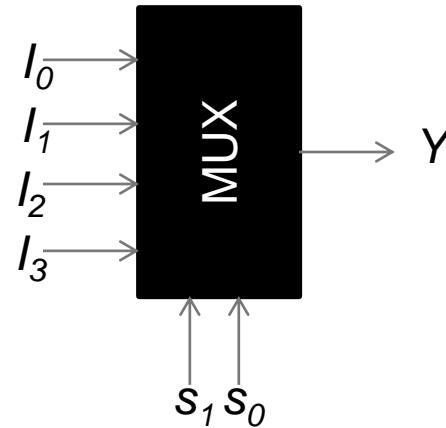
$s_1$	$s_0$	Y
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

- de ce e folositor acest circuit?

# CIRCUITE MAI COMPLEXE

- **multiplexare**

- un circuit care selectează un semnal digital de la intrare pe baza unui semnal de activare  $s$

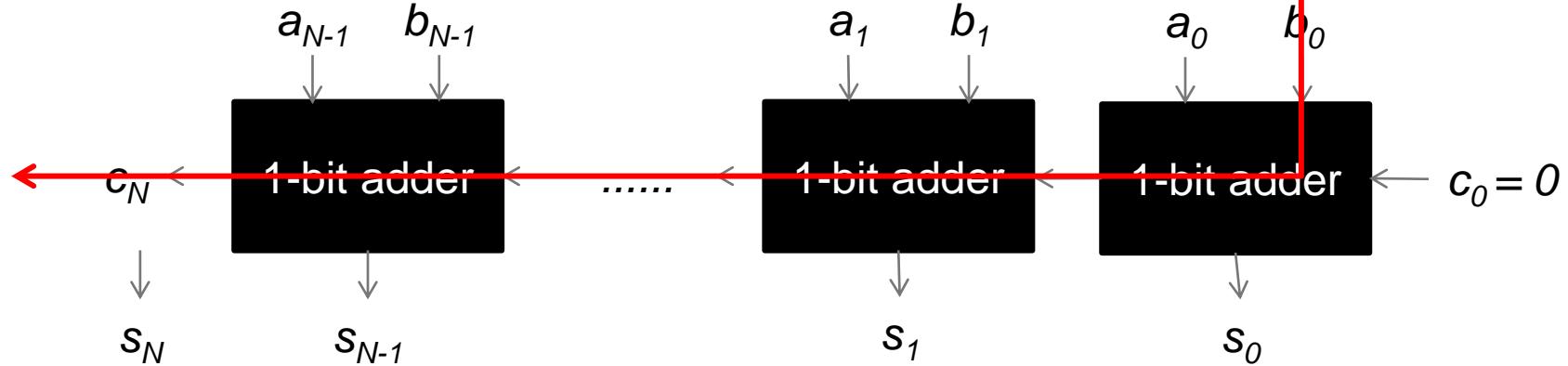


$s_1$	$s_0$	Y
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

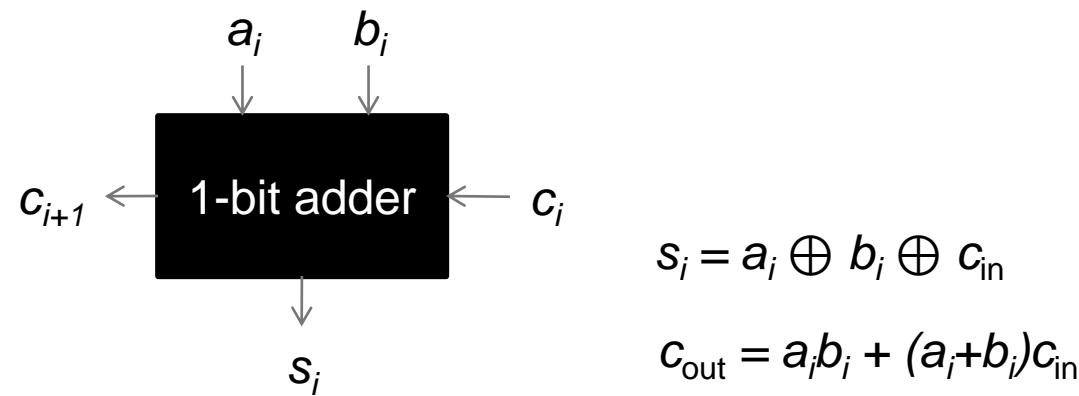
- de ce e folositor acest circuit?
  - implementare hardware pentru “if” și “case”
  - implementare hardware pentru operații shift
  - vom vedea la seminar

# CIRCUITE MAI COMPLEXE

- circuit de adunare

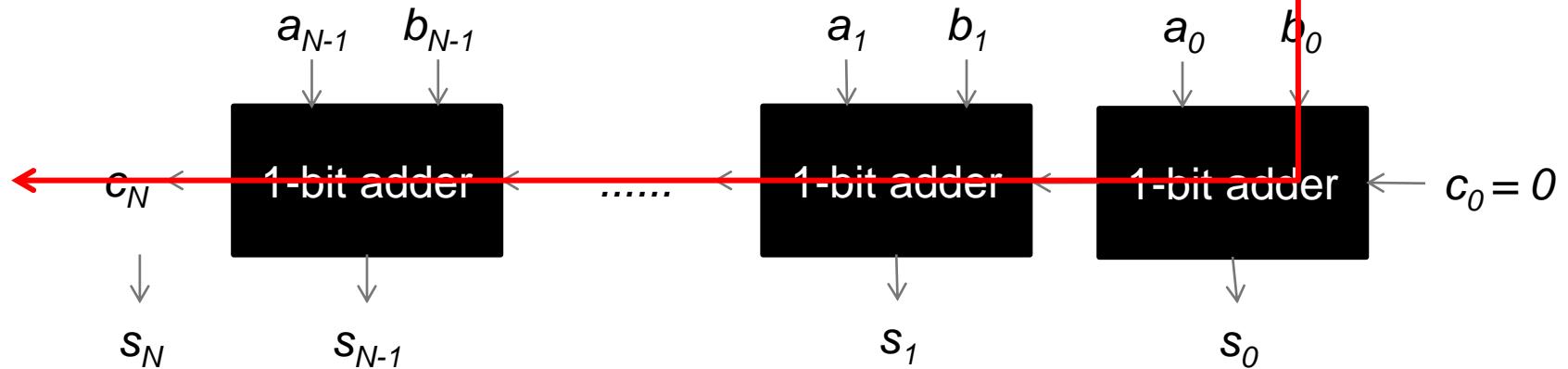


- care este problema fundamentală în circuitul de mai sus?
  - trebuie să așteptăm să putem calcula acei biți de carry



# CIRCUITE MAI COMPLEXE

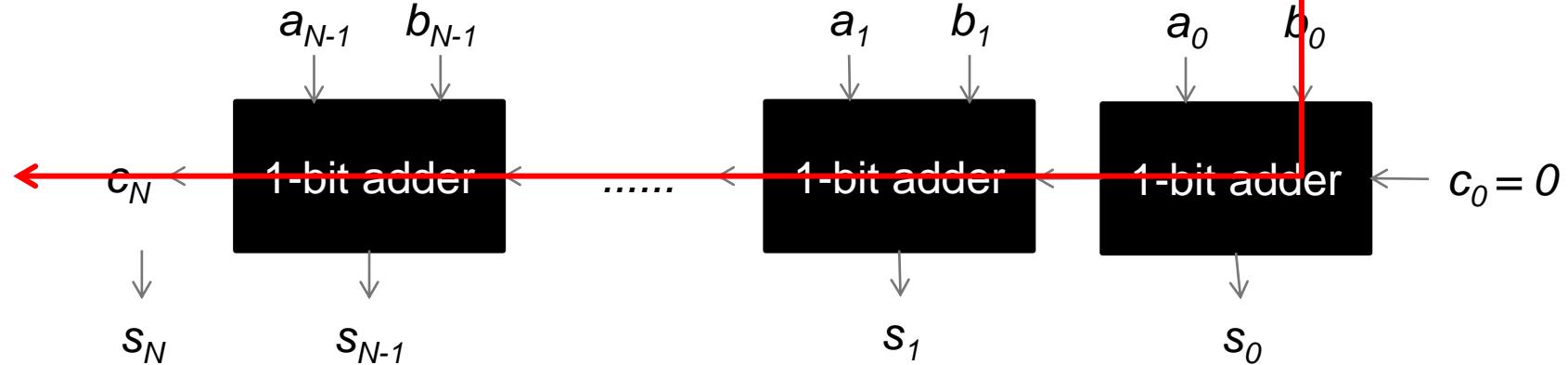
- circuit de adunare



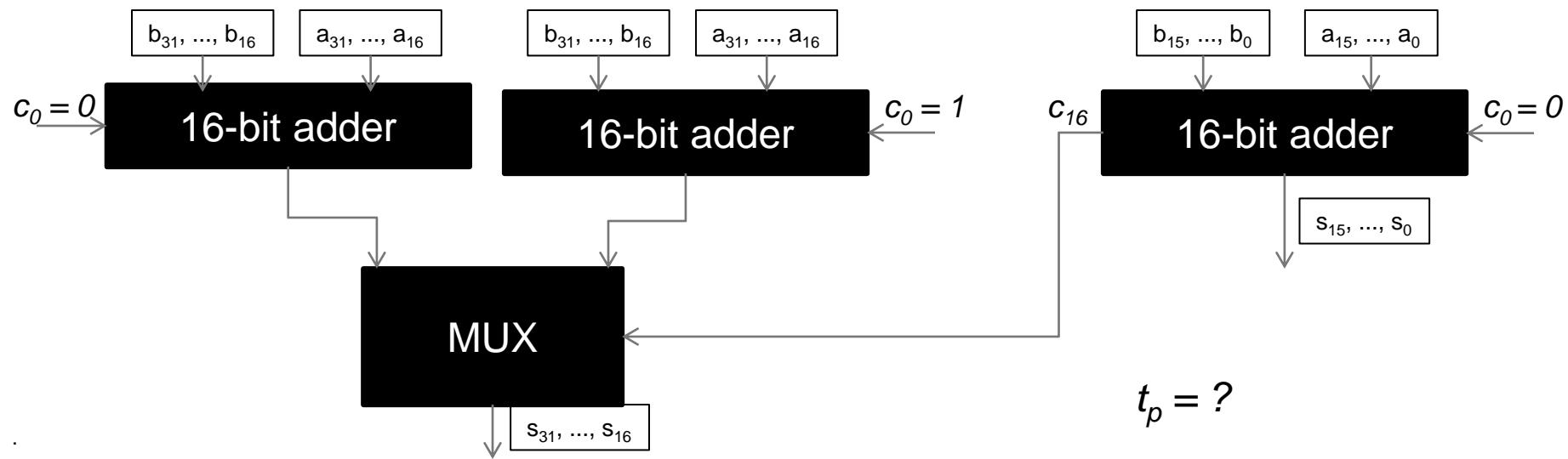
- cât este timpul de propagare în acest caz?
- $N t_P$
- ce putem face pentru a îmbunătății timpul de propagare?

# CIRCUITE MAI COMPLEXE

- circuit de adunare

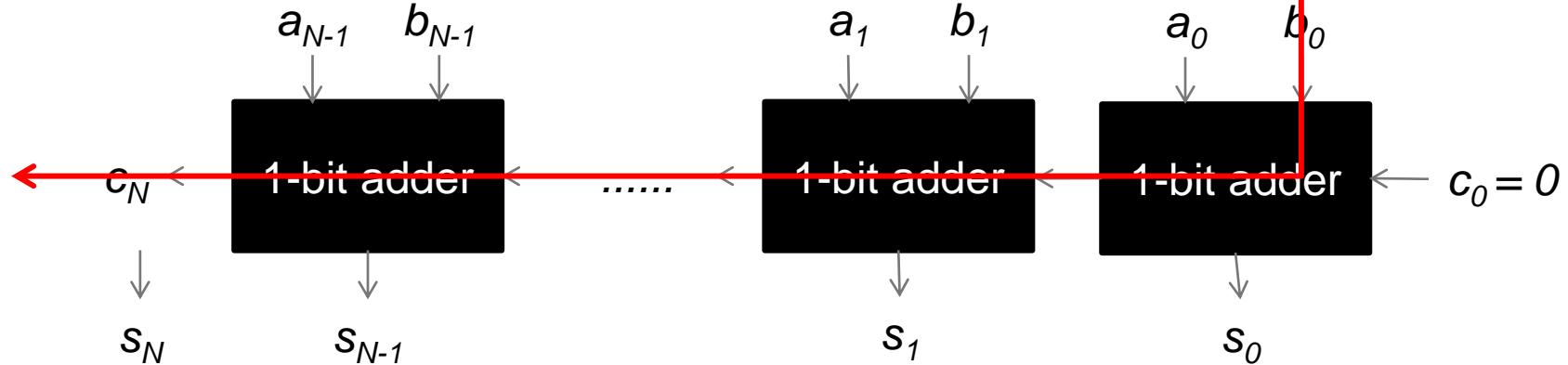


- cât este timpul de propagare în acest caz?
- $N t_P$

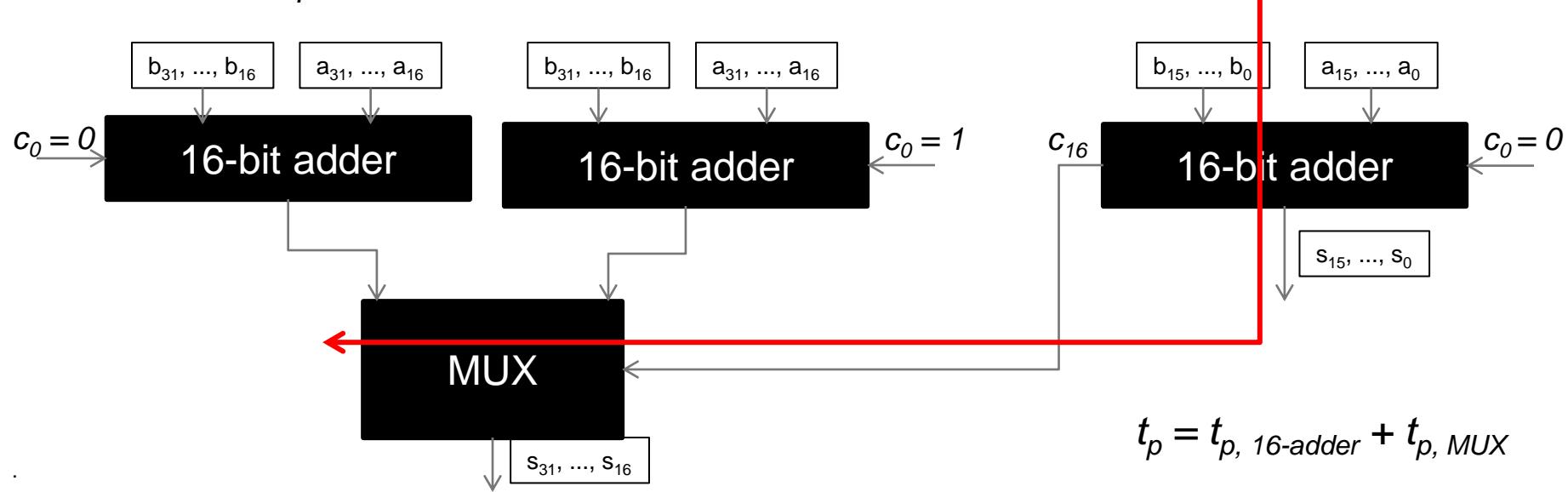


# CIRCUITE MAI COMPLEXE

- circuit de adunare

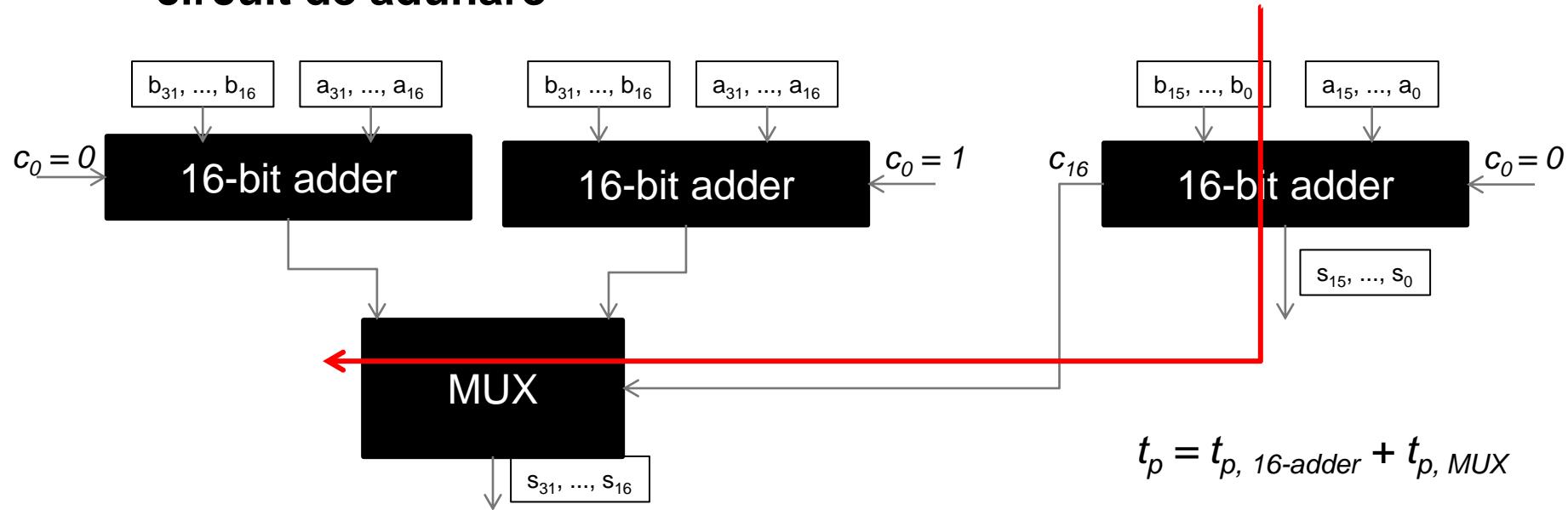


- cât este timpul de propagare în acest caz?
- $N t_P$



# CIRCUITE MAI COMPLEXE

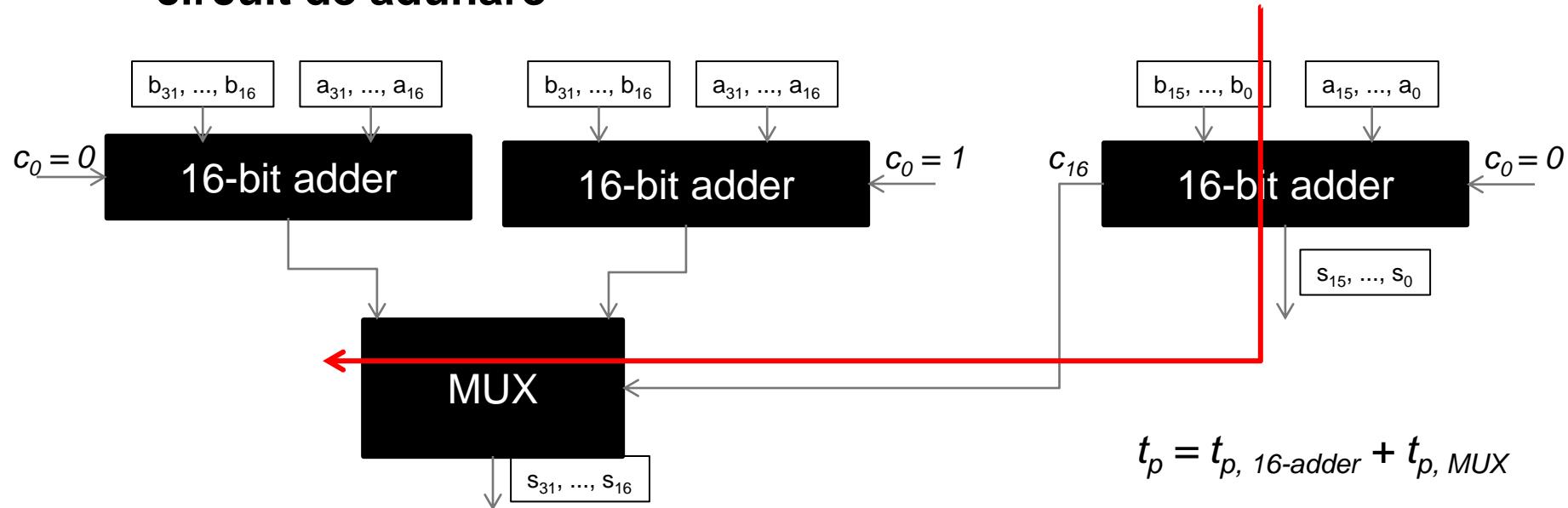
- circuit de adunare



- aici am considerat un exemplu de adunare pe 32 de biți
- putem aplica aceeași idee și pentru circuitele de 16 biți de sus
- $t_p = ?$

# CIRCUITE MAI COMPLEXE

- circuit de adunare



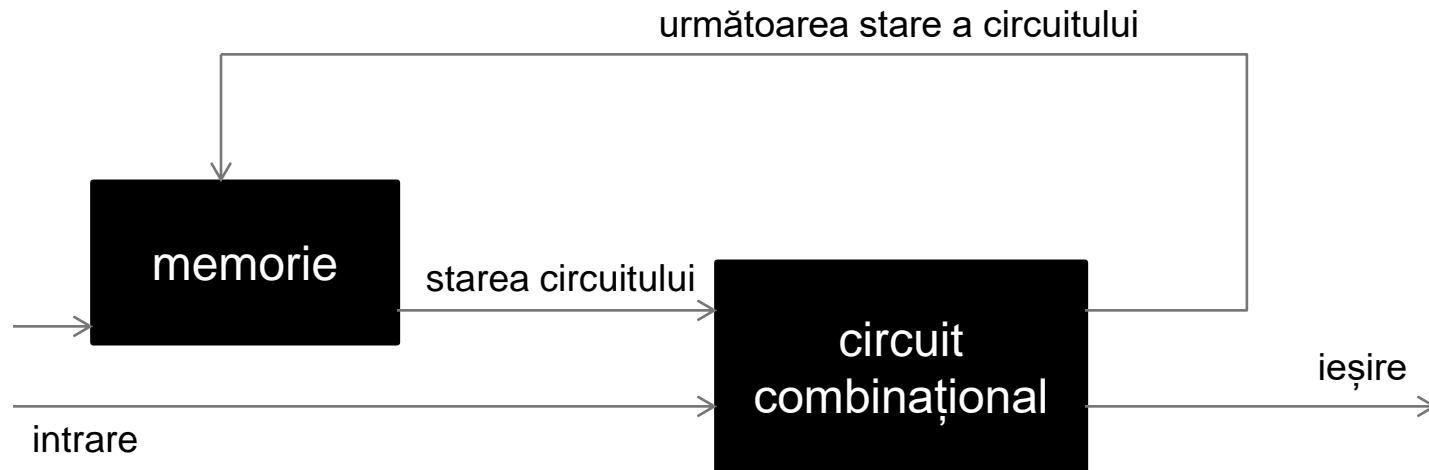
- aici am considerat un exemplu de adunare pe 32 de biți
- putem aplica aceeași idee și pentru circuitele de 16 biți de sus
- $t_p = O(\log_2 N)$

# CIRCUITE SECVENTIALE

- **circuitele combinaționale sunt eficiente (în general) dar au o problema majoră: sunt one-shot**
  - nu putem itera
  - nu permit niciun fel de “logică internă” sau “memorie internă”
  - nu există o stare internă a circuitului
  - sunt prea simple, asociază o funcție logică a intrărilor cu o ieșire (după un anumit timp)
  - unele lucruri nu pot fi implementate folosind doar logică combinațională
- **circuitele secvențiale adresează unele dintre limitările circuitelor combinaționale**

# CIRCUITE SECVENTIALE

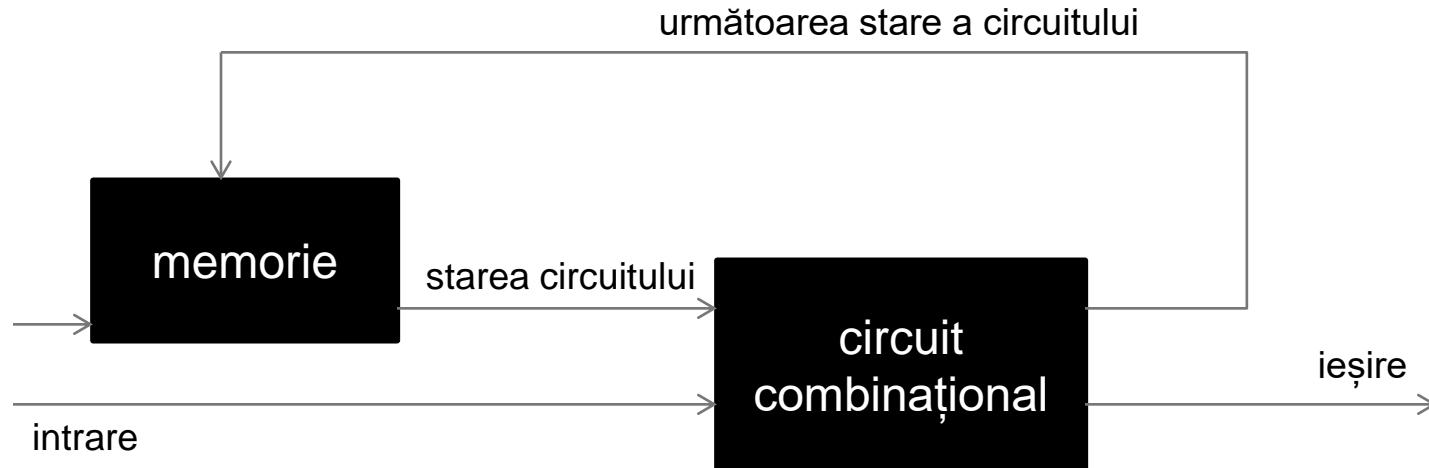
- avem nevoie să introducem elemente de tip “memorie”



- În felul acesta adăugăm o stare circuitului

# CIRCUITE SECVENTIALE

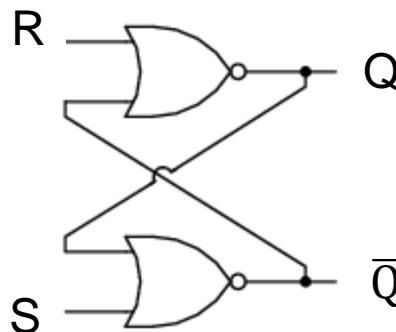
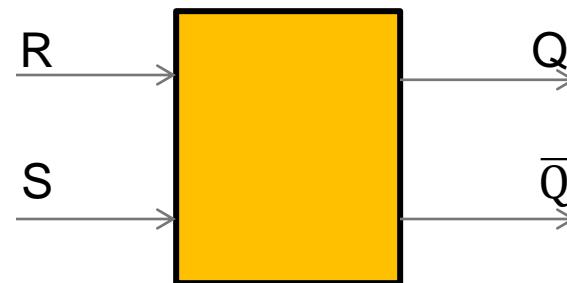
- avem nevoie să introducem elemente de tip “memorie”



- am vorbit până acum că biții pe care îi reprezentăm sunt voltaj
- dar energia electrică este dificil de stocat (în timp)
  - fenomen de scurgere (leakage)
- dacă vrem să memorăm ceva, trebuie să facem un refresh din când în când pentru a actualiza nivelul de energie electrică

# CIRCUITE SECVENȚIALE

- SR Latch (Set-Reset Latch)
  - memorează un bit de informație



S	R	Q	$\bar{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

nu se schimbă nimic

aici punem "0" în memorie

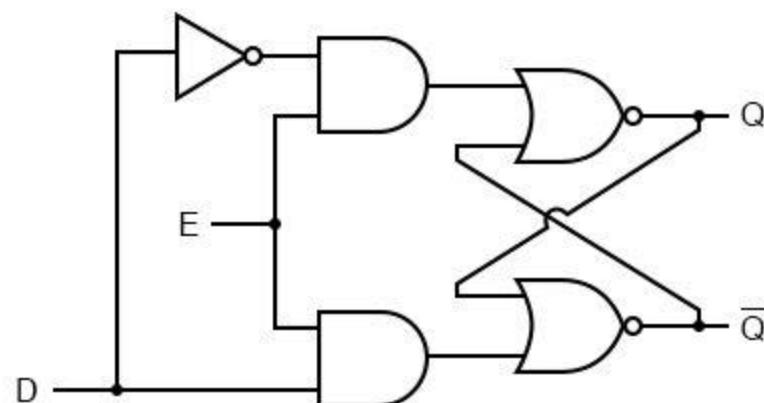
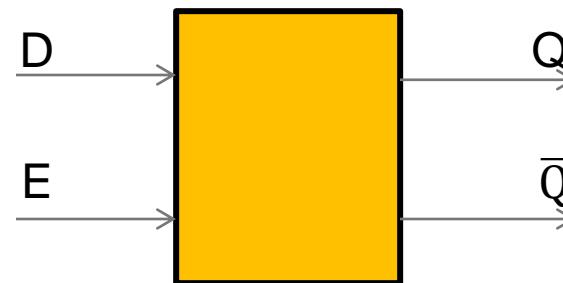
aici punem "1" în memorie

stare invalidă

# CIRCUITE SECVENȚIALE

- SR Latch (Set-Reset Latch)

- e bun dar are două intrări, putem face ceva cu o singură intrare?
- D Latch



E	D	Q	$\bar{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0

nu se schimbă nimic

nu se schimbă nimic

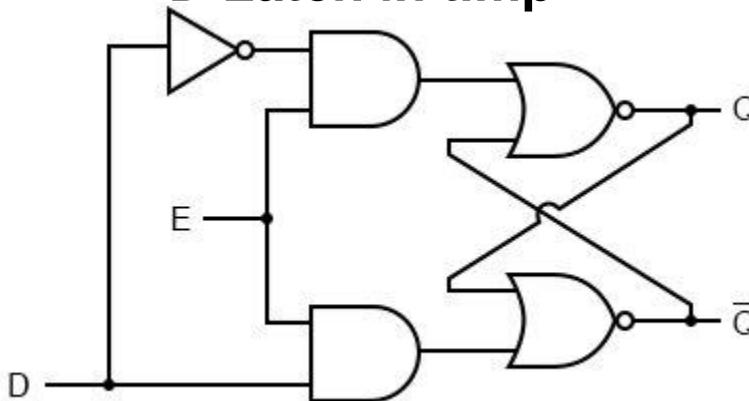
aici punem "0" în memorie

aici punem "1" în memorie

E de la Enable, adică activare  
dacă E = 0 nu se întâmplă nimic

# CIRCUITE SECVENȚIALE

- D Latch în timp



E	D	Q	$\bar{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0

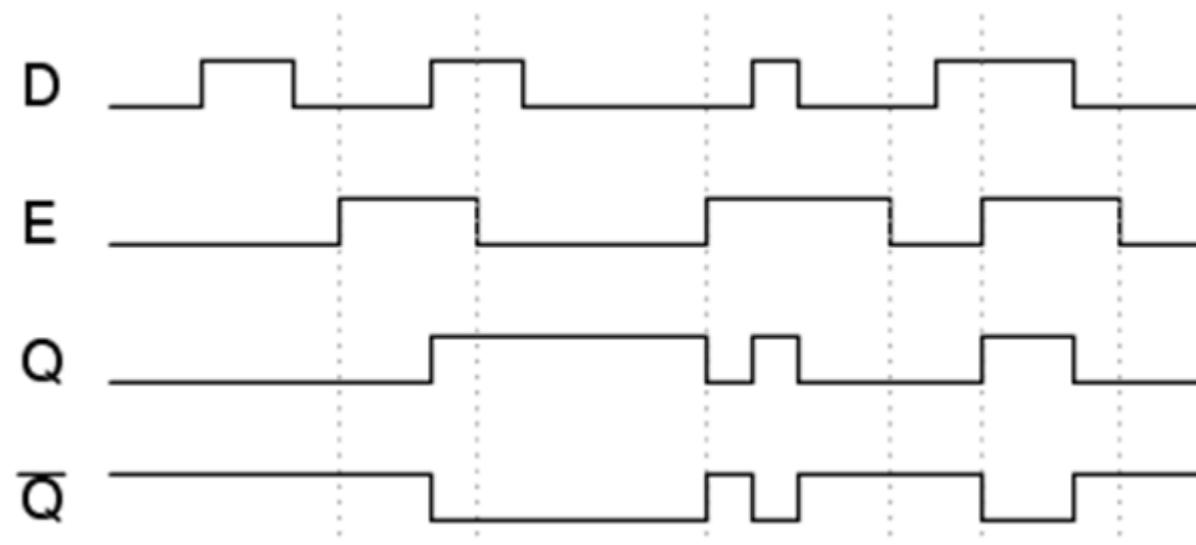
nu se schimbă nimic

nu se schimbă nimic

aici punem "0" în memorie

aici punem "1" în memorie

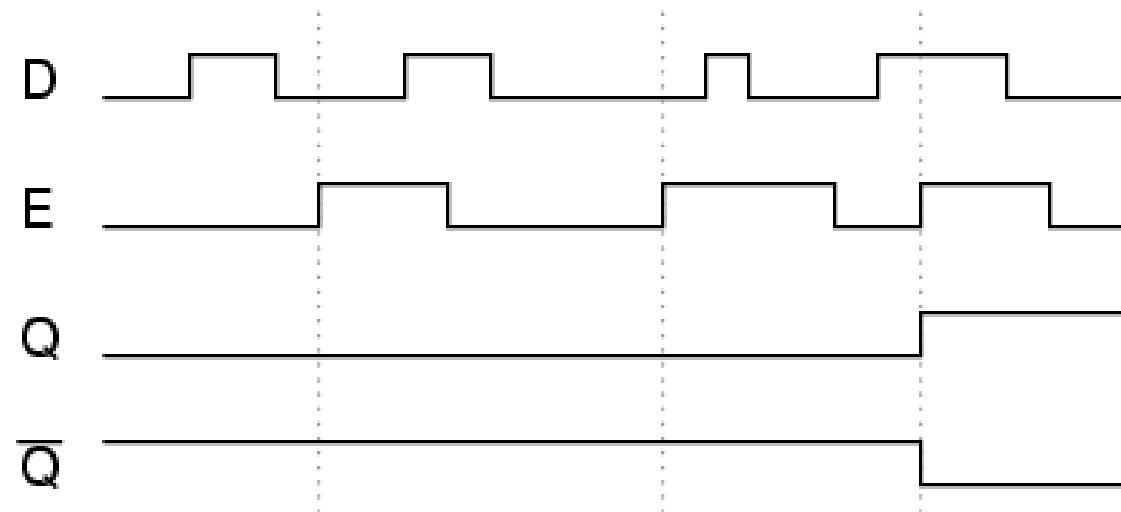
E de la Enable, adică activare  
dacă  $E = 0$  nu se întâmplă nimic



# CIRCUITE SECVENȚIALE

- D Flip-Flop

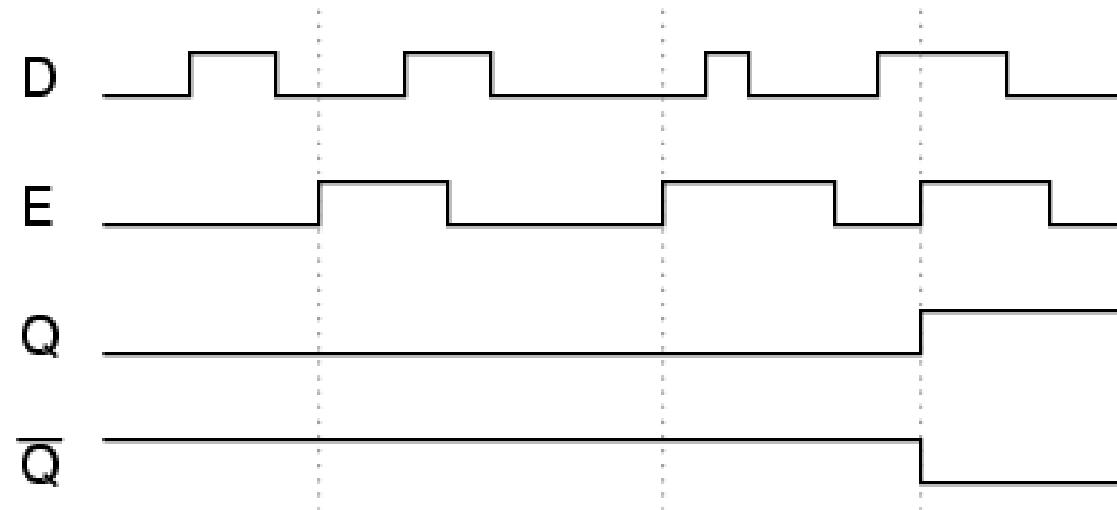
- D Latch e bun, dar vrem să sincronizăm mai multe dispozitive
- vrem ca activarea să se facă cand E crește, nu când E e activ



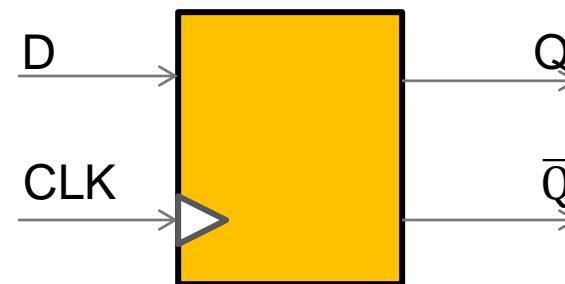
- E devine clock – adică ceasul sistemului
  - la un interval fix de timp, sistemul face ceva

# CIRCUITE SECVENȚIALE

- D Flip-Flop



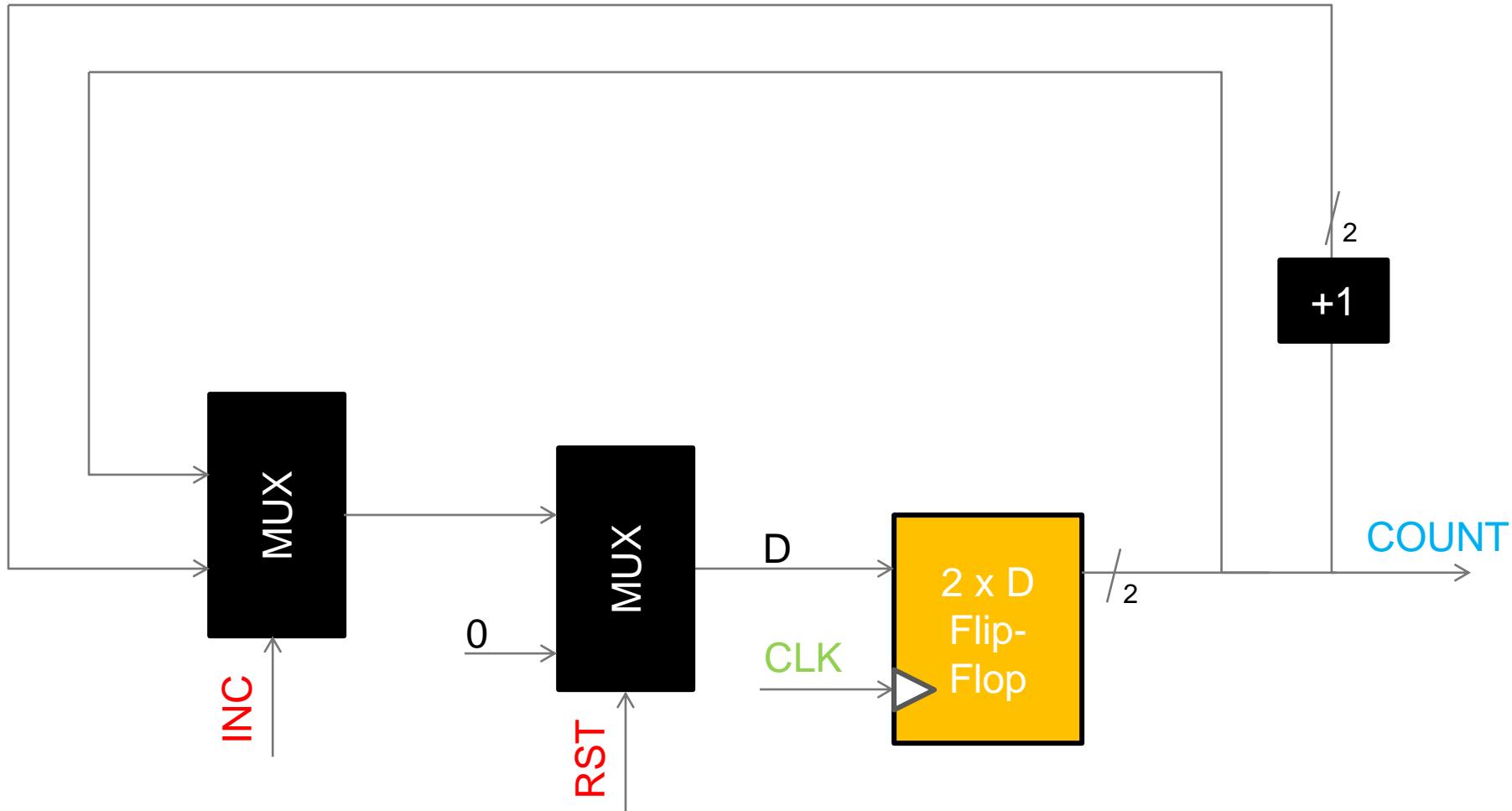
- E devine clock – adică ceasul sistemului
  - la un interval fix de timp (un ciclu), sistemul face ceva



un set de câteva D Flip Flops care au același CLK = un registru

# CIRCUITE SECVENTIALE

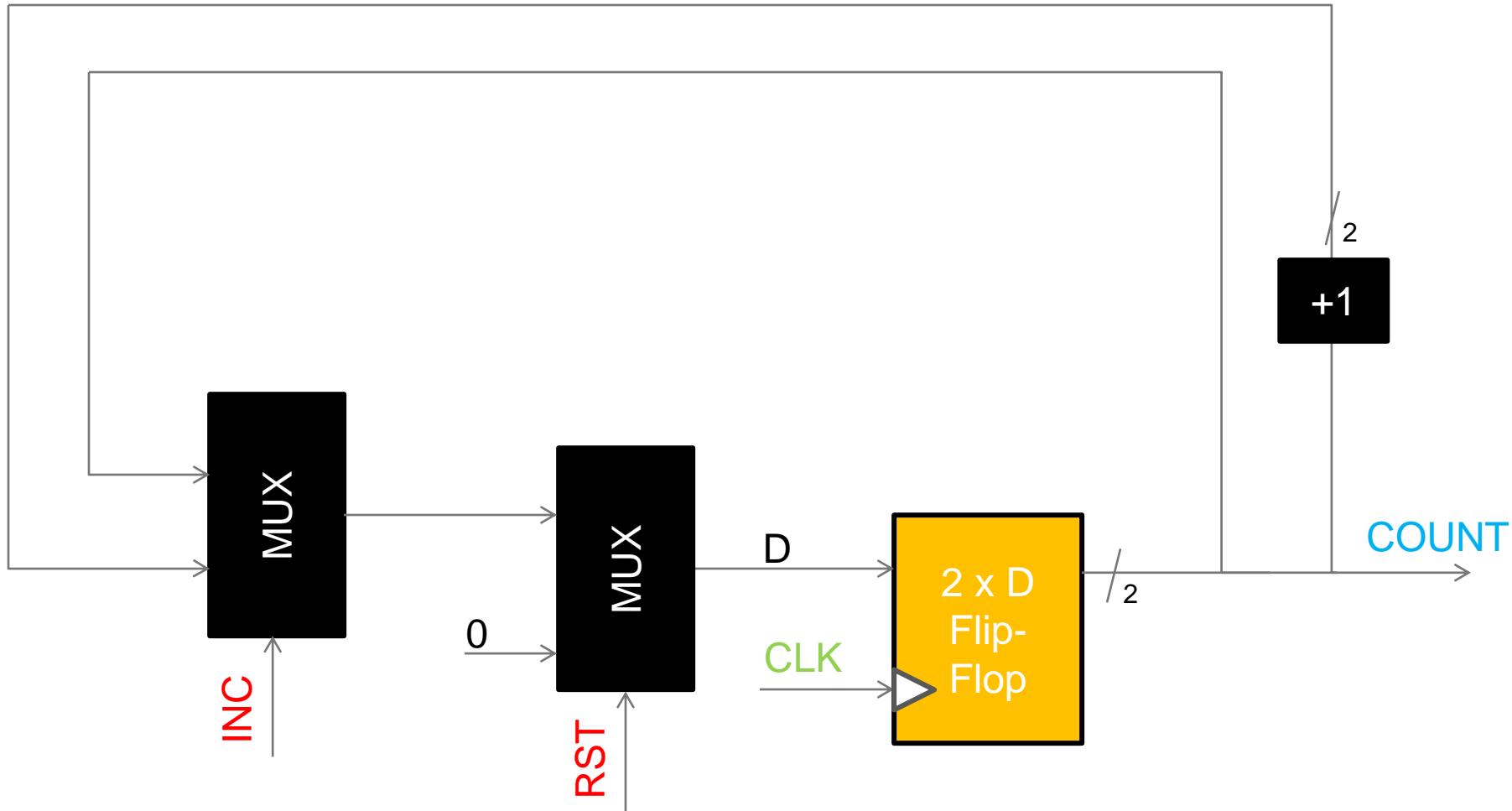
- un exemplu



ce face acest circuit?

# CIRCUITE SECVENTIALE

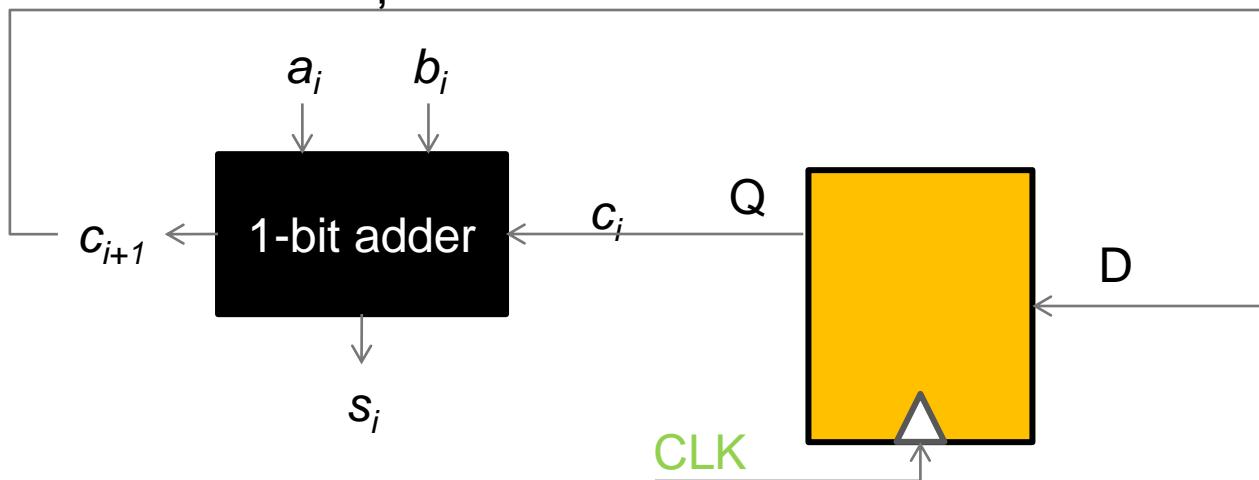
- un exemplu



este un counter pe 2 biți

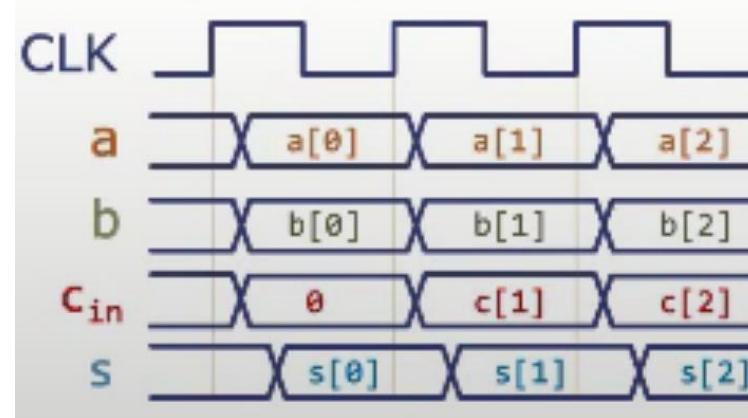
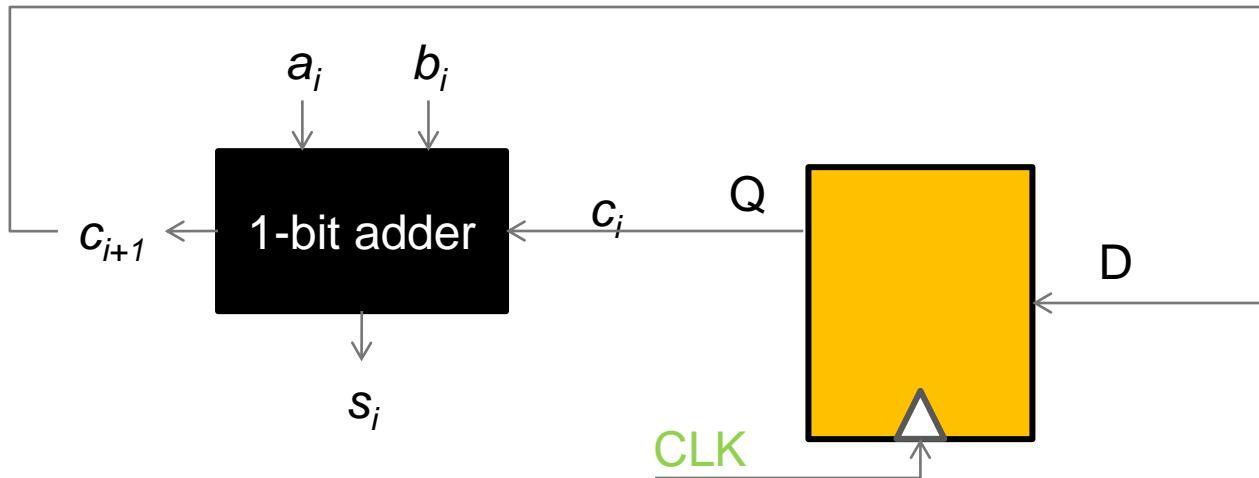
# CIRCUITE SECVENTIALE

- avantajele circuitelor secvențiale
  - avem stare internă
    - exemplu: avem un registru în care memorăm count-ul curent
  - avem variabila de timp
    - intrările/ieșirile nu sunt fixe
    - număr variabil de pași în rezolvare
    - exemplu: construiți un circuit care poate să adune două numere de dimensiune (număr de biți) variabil – adică nu prestatibilim pe cât biți e fiecare număr
      - noi aşa facem adunarea binară. cum arată circuitul?



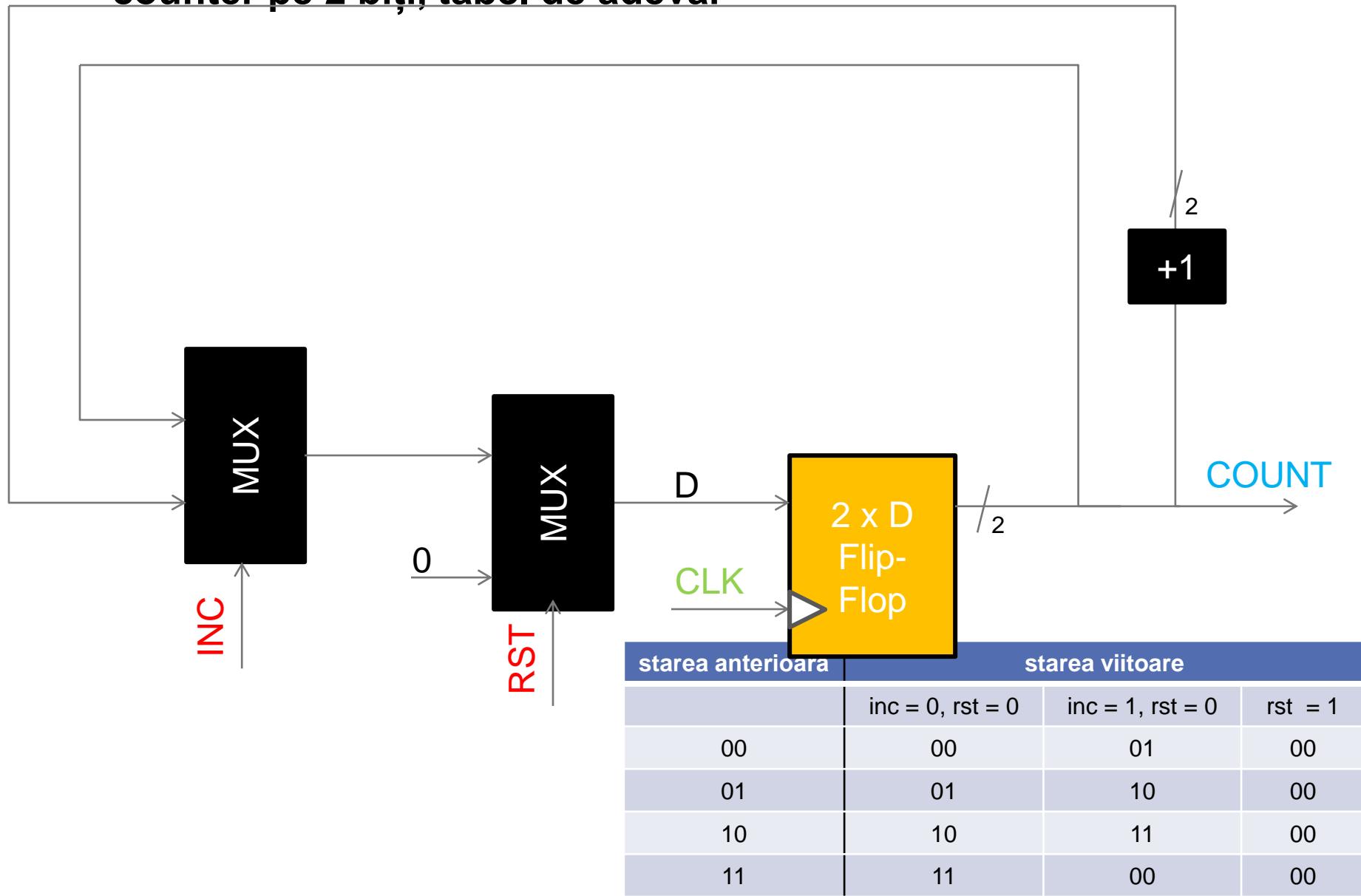
# CIRCUITE SECVENTIALE

- circuit de adunare, în timp



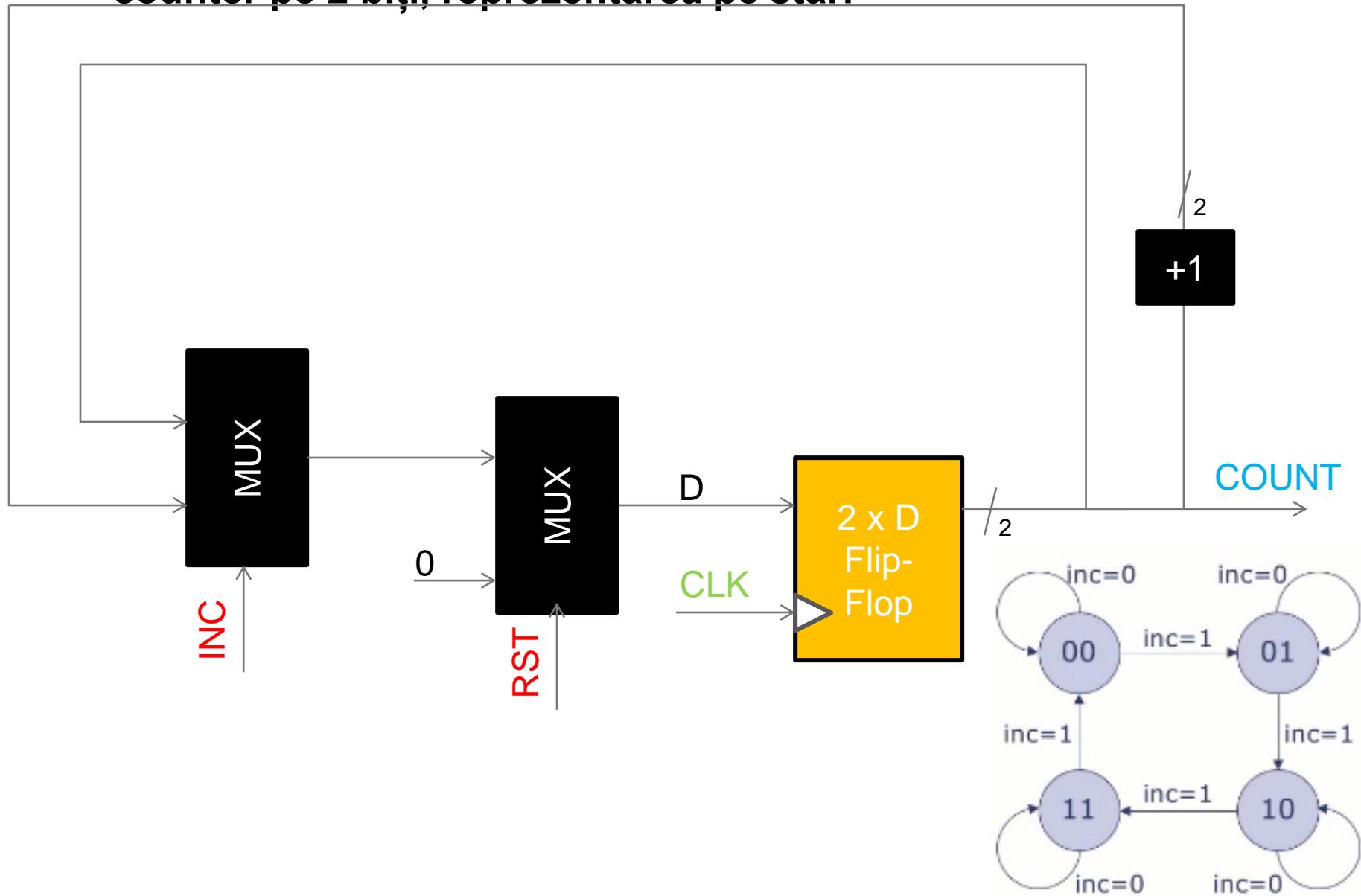
# CIRCUITE SECVENTIALE

- counter pe 2 biți, tabel de adevăr



# CIRCUITE SECVENTIALE

- counter pe 2 biți, reprezentarea pe stări



# **CE AM FĂCUT ASTĂZI**

- **tabele de adevăr și expresii booleene**
- **caracteristici ale porților logice**
- **circuitul de adunare combinațional**
- **circuite secvențiale**
  - SR Latch
  - D Latch
  - D Flip Flop
  - circuit de adunare secvențial

# **DATA VIITOARE ...**

- seminarul următor are multe exerciții cu circuite combinaționale și secvențiale
- mai multe circuite secvențiale mai complexe
- mai multe despre reprezentarea pe stare
- înmulțirea binară

# LECTURĂ SUPLIMENTARĂ

- PH book
  - Appendix B
- Computerphile, Logic gates playlist,  
<https://www.youtube.com/playlist?list=PLzH6n4zXucko1YVRjhnquPaNOfZrymVQH>
- Computerphile, Binary Addition & Overflow,  
<https://www.youtube.com/watch?v=WN8i5cwjkSE>
- Ben Eater, SR latch, <https://www.youtube.com/watch?v=KM0DdEaY5sY>
- Ben Eater, D latch, [https://www.youtube.com/watch?v=peCh\\_859q7Q](https://www.youtube.com/watch?v=peCh_859q7Q)
- Ben Eater, D flip-flop, <https://www.youtube.com/watch?v=YW-GkUguMM>



# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x05**

**ÎNMULȚIREA/ÎMPĂRTIREA NUMERELOR ÎNTREGI,  
REPREZENTAREA ÎN VIRGULĂ MOBILĂ**

Cristian Rusu

# **DATA TRECUTĂ**

- **detaliile unui circuit de adunare binară**
- **multiplexare**
- **circuite secvențiale**
  - SR Latch
  - D Latch
  - D Flip Flop
  - circuit adunare secvențial
- **reprezentarea pe stări a unui circuit**

# CUPRINS

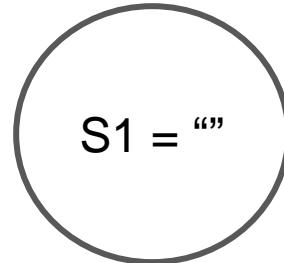
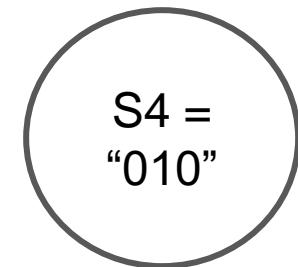
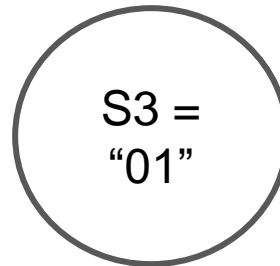
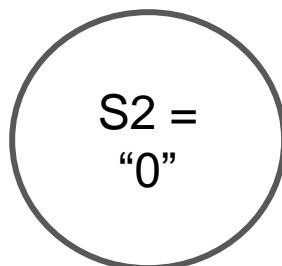
- logică secvențială + combinatorială, un exemplu
- înmulțirea numerelor întregi binare
- împărțirea numerelor întregi binare
- reprezentarea numerelor în virgulă mobilă
- lucrul cu numerele în virgulă mobilă

# **SECVENTIAL, SEMINAR 0x02, EX 10**

- un semnal digital A poate lua valori {0, 1} în timp iar noi vrem să detectăm dacă semnalul are valoarea 010 la un moment dat. Dacă această secvență de biți este detectată în A atunci o variabilă Y este setată la 1, altfel această variabilă este 0.
- definim 4 stări

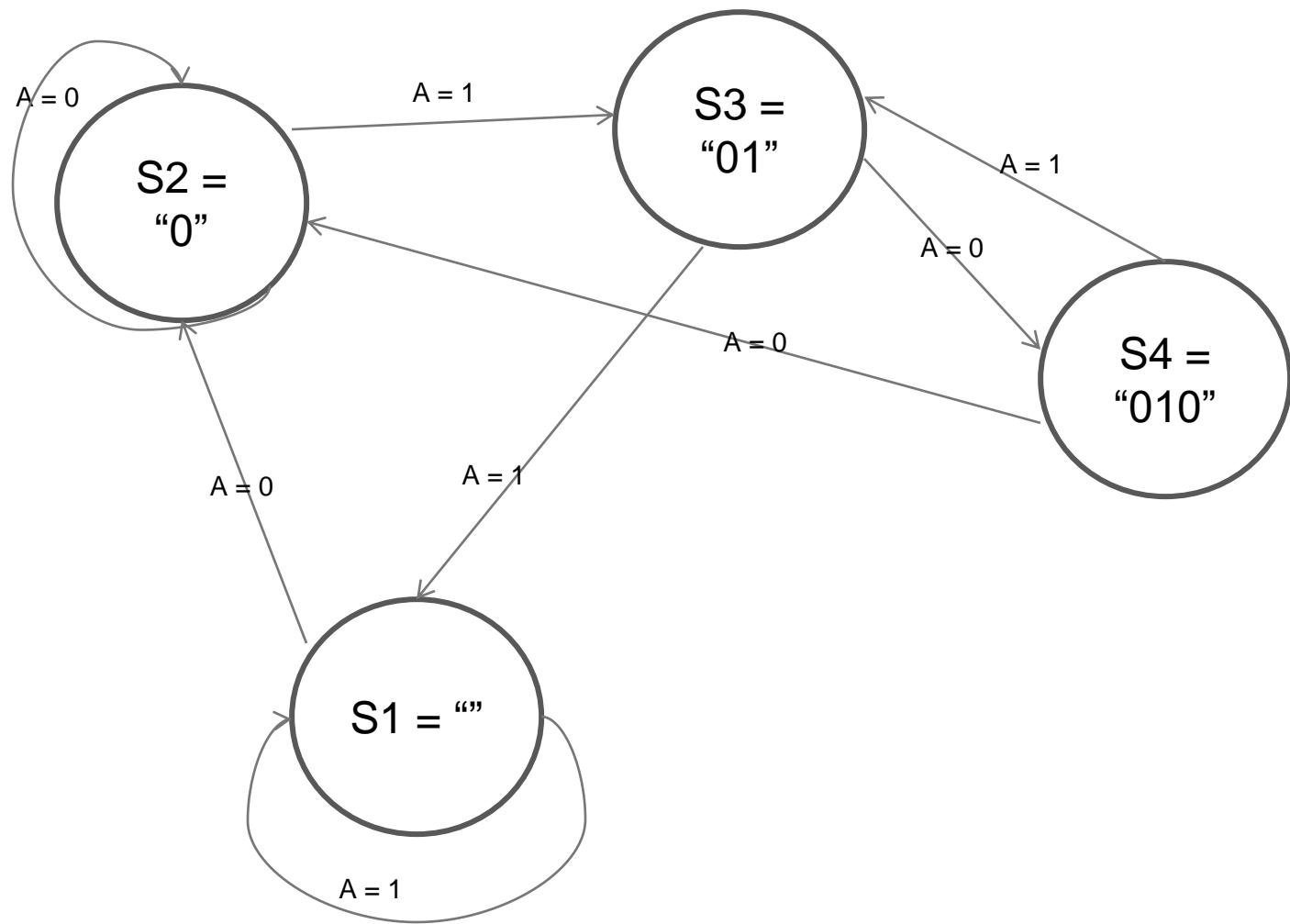
# SECVENTIAL, SEMINAR 0x02, EX 10

- un semnal digital A poate lua valori {0, 1} în timp iar noi vrem să detectăm dacă semnalul are valoarea 010 la un moment dat. Dacă această secvență de biți este detectată în A atunci o variabilă Y este setată la 1, altfel această variabilă este 0
- definim 4 stări

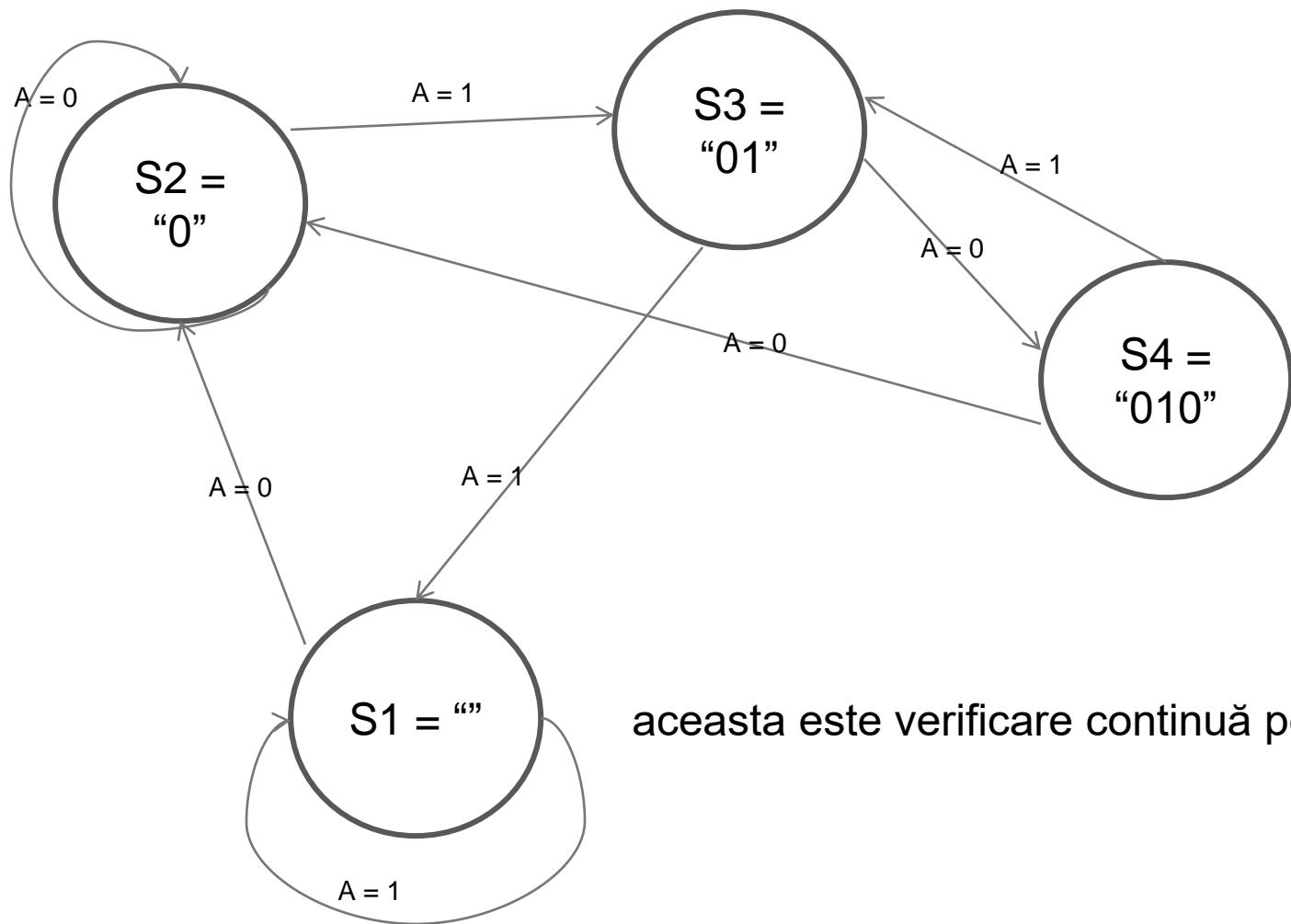


care sunt tranzitiiile între aceste stări?

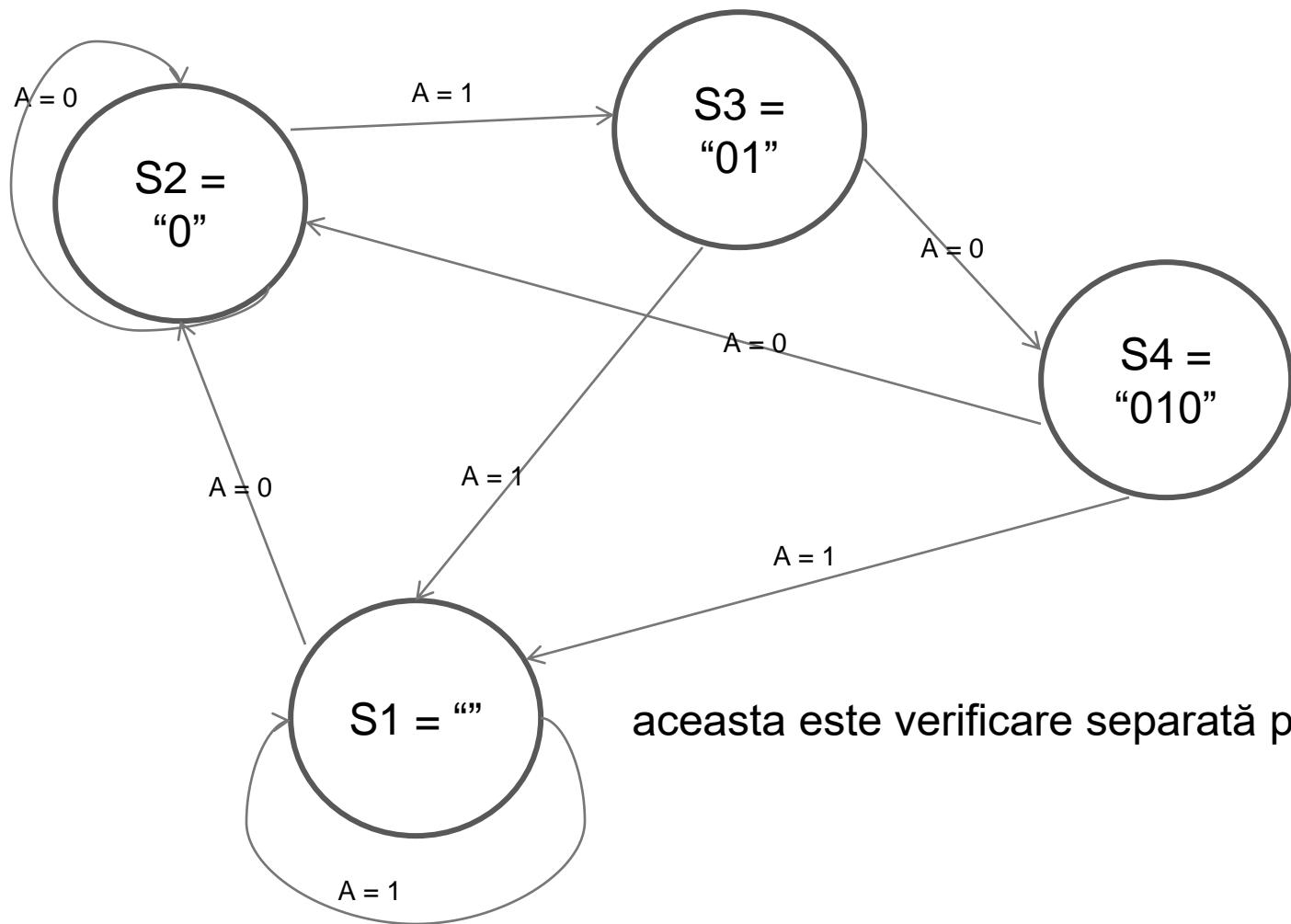
# SECVENTIAL, SEMINAR 0x02, EX 10



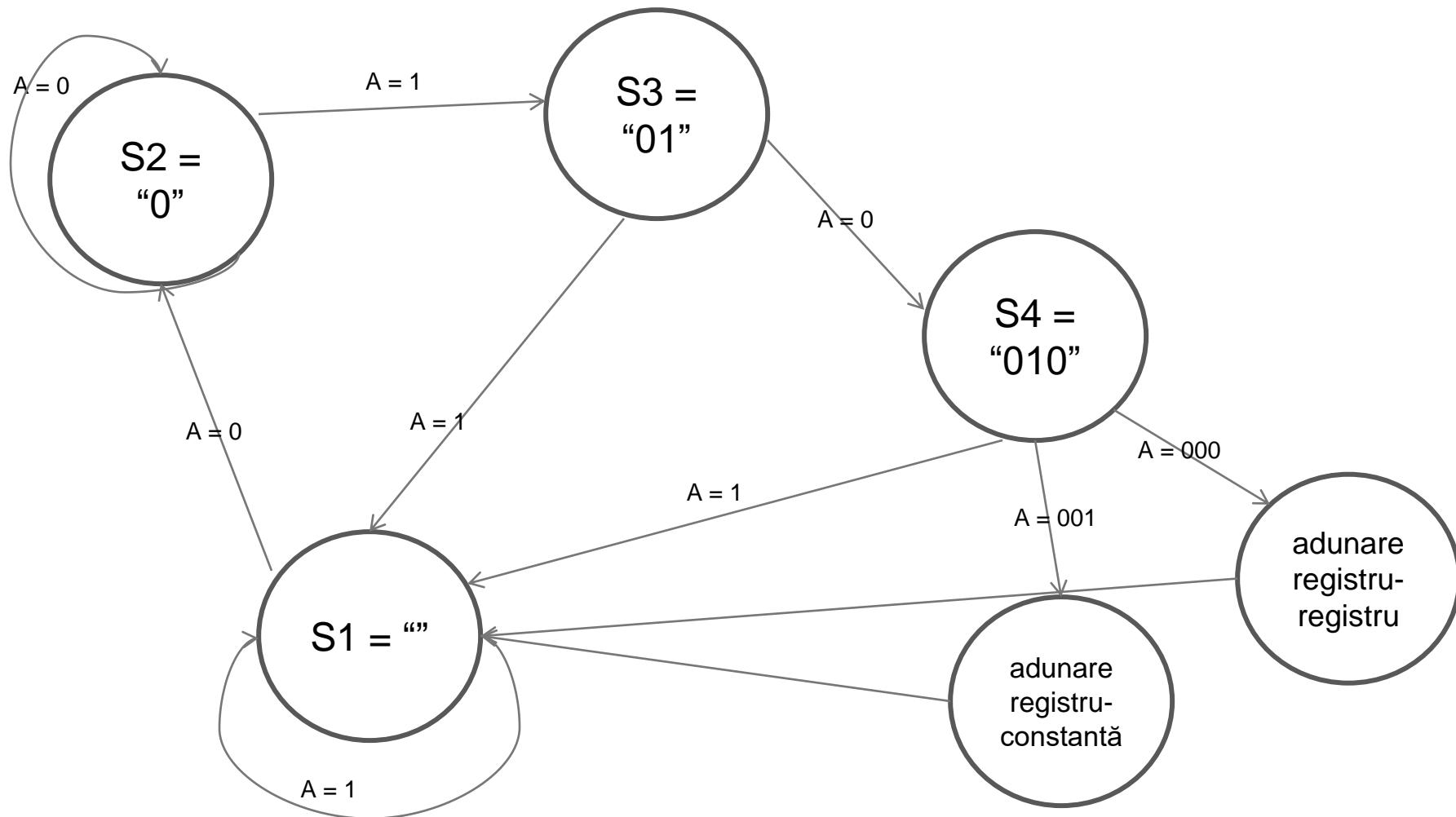
# SECVENTIAL, SEMINAR 0x02, EX 10



# SECVENTIAL, SEMINAR 0x02, EX 10

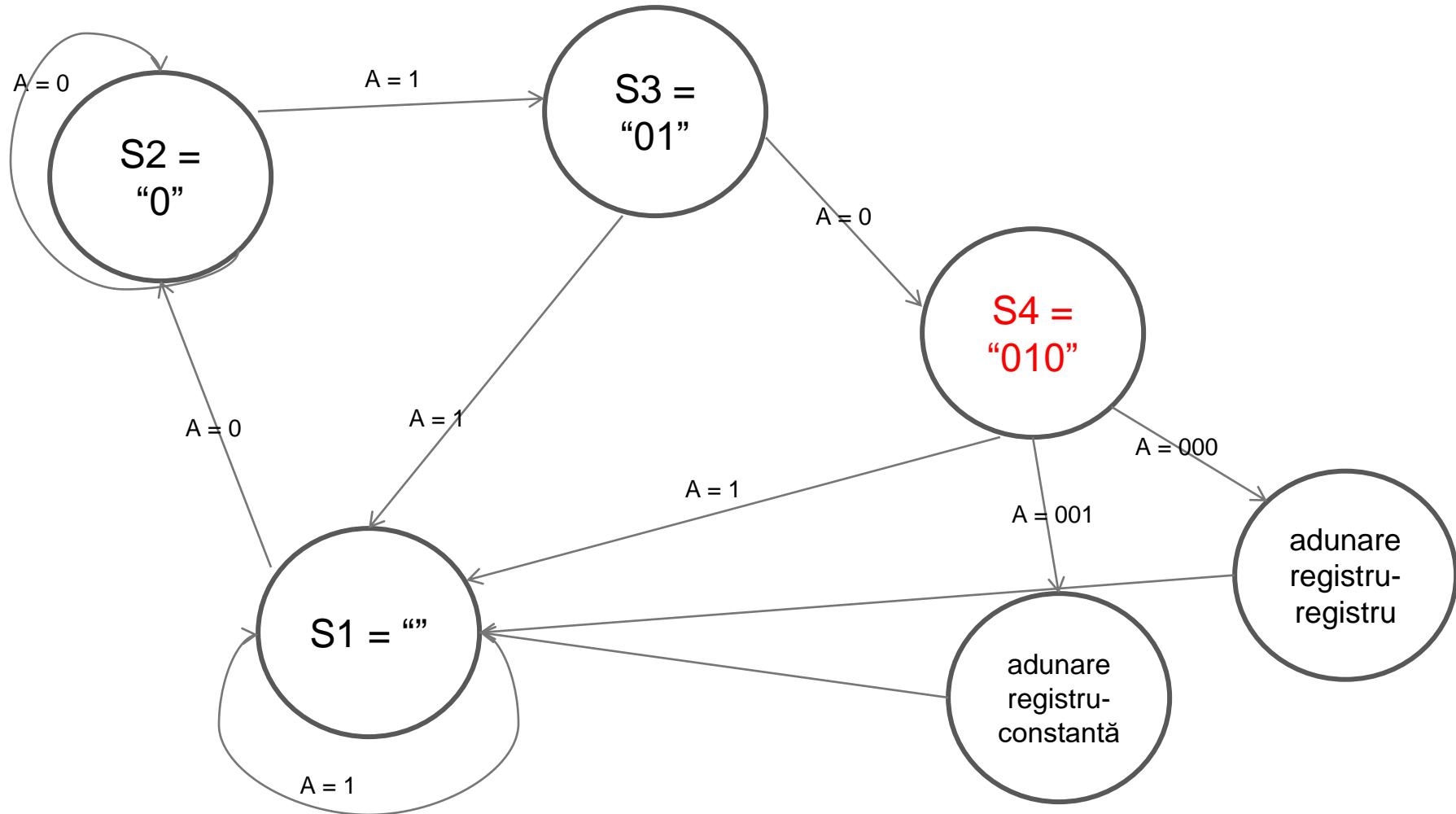


# CE FACE UN PROCESOR



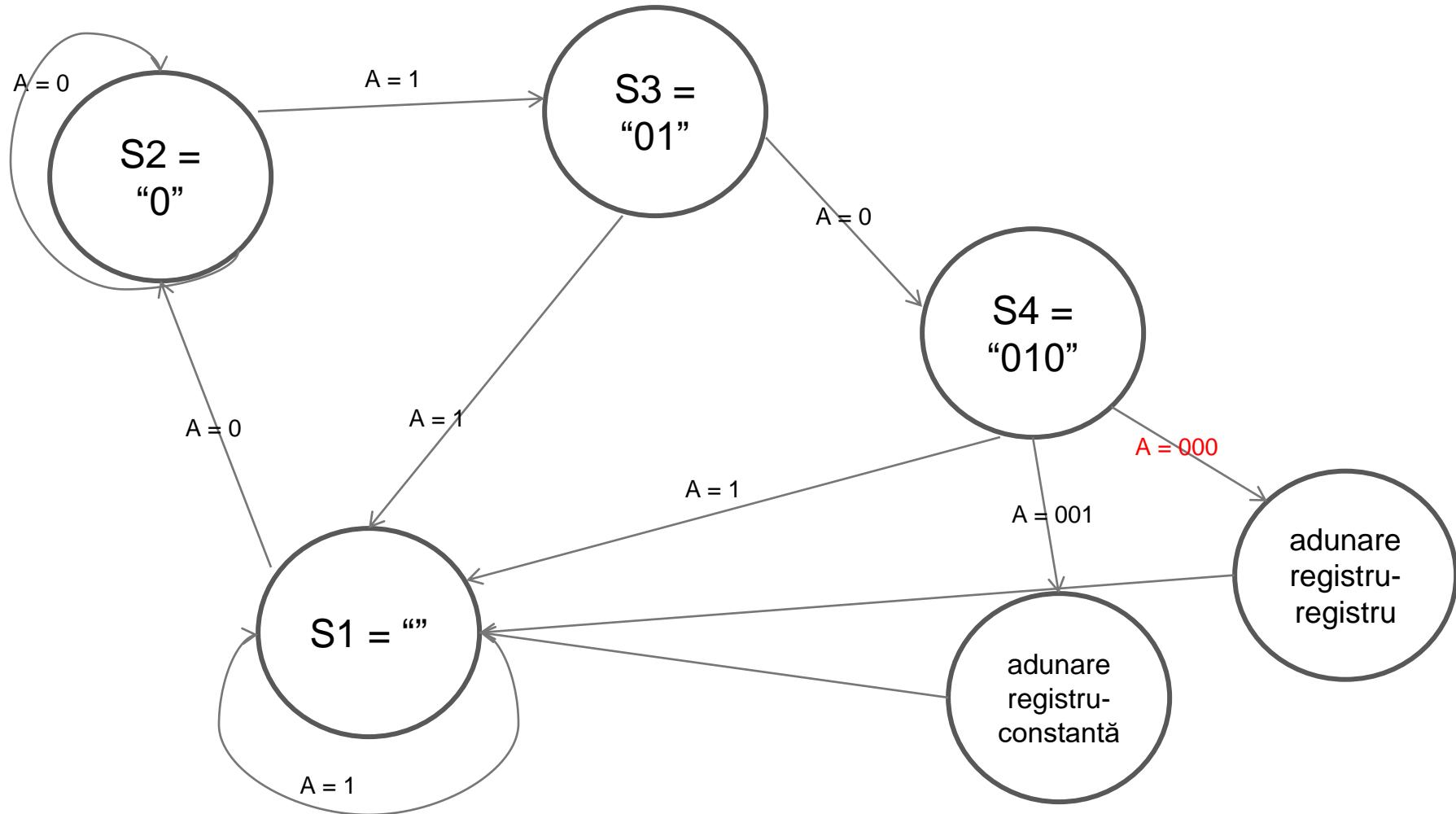
...0100001011010001010000100011001011 ...

# CE FACE UN PROCESOR



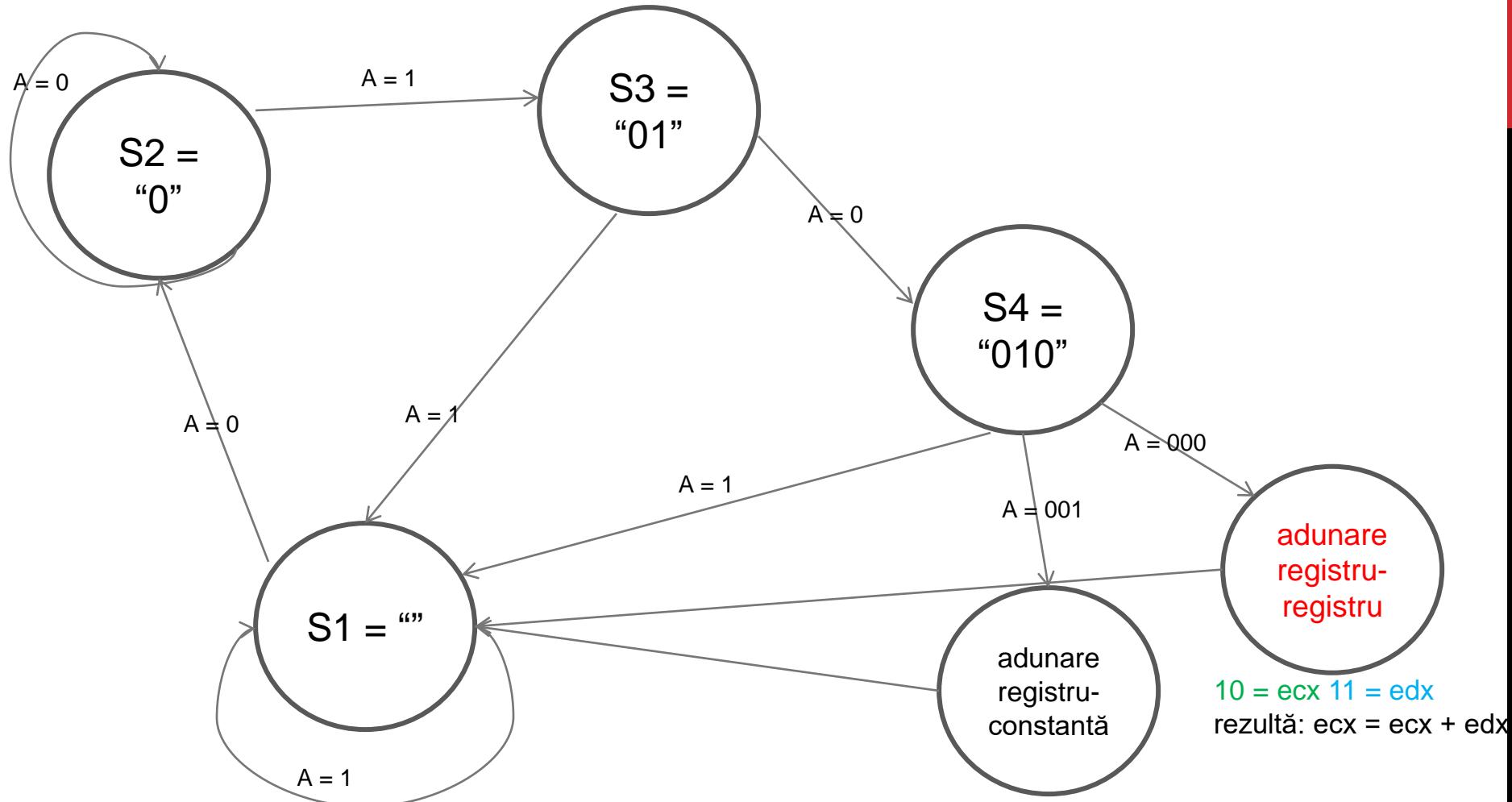
...0100001011010001010000100011001011 ...

# CE FACE UN PROCESOR



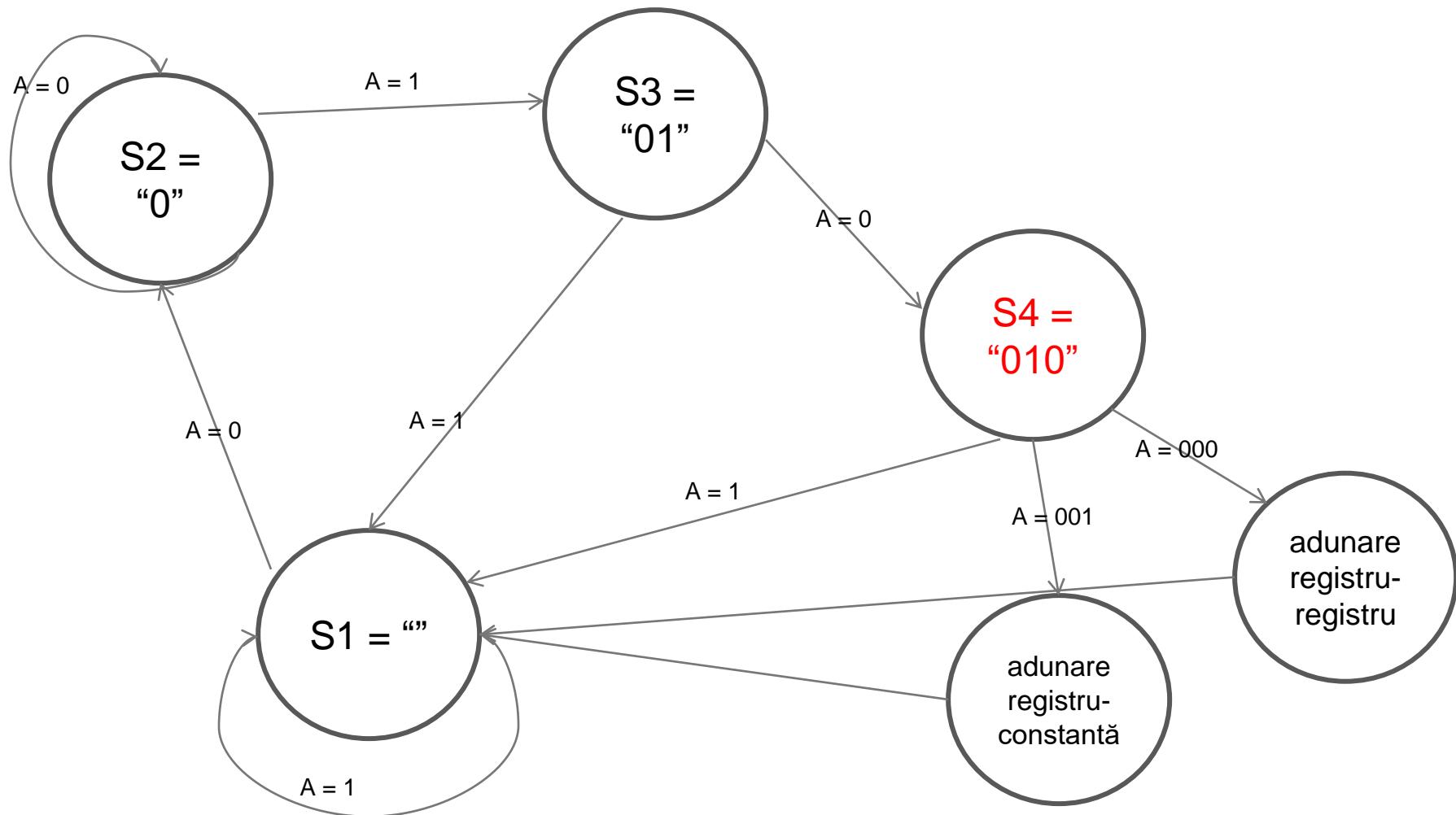
...0100001011010001010000100011001011 ...

# CE FACE UN PROCESOR



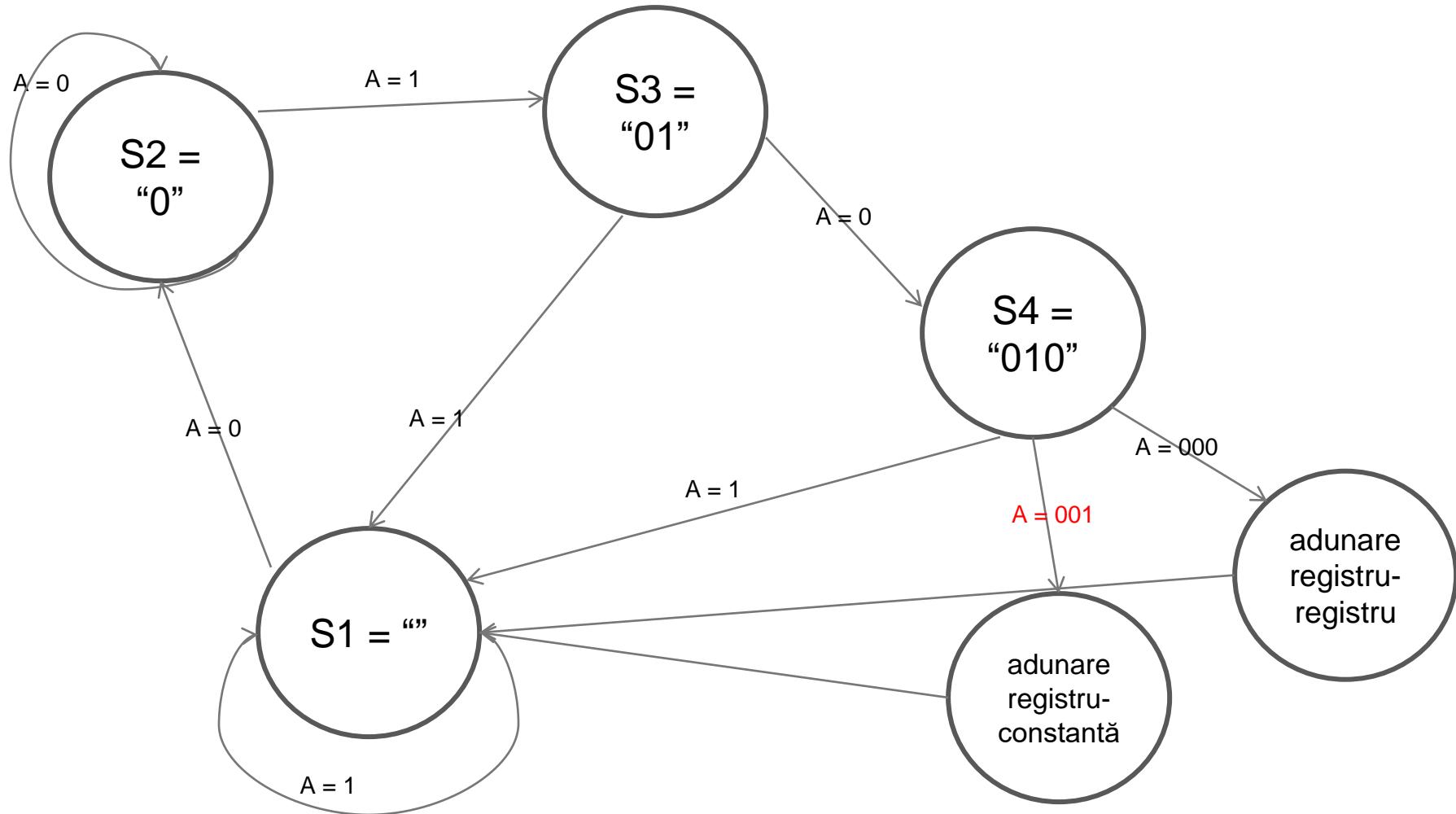
...0100001011010001010000100011001011 ...

# CE FACE UN PROCESOR



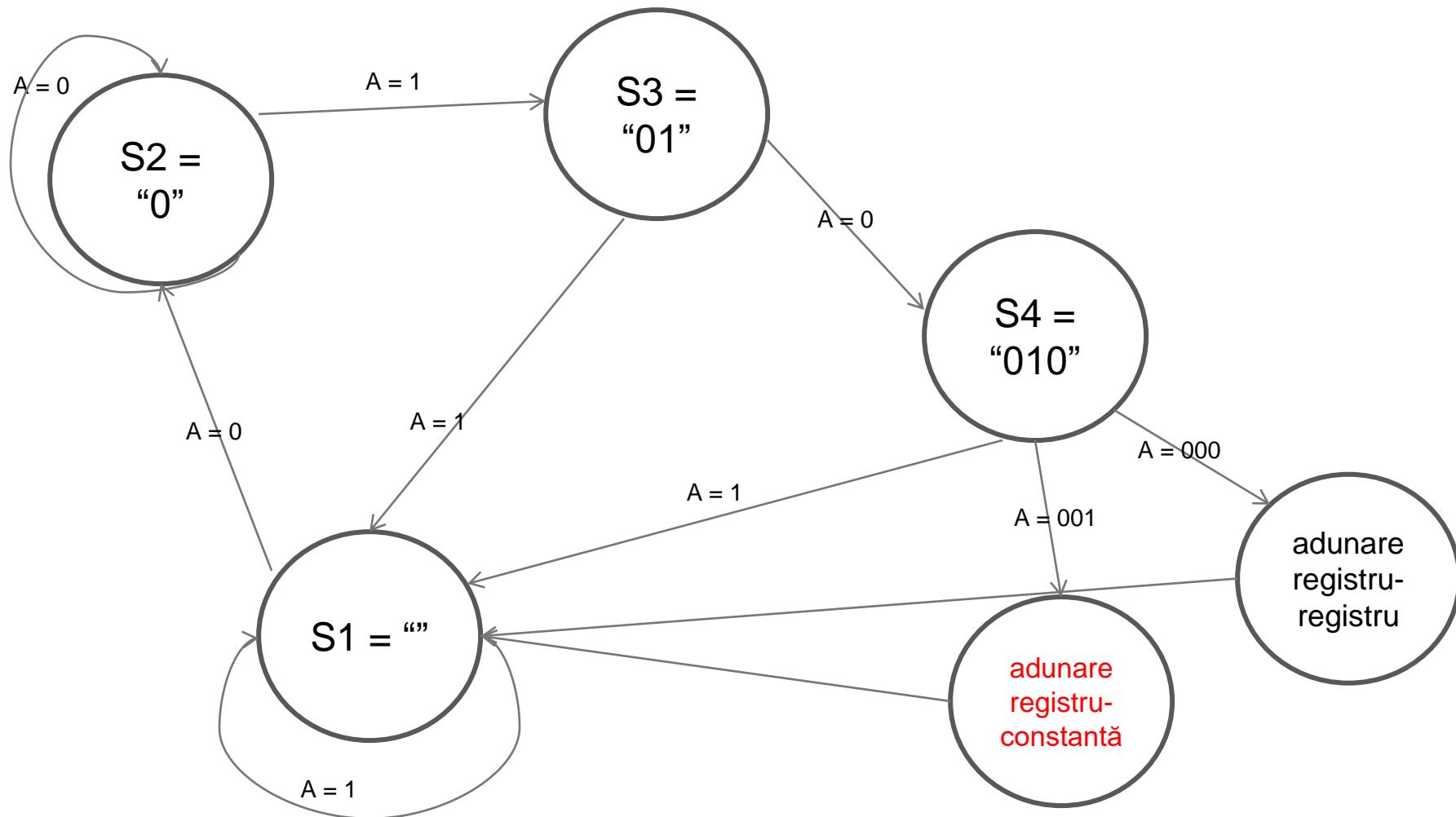
...0100001011**010**001010000100011001011 ...

# CE FACE UN PROCESOR



...010000101101001010000100011001011 ...

# CE FACE UN PROCESOR

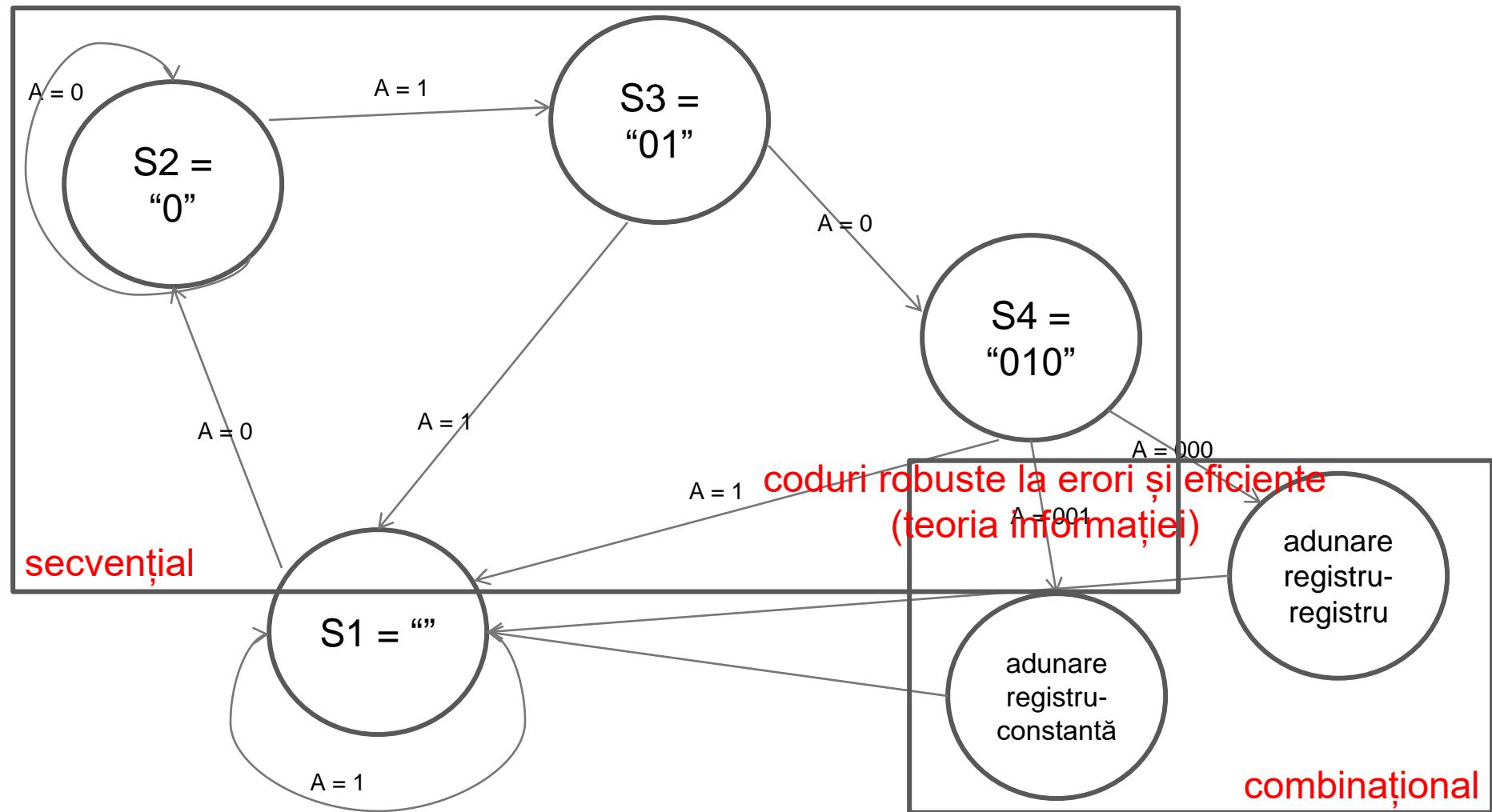


01 = ebx 0000100011001011 = 381

rezultă: ebx = ebx + 381

...0100001011010001010000100011001011 ...

# CE FACE UN PROCESOR



cod mașină ...0100001011010001010000100011001011 ...

# **CONTINUT NOU PENTRU CURS**

- **înmulțirea numerelor întregi binare**
- **împărțirea numerelor întregi binare**
- **reprezentarea numerelor în virgulă mobilă**
- **lucrul cu numerele în virgulă mobilă**

# ÎNMULTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \times b$

- $a$  și  $b$  pe N biți
- $s$  pe ?? biți

1	1	1	0
---	---	---	---

*a*

0	1	0	1
---	---	---	---

*b*

---

*s*

# ÎNMULTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \times b$

- $a$  și  $b$  pe  $N$  biți
- $s$  pe  $2N$  biți

1	1	1	0
---	---	---	---

$a$

0	1	0	1
---	---	---	---

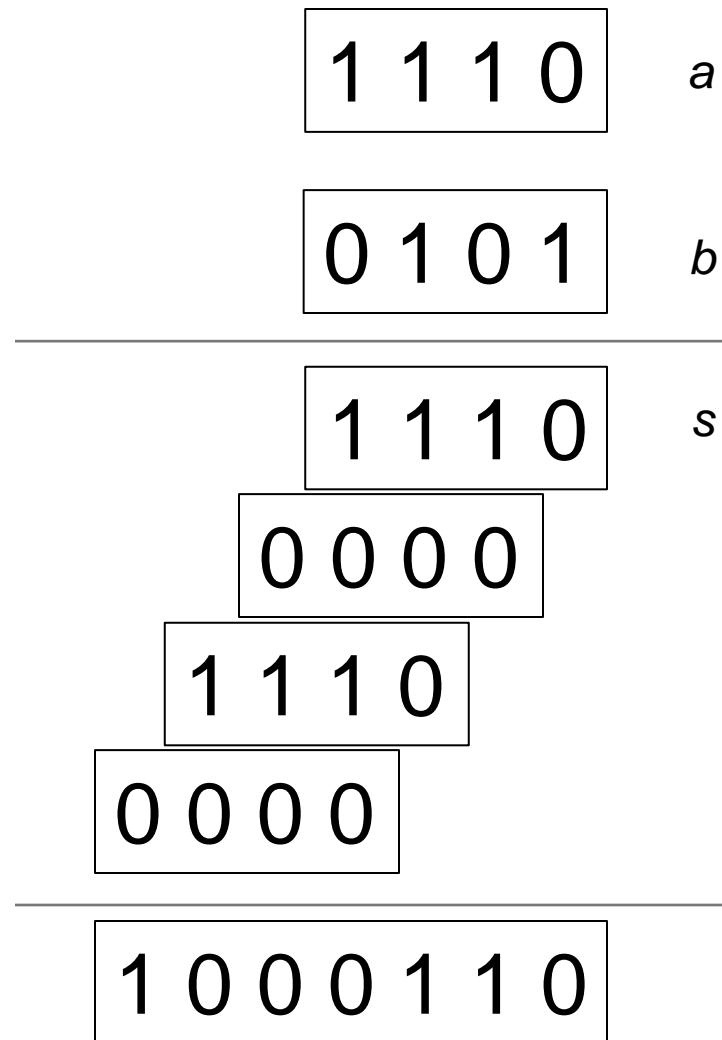
$b$

---

$s$

# ÎNMULȚIREA NUMERELEOR ÎNTREGI

- exemplu,  $s = a \times b$



# ÎNMULȚIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \times b$

$$\begin{array}{r} 1110 \\ a = 14 \\ \times 0101 \\ \hline 1110 \\ 0000 \\ 1110 \\ 0000 \\ \hline 1000110 \\ s = 70 \end{array}$$

# ÎNMULȚIREA NUMERELEOR ÎNTREGI

- exemplu,  $s = a \times b$

$$\begin{array}{r} 1110 \\ a = 14 \\ \times 0101 \\ \hline 1110 \\ 0000 \\ 1110 \\ 0000 \\ \hline 1000110 \\ s = 70 \end{array}$$

ce am făcut aici este corect,  
dar am presupus că am primit  
numere naturale. ce se  
întâmplă dacă  $a$  și  $b$  sunt în  
complement față de doi?

# ÎNMULTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \times b$

$$\begin{array}{r} 1\ 1\ 1\ 0 \\ \hline \end{array} \quad a = -2$$

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ \hline \end{array} \quad b = 5$$

$$s = -10$$

primul pas: extindem operanzii pe 8 biți

# ÎNMULȚIREA NUMERELEOR ÎNTREGI

- exemplu,  $s = a \times b$

1 1 1 1 1 1 1 1 0

$$a = -2$$

0 0 0 0 0 1 0 1

$$b = 5$$

1 1 1 1 1 1 1 1 0

$$s = -10$$

0 0 0 0 0 0 0 0

al doilea pas: facem operația de înmulțire obișnuită

1 1 1 1 1 1 1 0

...

.....1 1 1 1 1 0 1 1 0

# ÎNMULȚIREA NUMERELEOR ÎNTREGI

- exemplu,  $s = a \times b$

1 1 1 1 1 1 1 1 0

$$a = -2$$

0 0 0 0 0 1 0 1

$$b = 5$$

1 1 1 1 1 1 1 1 0

$$s = -10$$

0 0 0 0 0 0 0 0

al treilea pas: rezultatul este pe  
8 biți în complement față de doi

1 1 1 1 1 1 1 0

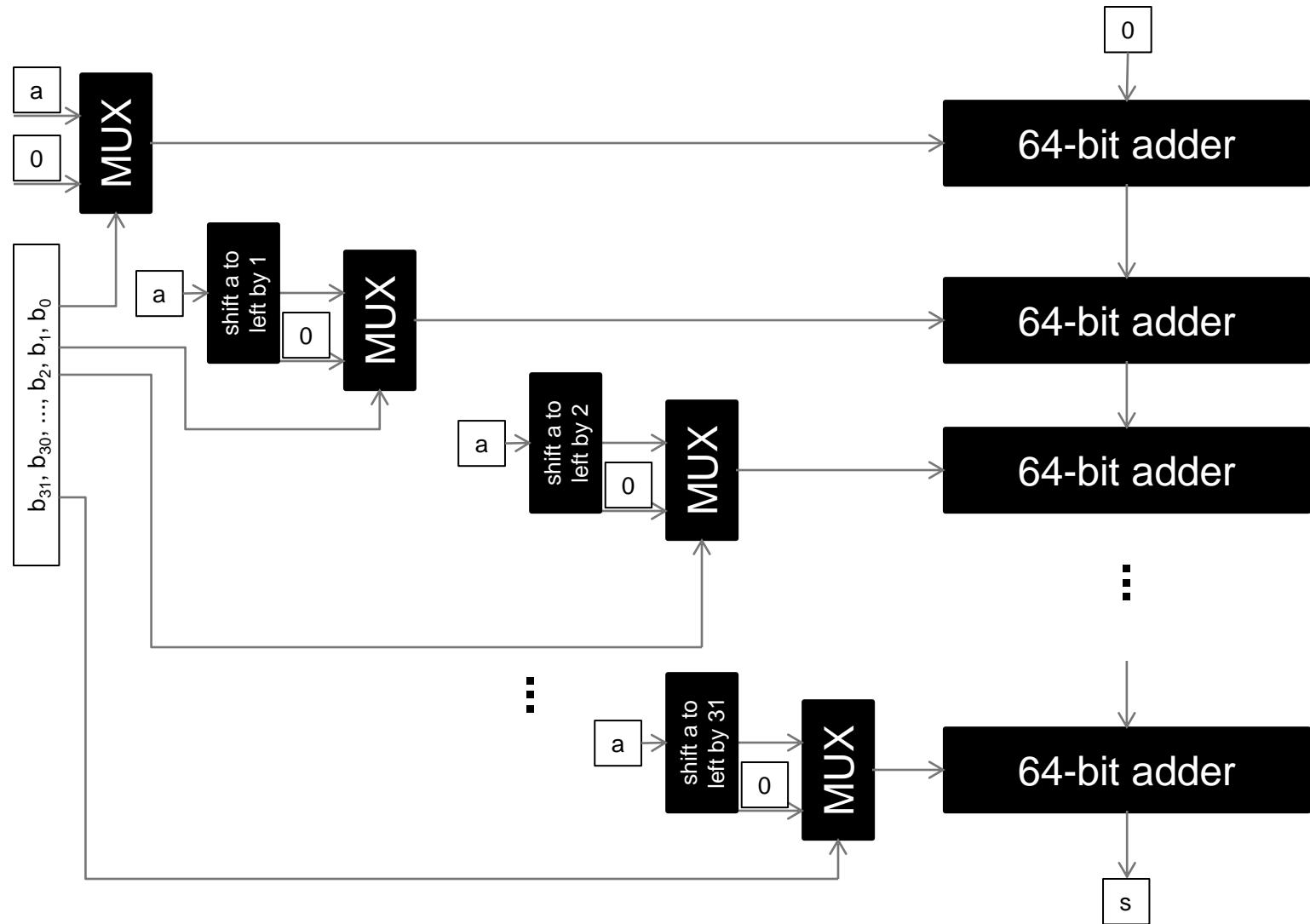
...

.....1 1 1 1 1 0 1 1 0

# ÎNMULTIREA NUMERELOR ÎNTREGI

- circuitul combinațional

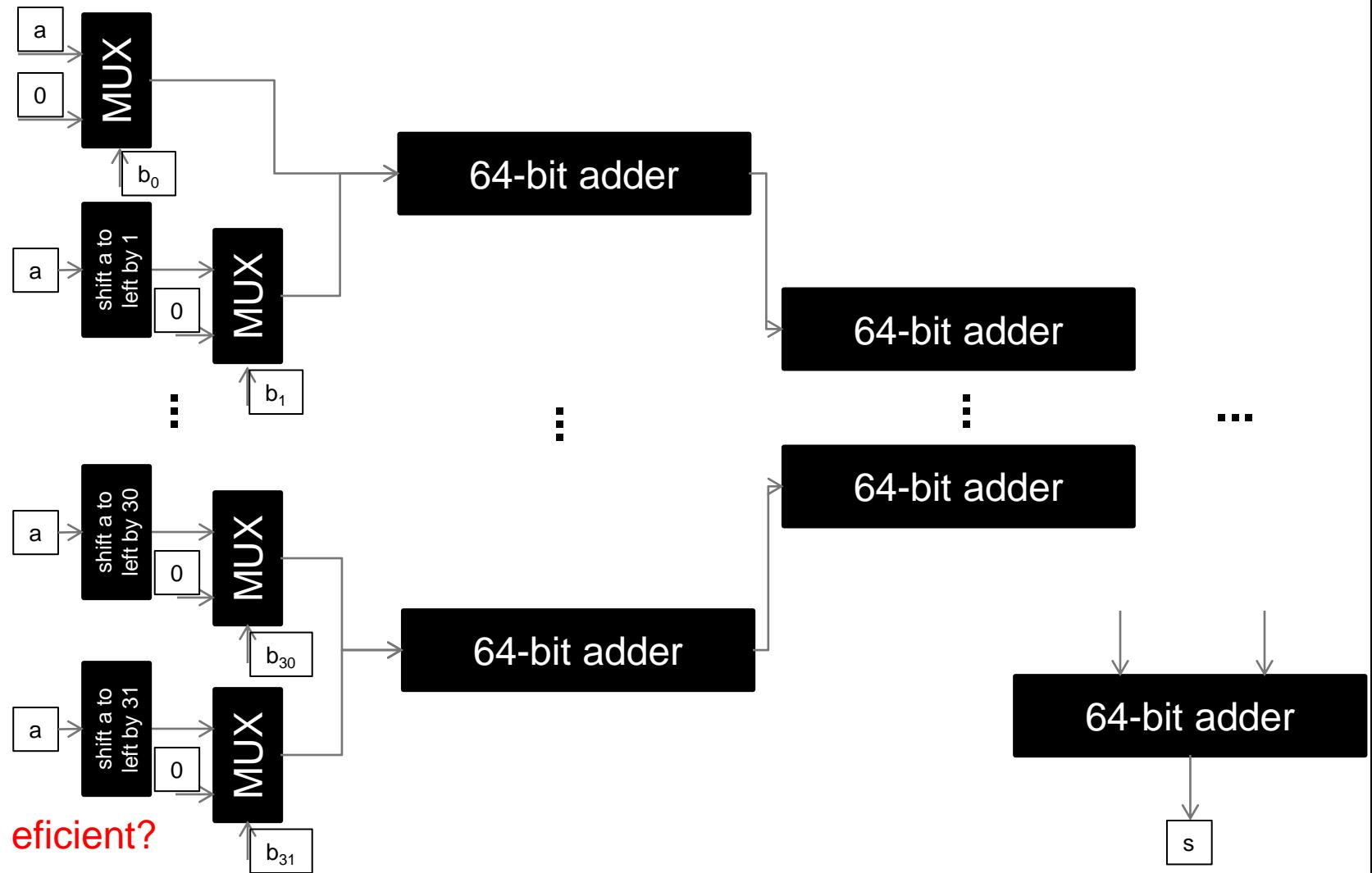
- $s = a \times b$ , a și b sunt numere pe 32 de biți



# ÎNMULTIREA NUMERELOR ÎNTREGI

- circuitul combinațional

- $s = a \times b$ , a și b sunt numere pe 32 de biți

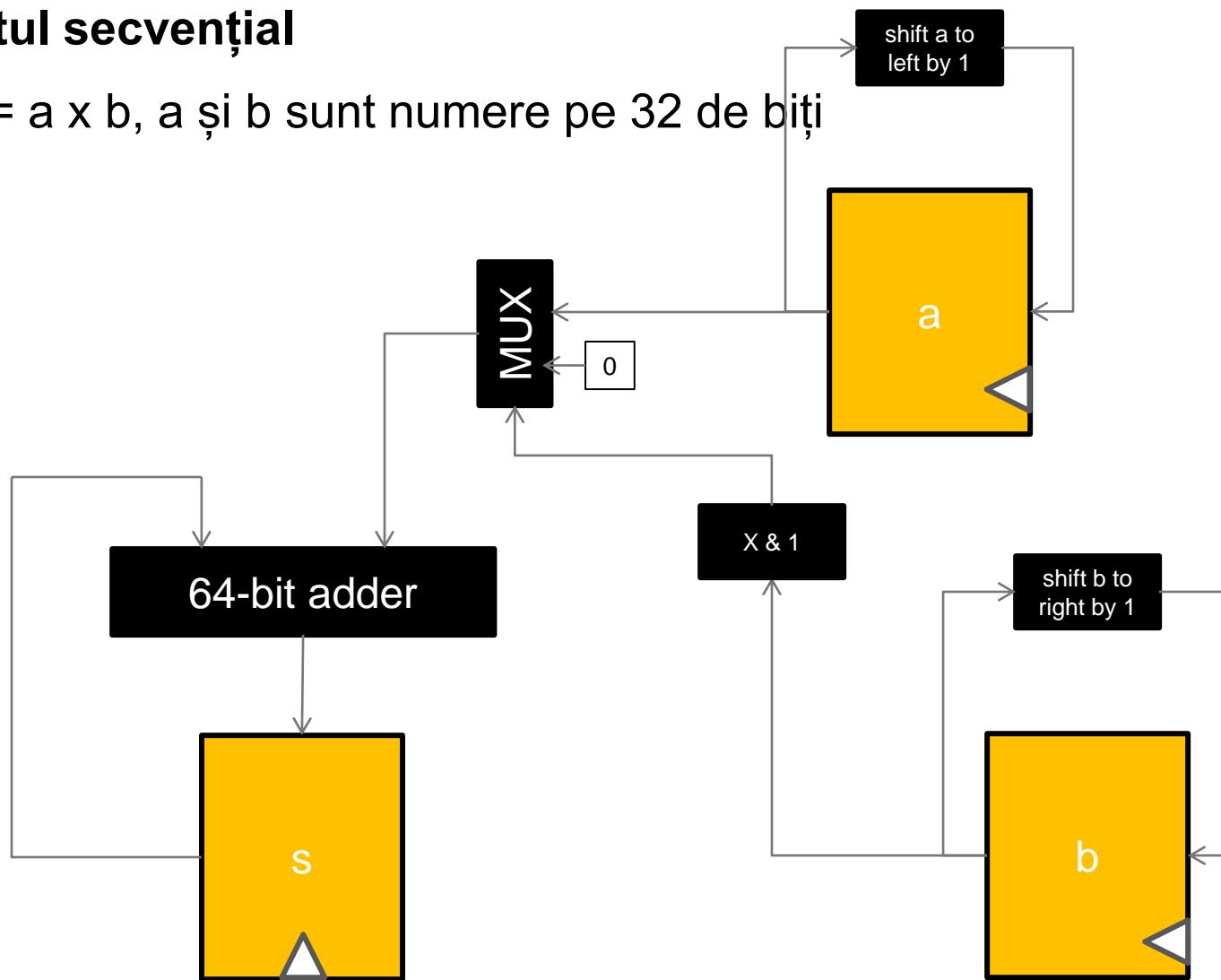


ceva mai eficient?

# ÎNMULTIREA NUMERELOR ÎNTRREGI

- **circuitul secvențial**

- $s = a \times b$ , a și b sunt numere pe 32 de biți



# ÎMPĂRTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \div b$

1 0 0 1 1 1

1 1

---

# ÎMPĂRTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \div b$

1	0	0	1	1	1
---	---	---	---	---	---

1	1
---	---

---

0
---

# ÎMPĂRTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \div b$

1	0	0	1	1	1
---	---	---	---	---	---

1	1
---	---

---

0	0
---	---

# ÎMPĂRTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \div b$

1 0 0 1 1 1

1 1

---

0 0 1

# ÎMPĂRTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \div b$

$$\begin{array}{r} & 1 & 1 & 1 & 1 \\ \overline{)1} & 1 & \\ \hline & 0 & 0 & 1 \end{array}$$

# ÎMPĂRTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \div b$

1	1	1	1
---	---	---	---

1	1
---	---

---

0	0	1	1
---	---	---	---

# ÎMPĂRTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \div b$

11

11

---

00110

# ÎMPĂRTIREA NUMERELOR ÎNTREGI

- exemplu,  $s = a \div b$

$$\begin{array}{r} 1\ 1 \\ \hline \end{array}$$

$$a = 39$$

$$\begin{array}{r} 1\ 1 \\ \hline \end{array}$$

$$b = 3$$

---

$$\begin{array}{r} 0\ 0\ 1\ 1\ 0\ 1 \\ \hline \end{array}$$

$$s = 13$$

# ÎMPĂRTIREA NUMERELOR ÎNTREGI

- $s = a \div b$ 
  - ce se întâmplă dacă  $a$  sau  $b$  sunt variabile negative?
  - rezultatul este negativ dacă  $a$  și  $b$  au semne diferite (XOR logic)
  - în general
    - $a = s \times b + r$
    - semnul lui  $r$  este semnul lui  $a$
- circuitul pentru împărțire nici nu vom încerca să îl facem
- din cauza acestei complexități ridicate, compilatoarele și sistemele de calcul vor face tot posibilul pentru a evita o împărțire
- vedem mai multe exemple la seminar ...

# REPREZENTAREA ÎN VIRGULĂ MOBILĂ

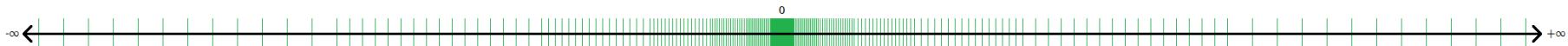
- am discutat la Seminar 0x00 despre reprezentarea în virgulă fixă

...	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$		$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$	$2^{-5}$	$2^{-6}$	$2^{-7}$	...
-----	-------	-------	-------	-------	-------	-------	-------	-------	--	----------	----------	----------	----------	----------	----------	----------	-----

- exemplu: 7.5 e scris ca 111.1
- care este problema cu această reprezentare?
  - partea întreagă este separată de partea fracționară
  - fiecare are nevoie de un număr de biți prestabilit
  - asta poate să fie ineficient
  - vrem ca numărul de biți total să fie alocat “dinamic”, în funcție de numărul pe care trebuie să îl reprezentăm

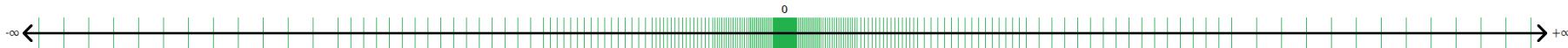
# REPREZENTAREA ÎN VIRGULĂ MOBILĂ

- când trebuie să reprezentăm un număr real
  - nu putem să avem precizie infinită
  - avem un număr finit de biți, deci putem să scriem biții în circuite
  - avem nevoie de precizie variabilă
  - putem avea precizie “infinită” dacă avem numere raționale (și vom salva separat numărătorul și numitor ca întregi)
- standardul: IEEE 754 Floating Point
  - densitatea nu este uniformă pe linia reală



# REPREZENTAREA ÎN VIRGULĂ MOBILĂ

- standardul: IEEE 754 Floating Point
  - densitatea nu este uniformă pe linia reală



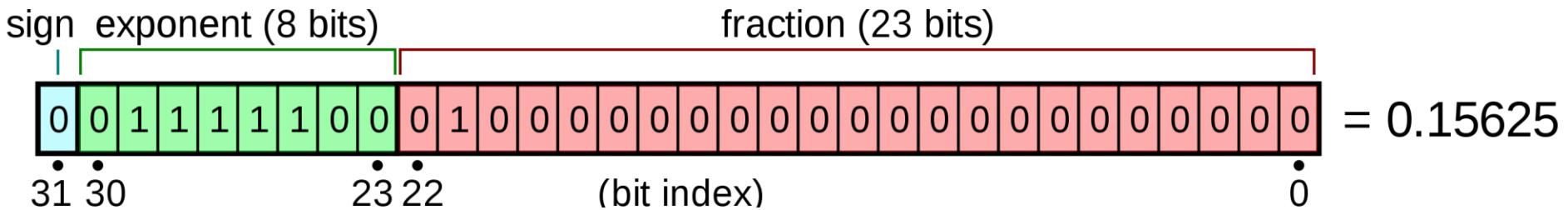
- sunt câteva consecințe
  - $(0.1 + 0.2) == 0.3$  versus  $(0.2 + 0.3) == 0.5$
  - $(0.7 + 0.2) + 0.1$  versus  $(0.7 + 0.1) + 0.2$  (nu avem asociativitatea)
  - $\text{math.sqrt}(3)*\text{math.sqrt}(3) == 3$  versus  $\text{math.sqrt}(3*3) == 3$
  - diferența cu numere întregi
    - dacă folosim tip de date întreg:  $16777216 + 1 = 16777217$
    - dacă folosim tip de date FP:  $16777216.0 + 1 = 16777216.0$
  - $\text{float}(123456789101112) + 1.0 = 123456789101113.0$
  - $\text{float}(1234567891011121) + 1.0 = 1234567891011122.0$
  - $\text{float}(12345678910111213) + 1.0 = 1.2345678910111212e+16$

# REPREZENTAREA ÎN VIRGULĂ MOBILĂ

- reprezentarea științifică
  - $12345 = 1.2345 \times 10^4$
  - $1024 = 1.024 \times 10^3$
  - $0.00125 = 1.25 \times 10^{-3}$
  - $\text{float}(1234567891011121) + 1.0 = 1234567891011122.0$
  - $\text{float}(12345678910111213) + 1.0 = 1.2345678910111212e+16$
  - $101010 = 1.0101 \times 2^4$ 
    - în sistemul binar, primul bit din reprezentare este mereu 1

# REPREZENTAREA ÎN VIRGULĂ MOBILĂ

- standardul: IEEE 754 Floating Point



- $x = (-1)^s \cdot 1.\text{mmmmmmmmmmmmmmmmmmmmmmmmmmmm} \cdot 2^{\text{eeeeeee}-127}$
- exemple:
  - $0.15625 = (-1)^0 \cdot 1.0100\dots0 \cdot 2^{b(01111100)-127} = 1.25 \cdot 2^{-3} = 1.25/8$
  - alte exemple:
    - $(-1)^0 \cdot 1.1000\dots0 \cdot 2^{b(01111100)-127} = 1.5 \cdot 2^{-3} = 1.5/8 = 0.1875$
    - $1 = (-1)^0 \cdot 1.00000000000000000000000 \cdot 2^{01111111-127}$
    - $-1 = (-1)^1 \cdot 1.00000000000000000000000 \cdot 2^{01111111-127}$
    - $2 = (-1)^0 \cdot 1.00000000000000000000000 \cdot 2^{10000000-127}$
    - $\infty = (-1)^0 \cdot 1.00000000000000000000000 \cdot 2^{11111111-127}$
    - $-\infty = (-1)^1 \cdot 1.00000000000000000000000 \cdot 2^{11111111-127}$

mai multe exemple, la Seminarul 0x03

# **CE AM FĂCUT ASTĂZI**

- logică secvențială + combinațională, un exemplu
- înmulțirea numerelor întregi binare
- împărțirea numerelor întregi binare
- reprezentarea numerelor în virgulă mobilă
- lucrul cu numerele în virgulă mobilă

# **DATA VIITOARE ...**

- **începem să discutăm despre arhitecturi de calcul**
- **începem să discutăm despre seturi de instrucțiuni**

# LECTURĂ SUPLIMENTARĂ

- PH book
  - 3.3 Multiplication
  - 3.4 Division
  - 3.5 Floating Point
- Computerphile, Floating Point Numbers,  
<https://www.youtube.com/watch?v=PZRI1IfStY0>
- Computeophile, Floating Point Numbers (Part1: Fp vs Fixed),  
<https://www.youtube.com/watch?v=f4ekifyijlg>
- Computeophile, Floating Point Numbers (Part2: Fp Addition),  
[https://www.youtube.com/watch?v=782QWNOD\\_Z0](https://www.youtube.com/watch?v=782QWNOD_Z0)



# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x06**

**SETURI DE INSTRUCȚIUNI, LIMBAJUL DE  
ASAMBLARE, ARHITECTURA CALCULATOARELOR**

Cristian Rusu

# **DATA TRECUTĂ**

- **logică secvențială, exemple**
- **înmulțirea numerelor întregi binare**
- **împărțirea numerelor întregi binare**
- **reprezentarea numerelor în virgulă mobilă**
- **operații cu numerele în virgulă mobilă**

# CUPRINS

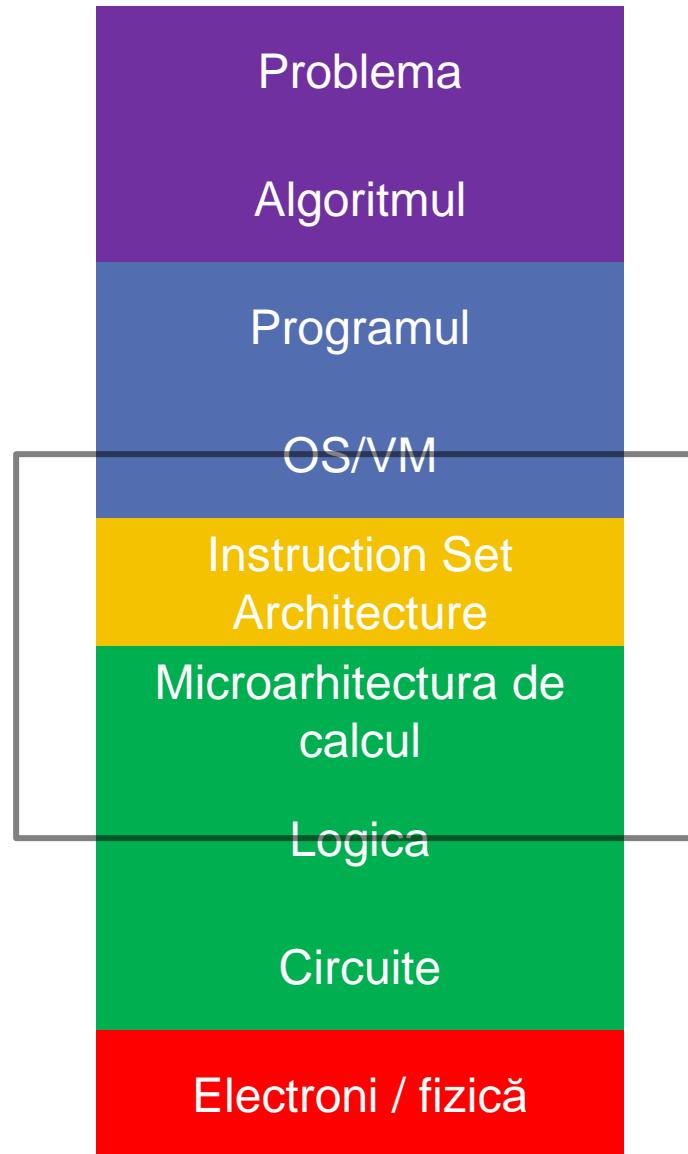
- Instruction Set Architecture (ISA)
- arhitectura de bază a calculatoarelor

# STRUCTURA CURSULUI - UNDE SUNTEM

- circuite digitale
  - teoria informației și abstractizarea digitală
  - funcții și circuite logice
- arhitecturi de calcul
  - seturi de instrucțiuni
  - limbajul assembly
  - compilatoare
  - pipelining
  - ierarhia memoriei
- organizarea calculatoarelor
  - unitatea de procesare centrală
  - performanța calculatoarelor
  - dispozitive periferice și întreruperi
  - calcul paralel

# STRUCTURA CURSULUI - UNDE SUNTEM

- “The purpose of computing is insight, not numbers.” (Richard Hamming)

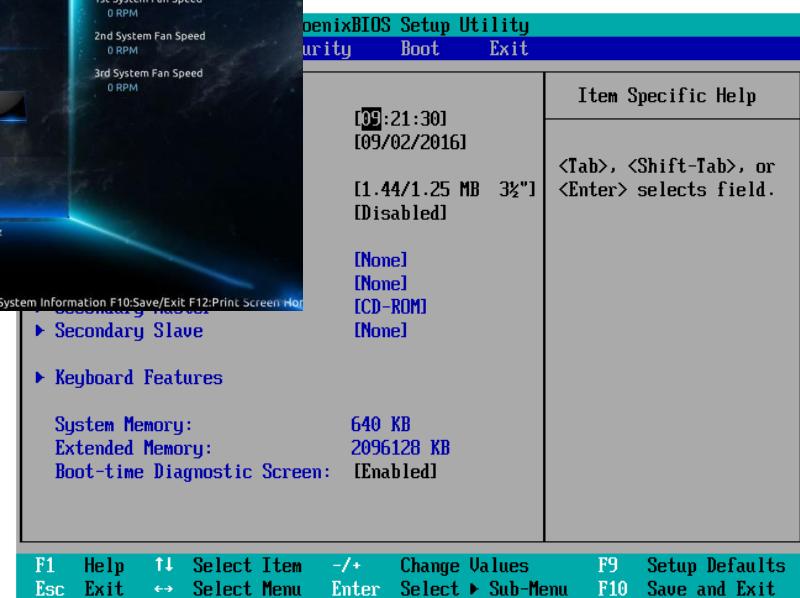
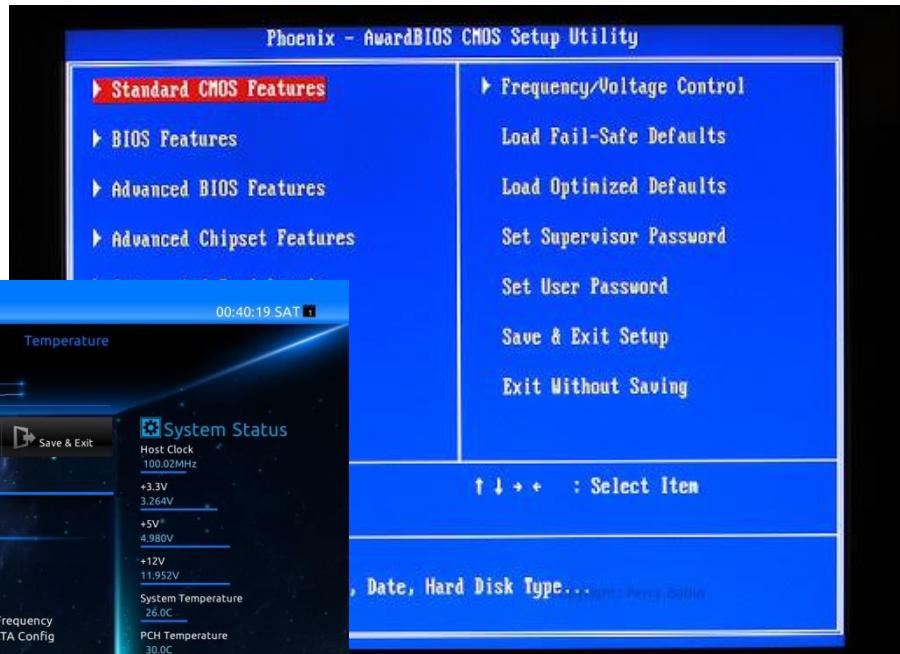


# OPERAȚII DE BAZĂ CU SISTEME DE CALCUL

- pornirea sistemului
  - în general, un buton de “power on” / “power off” **cum funcționează un astfel de buton? atât la pornire cât și la oprire?**
  - realizează alimentarea cu electricitate a componentelor
  - **CPU** este activat
  - CPU caută/pornește **BIOS** (Basic Input Output System)
    - testează componentele hardware (RAM, I/O, HD, etc.)
    - BIOS este scris în **ROM** (Read Only Memory) pe placă de bază
    - este scris într-un tip de memorie nevolatilă
    - pentru execuție, BIOS-ul este încărcat în RAM
    - BIOS știe cât e ceasul (**CMOS Real-Time Clock**) și hardware-ul
    - îl accesați automat când porniți calculatorul, fie cu F2 (în general)
  - CPU/BIOS pornesc Boot Code (caută sistemul de operare)
    - sistemul de operare este în general pe HD (poate fi și pe CD, stick)
    - sistemul de operare este încărcat în RAM pentru execuție

# OPERATII DE BAZĂ CU SISTEME DE CALCUL

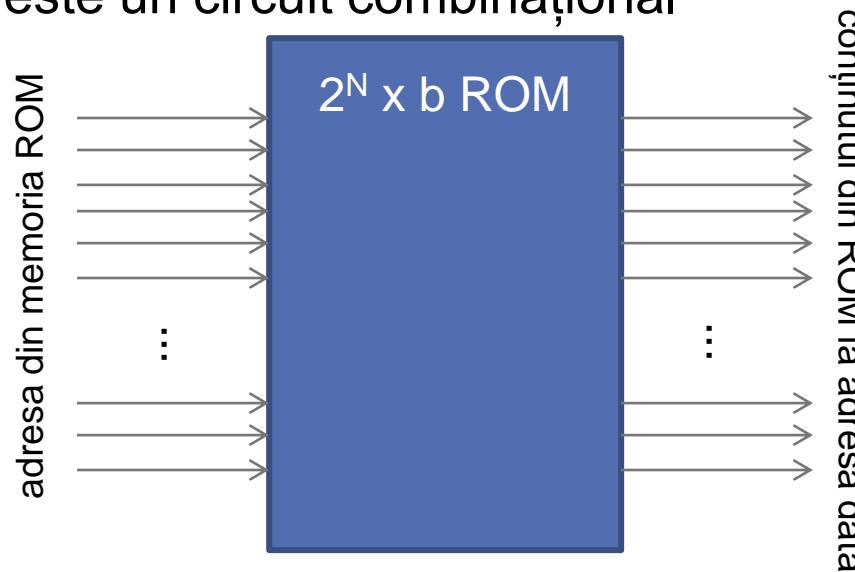
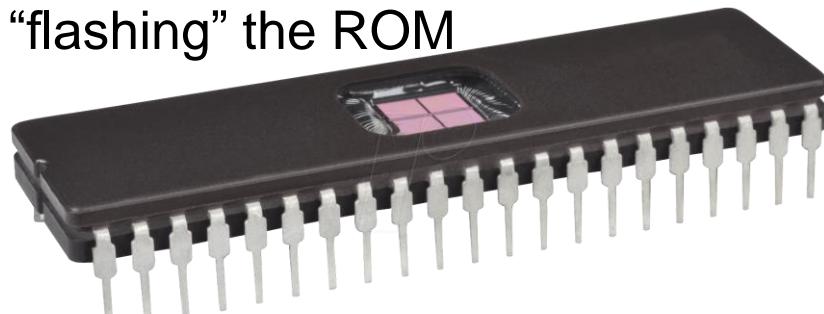
- BIOS



# OPERAȚII DE BAZĂ CU SISTEME DE CALCUL

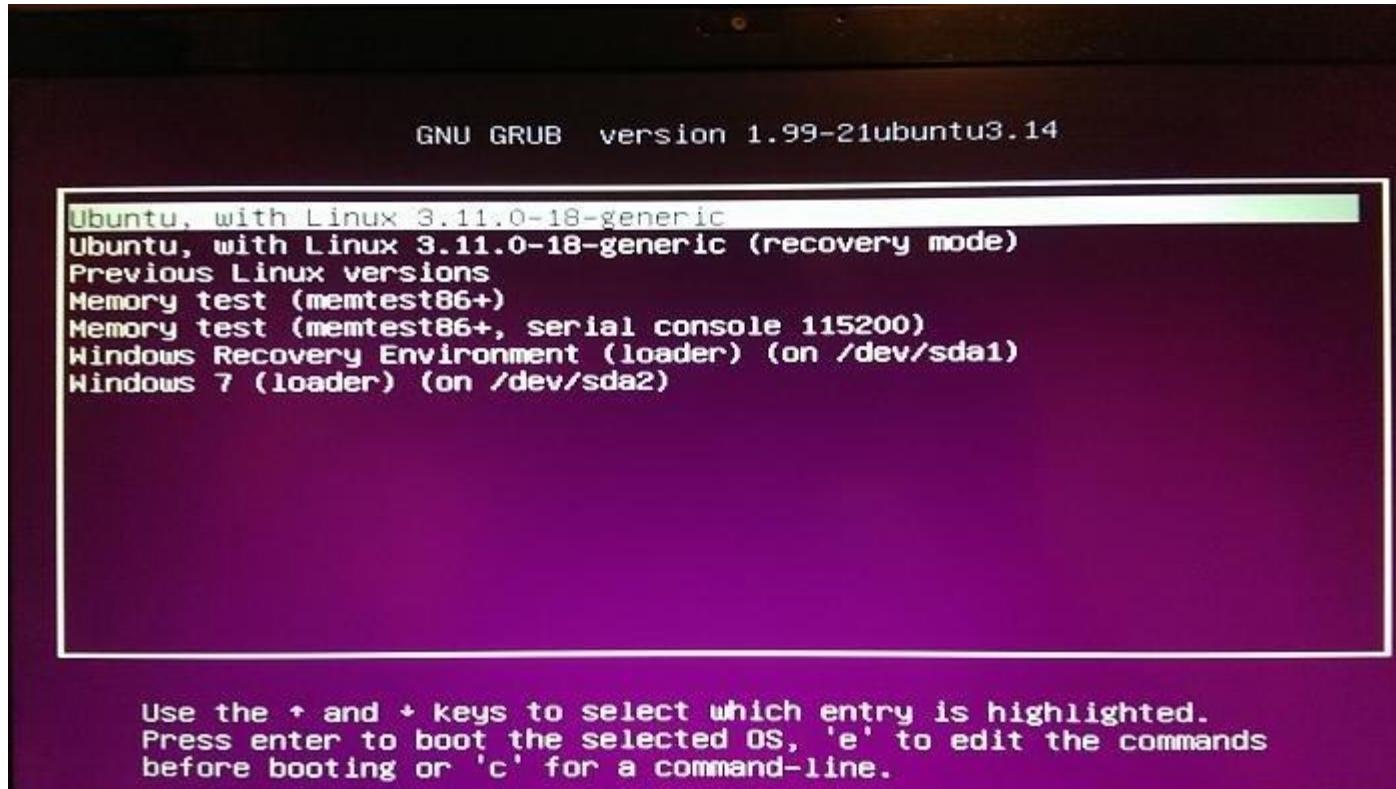
## • BIOS

- este scris în ROM (Read Only Memory)
- câteodată în Programmable ROM / Erasable Programmable ROM / Electrically Erasable Programmable ROM
- să scriem în ROM: “burning” sau “flashing” the ROM
- este un tip de *firmware*
- în general este un circuit combinațional



# OPERAȚII DE BAZĂ CU SISTEME DE CALCUL

- mai multe sisteme de operare pe același sistem de calcul



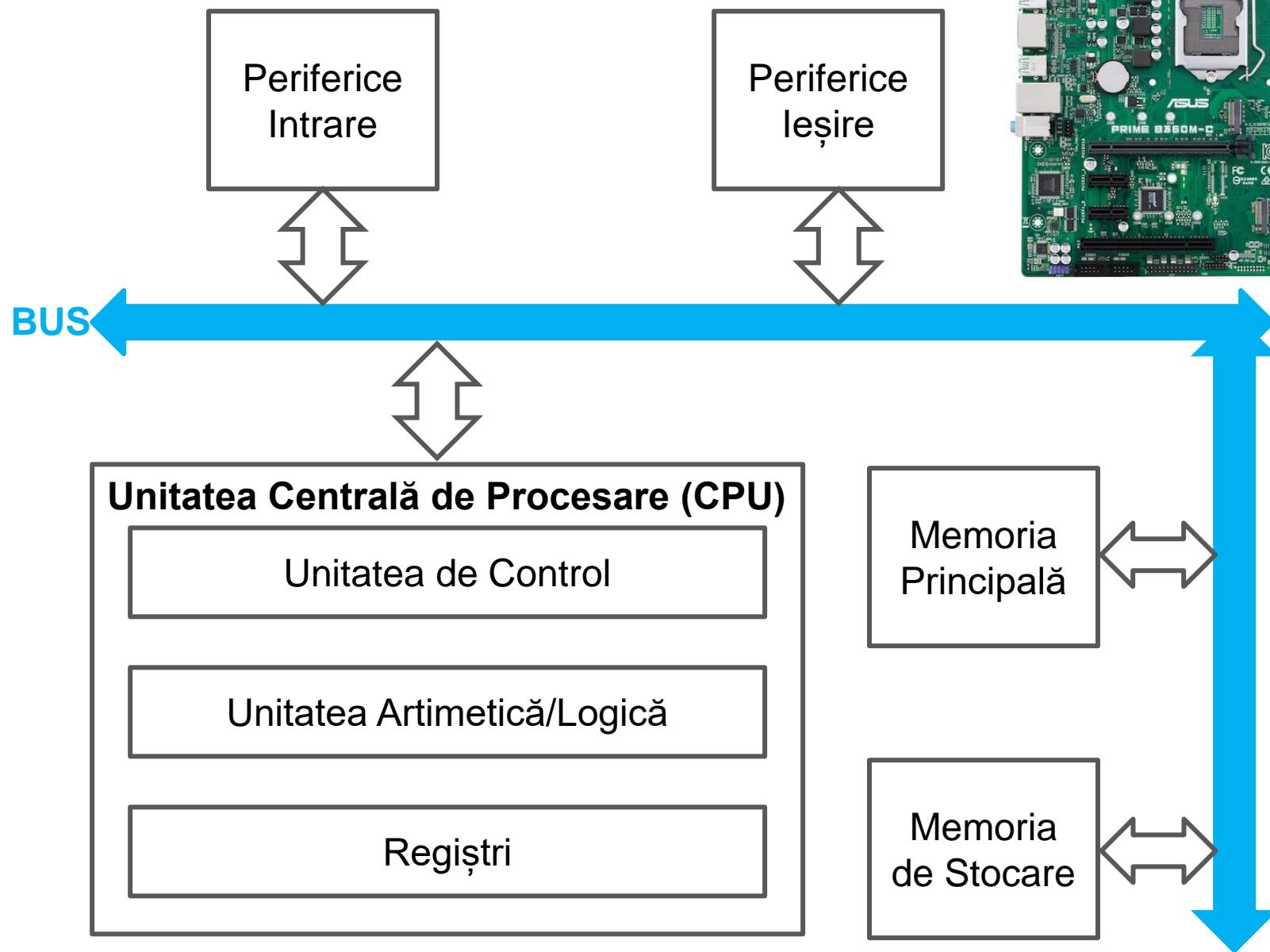
# OPERAȚII DE BAZĂ CU SISTEME DE CALCUL

- OS-ul preia controlul de la BIOS
  - din acest moment doar OS-ul are acces direct la periferice
  - accesul este realizat prin *drive*
  - OS-ul oferă o imagine abstractizată a memoriei pentru fiecare proces pornit
- din momentul în care OS-ul pornește, sistemul de calcul intră în ciclul obișnuit de procesare (secvența de boot s-a terminat)

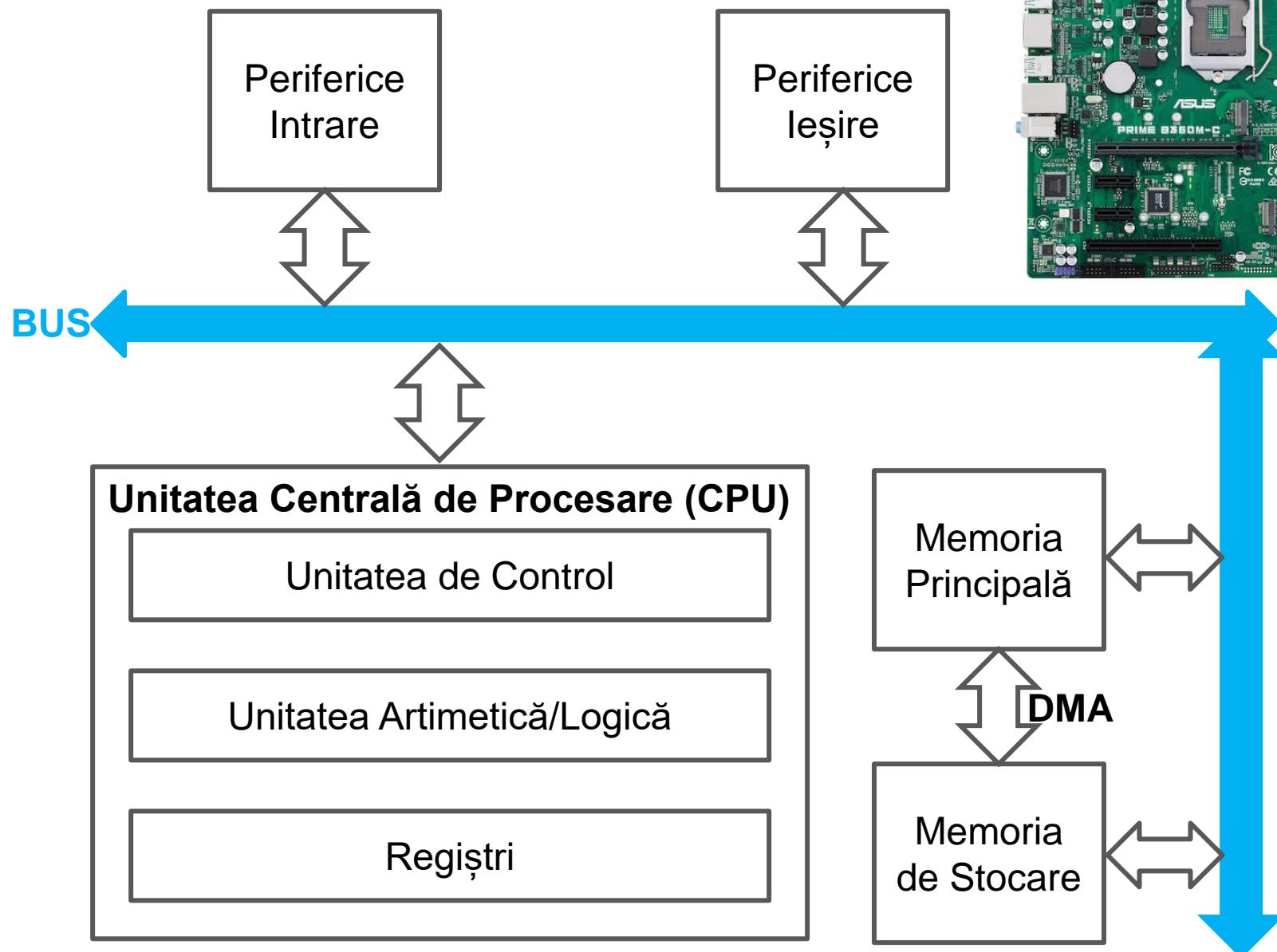
# ARHITECTURA DE BAZĂ

- **un sistem de calcul trebuie să fie capabil:**
  - să calculeze
    - să execute instrucțiuni
  - să comunice
    - să transfere biți între componente electronice
  - să stocheze
    - date care să fie folosite de instrucțiuni
    - instrucțiuni pentru execuție

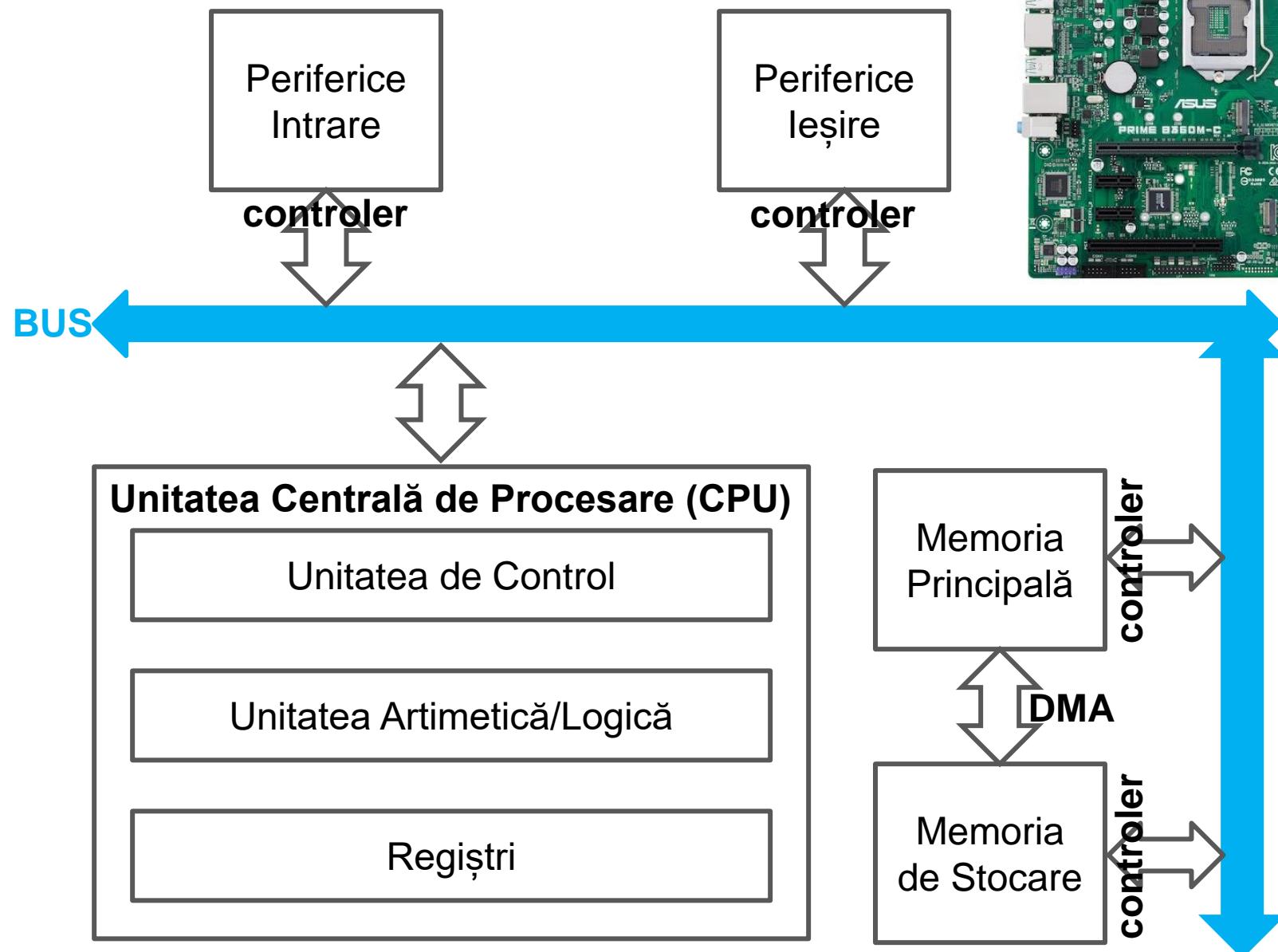
# ARHITECTURA DE BAZĂ



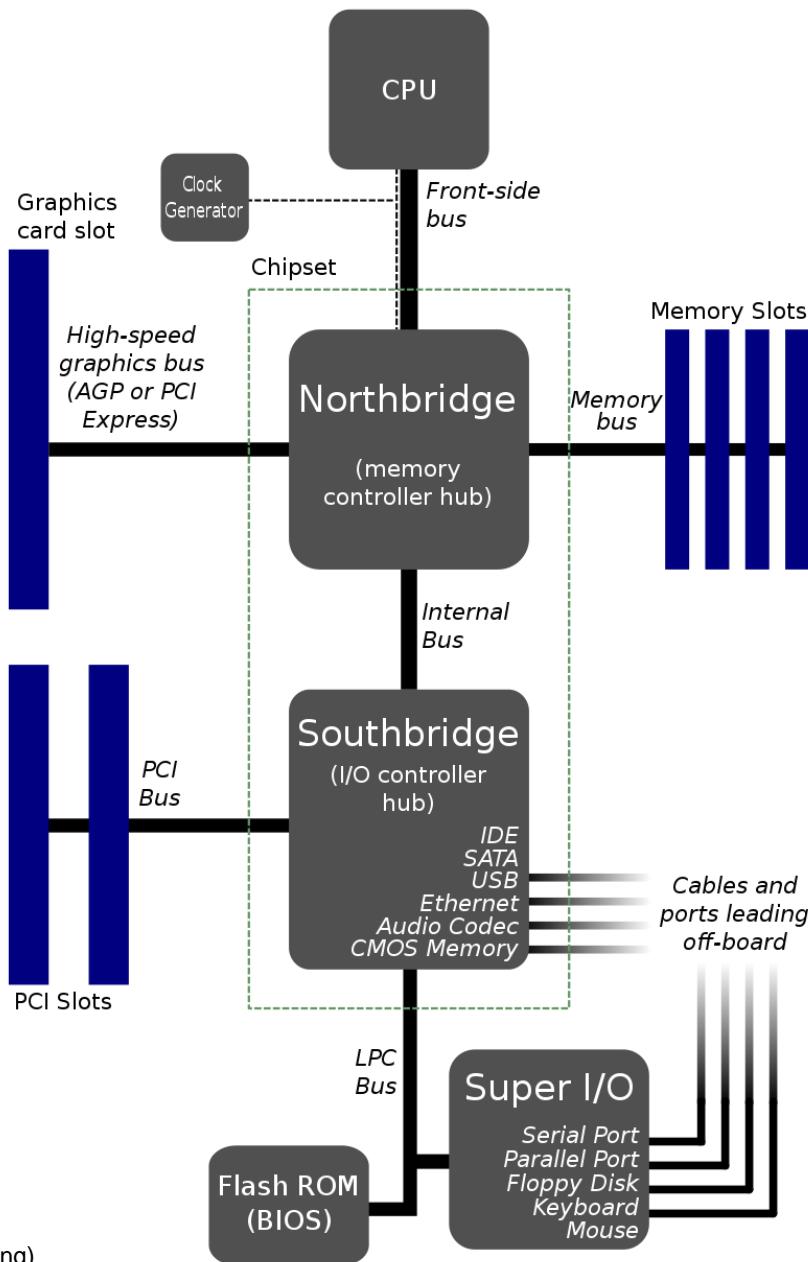
# ARHITECTURA DE BAZĂ



# ARHITECTURA DE BAZĂ



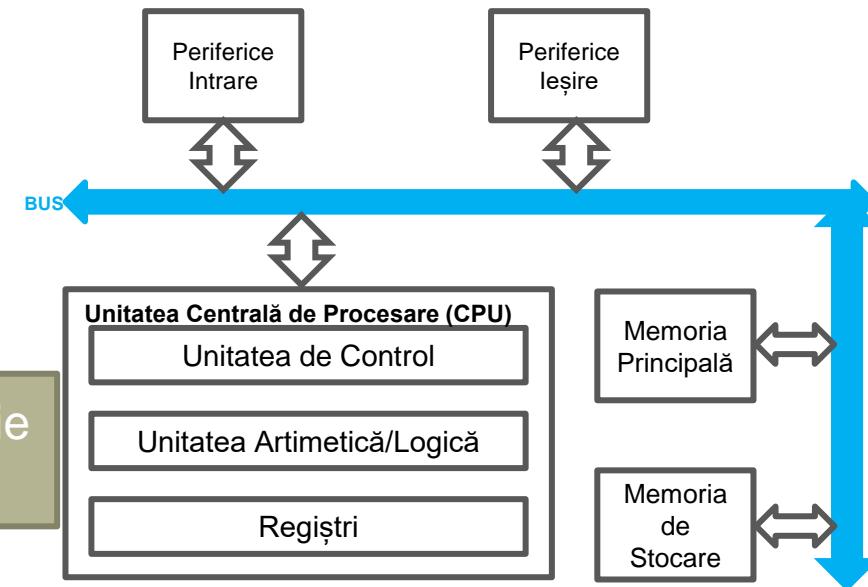
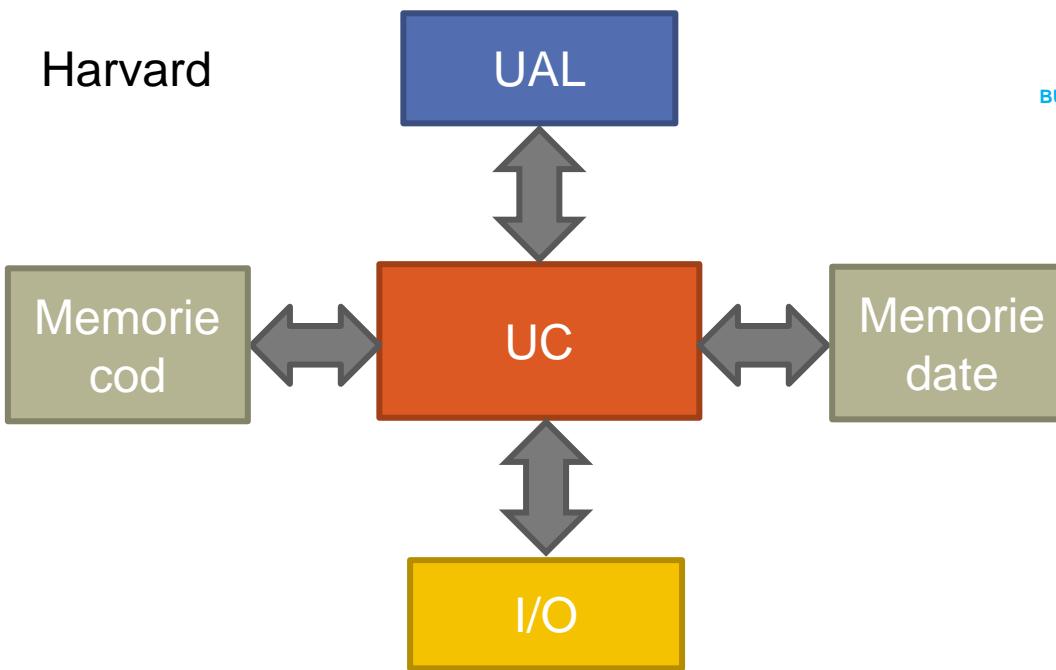
# ARHITECTURA DE BAZĂ



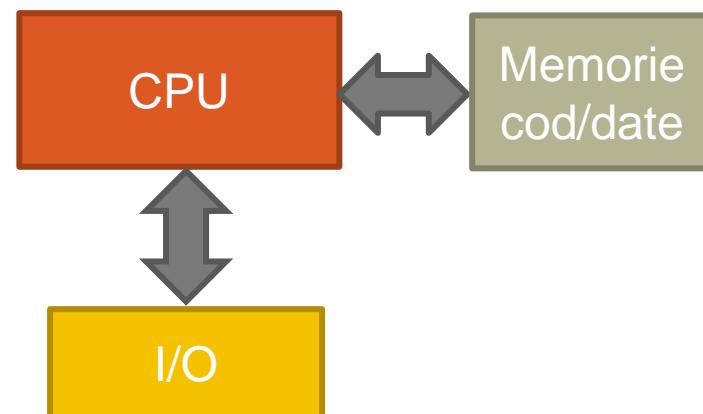
# ARHITECTURA DE BAZĂ

- Tipul arhitecturii de calcul

Harvard



von Neumann

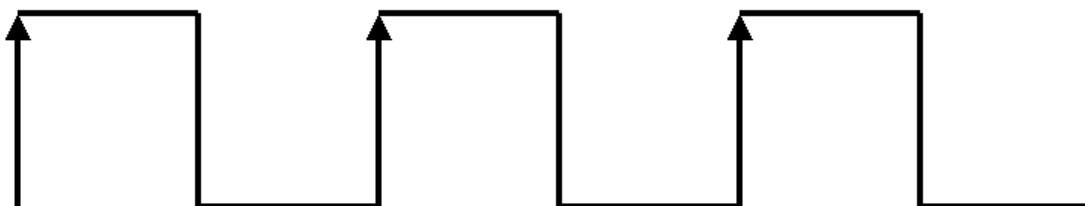
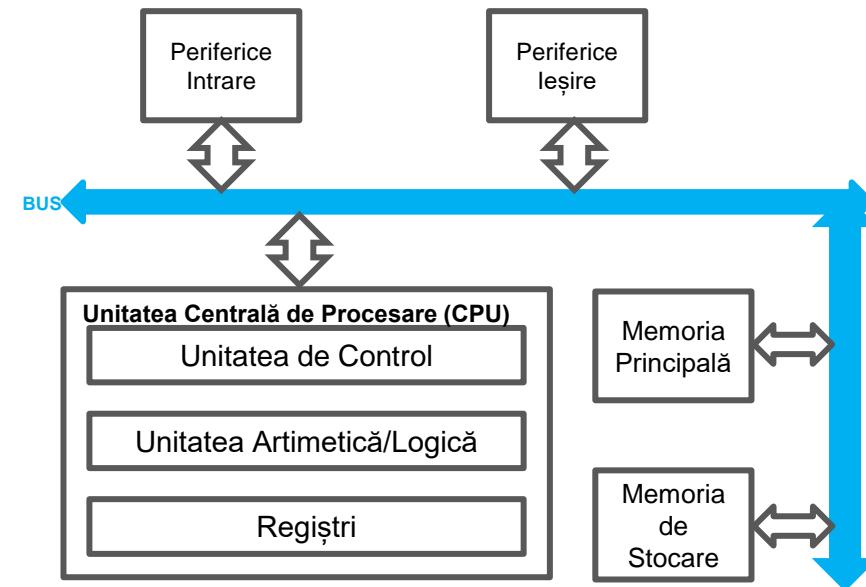


calculatoarele recente încep să nu mai fie von Neumann

# ARHITECTURA DE BAZĂ

- **Unitatea Centrală de Procesare**

- a.k.a. CPU
- este “creierul” unității de calcul
- execută instrucțiuni
- 5 componente principale:
  - **Clock**
    - este un circuit special care generează “ceasul”

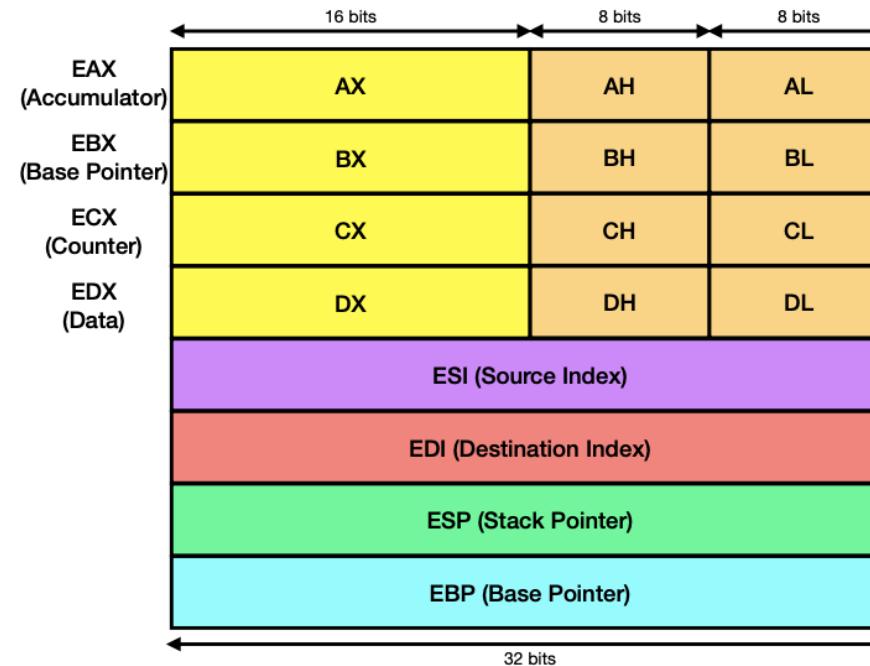
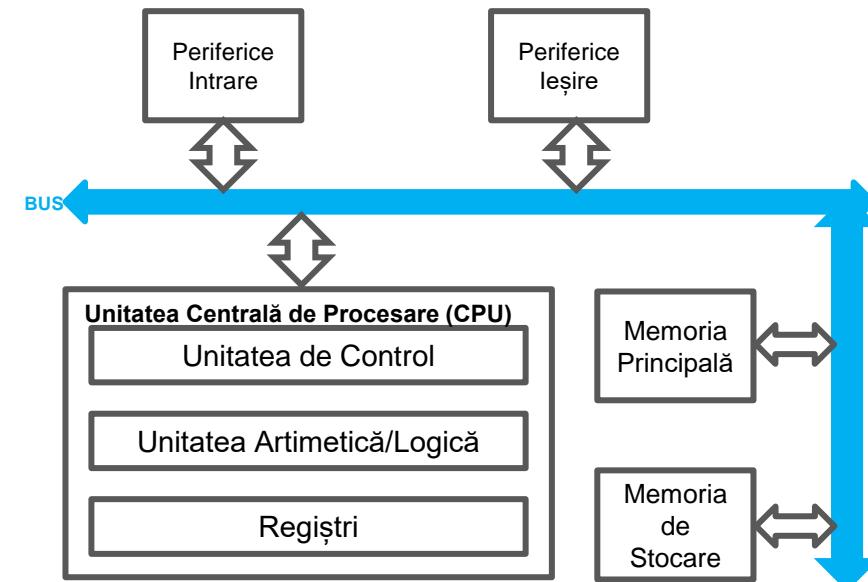


- este frecvența la care operează (calcule și sincronizarea componentelor secvențiale) CPU-ul
- cu cât este mai mare frecvența, cu atât mai bine (în general)
- se măsoară în MHz sau GHz

# ARHITECTURA DE BAZĂ

- Unitatea Centrală de Procesare

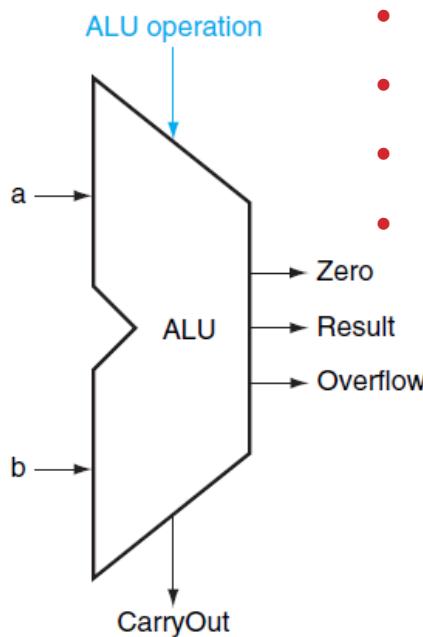
- a.k.a. CPU
- este “creierul” unității de calcul
- execută instrucțiuni
- 5 componente principale:
  - regiștri (“memoria”)



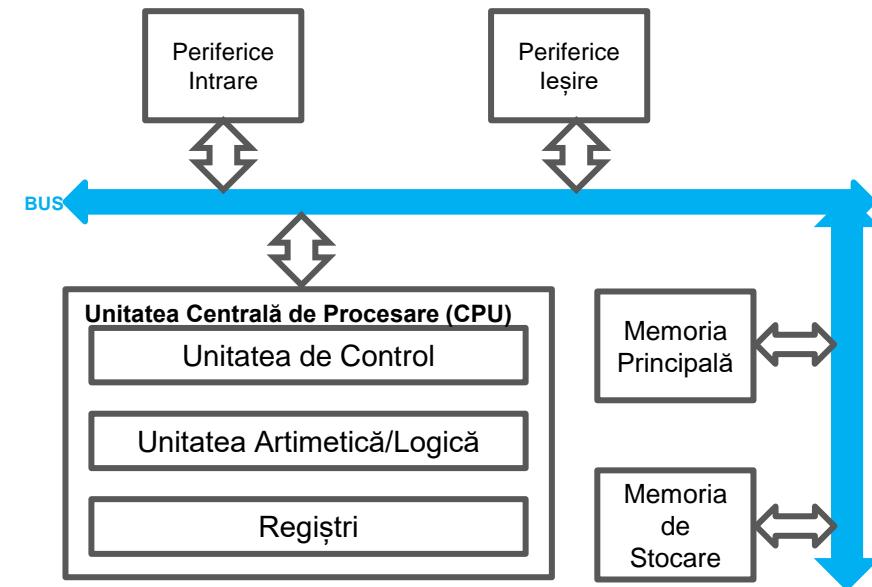
# ARHITECTURA DE BAZĂ

- Unitatea Centrală de Procesare

- a.k.a. CPU
- este “creierul” unității de calcul
- execută instrucțiuni
- 5 componente principale:
  - **UAL (“operații”)**



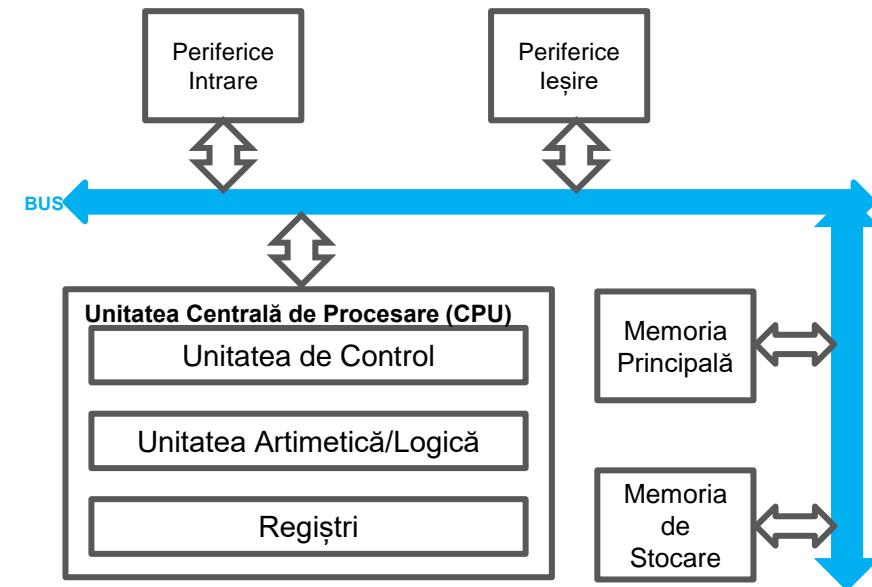
după orice operație, știm  
“gratuit” dacă rezultatul a fost  
sau nu zero – este folositor?



# ARHITECTURA DE BAZĂ

- **Unitatea Centrală de Procesare**

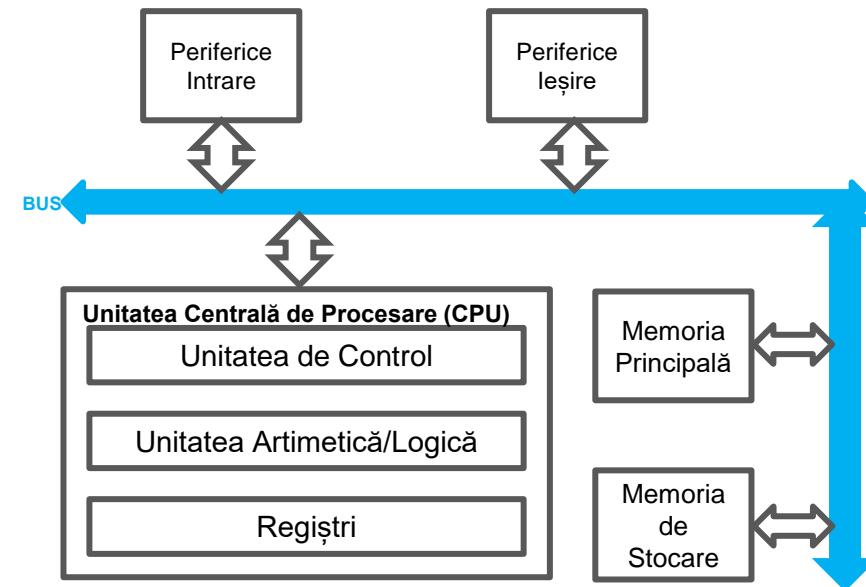
- a.k.a. CPU
- este “creierul” unității de calcul
- execută instrucțiuni
- 5 componente principale:
  - **BUS**
    - CPU are nevoie de șiruri de biți din memoria principală sau cea de stocare
    - CPU are nevoie să scrie înapoi în memorie rezultate
    - CPU coordonează perifericele



# ARHITECTURA DE BAZĂ

- **Unitatea Centrală de Procesare**

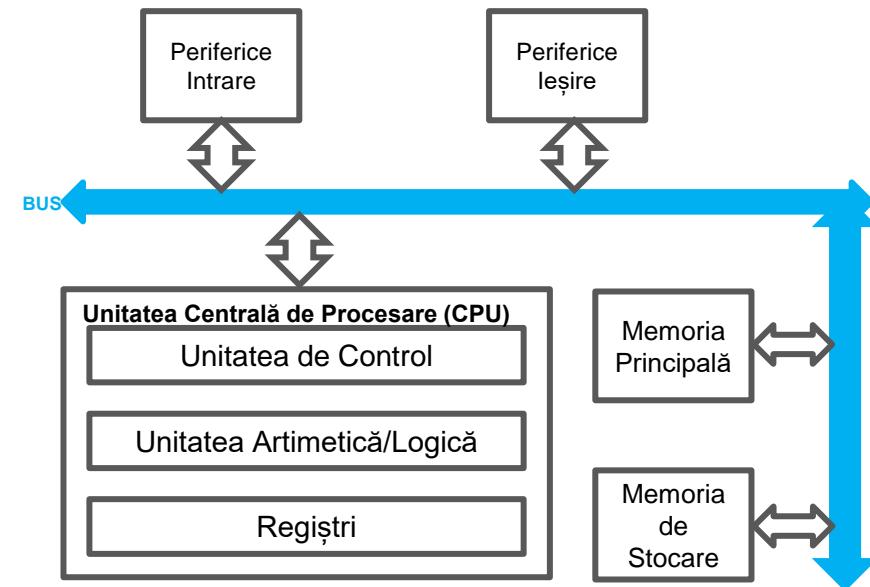
- a.k.a. CPU
- este “creierul” unității de calcul
- execută instrucțiuni
- 5 componente principale:
  - **UC (“instrucțiunile”)**
    - fetch
      - citim din memorie codul care trebuie executat
      - de unde din memorie? Instruction Pointer ne spune
    - decode
      - circuitul “Instruction Decoder” analizează biții citiți din memorie ca să “înțeleagă” ce să facă cu ei
    - execute
      - execută instrucțiunea decodată
      - poate duce la schimbarea IP sau la transmiterea ceva pe BUS către memorie
    - calculează următorul IP



# ARHITECTURA DE BAZĂ

- **Unitatea Centrală de Procesare**

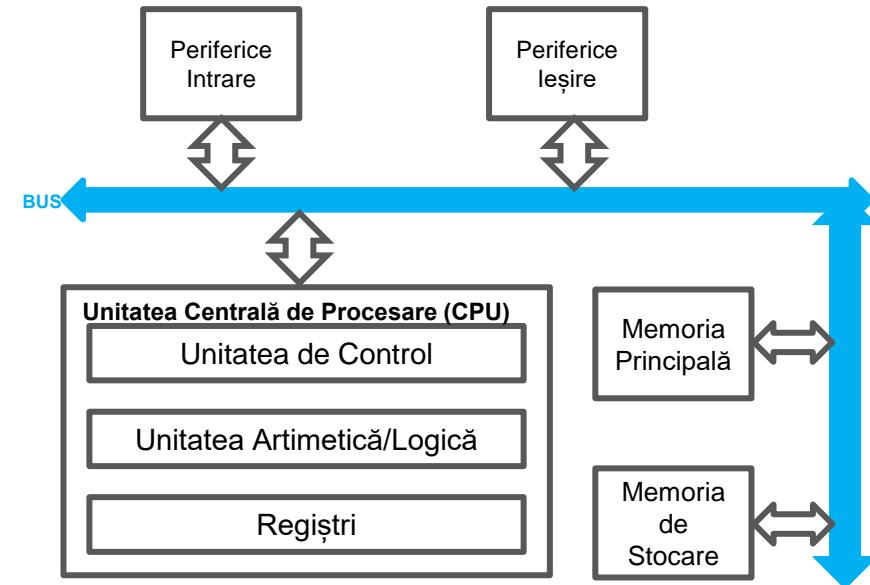
- a.k.a. CPU
- este “creierul” unității de calcul
- execută instrucțiuni
- 5 componente principale:
  - **UC (“instrucțiunile”)**
    - fetch
      - IP = 10011 (locația în memorie de unde să citim biții)
      - după citire, IP este actualizat
    - decode
      - s-a citit “11000110” care este decodat în
        - opcode = 110, operand1 = 00 operand2 = 110
        - de exemplu: 110 = “adună valoarea imediată A la registrul R”, R = 00 este EAX (prin convenție), A = 110 (adică 6)
    - execute
      - trimite  $EAX \leftarrow EAX + 6$  la UAL
      - citește rezultatul din UAL și pune-l în registrul EAX



# ARHITECTURA DE BAZĂ

- **Unitatea Centrală de Procesare**

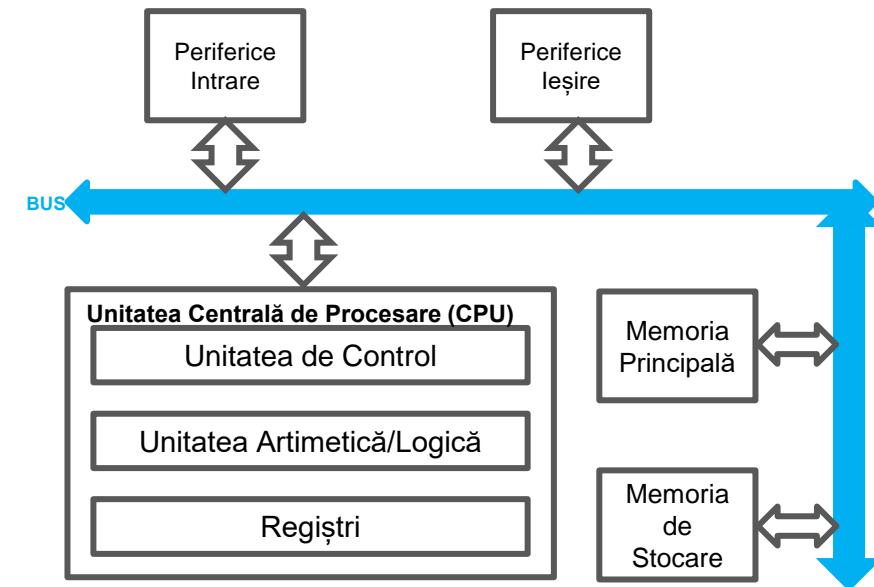
- a.k.a. CPU
- este “creierul” unității de calcul
- execută instrucțiuni
- 5 componente principale:
  - **UC (“instrucțiunile”)**
    - fetch
      - IP = 10011 (locația în memorie de unde să citim biții)
      - după citire, IP este actualizat
    - decode
      - s-a citit “1110011” care este decodat în
        - opcode = 111, operand1 = 00 operand2 = 11
        - de exemplu: 111 = “adună registrul A la registrul R”, R = 00 este EAX, A = 11 este EDX (prin convenție)
    - execute
      - trimite  $EAX \leftarrow EAX + EDX$  la UAL
      - citește rezultatul din UAL și pune-l în registrul EAX



# ARHITECTURA DE BAZĂ

- Unitatea Centrală de Procesare

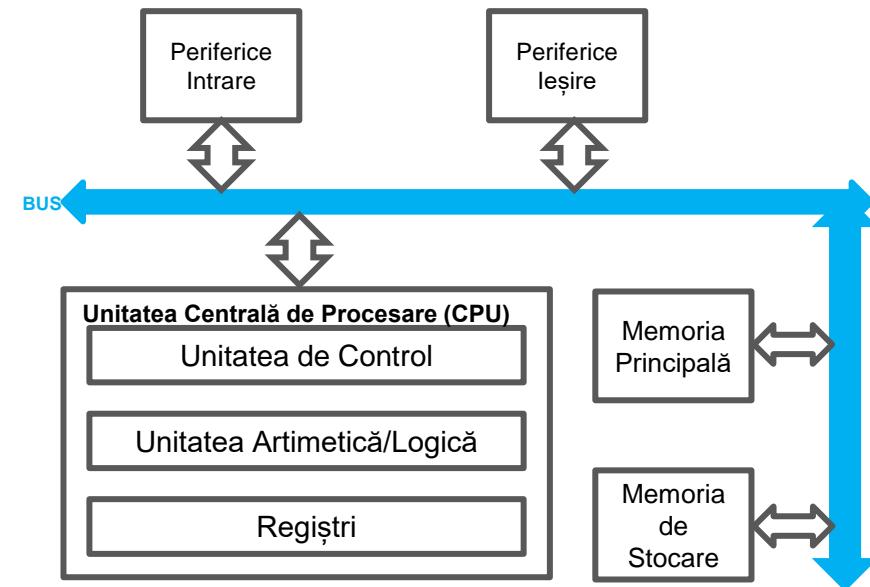
- a.k.a. CPU
- este “creierul” unității de calcul
- execută instrucțiuni
- 5 componente principale:
  - UC (“instructiunile”)



IF – Instruction Fetch (citirea din memorie a instructiunilor)  
ID – Instruction Decode (circuit secvențial care decodează)  
EX – Execute (execuția propriu-zisă)  
MEM – Memory Access (orice access memorie)  
WB – Write Back (scrie rezultatul înapoi în memorie)

# ARHITECTURA DE BAZĂ

- Unitatea Centrală de Procesare
  - a.k.a. CPU
  - este “creierul” unității de calcul
  - execută instrucțiuni
- 5 componente principale:
  - Clock
  - regiștri (“memoria”)
  - UAL (“operații”)
  - BUS
  - UC (“instrucțiunile”)

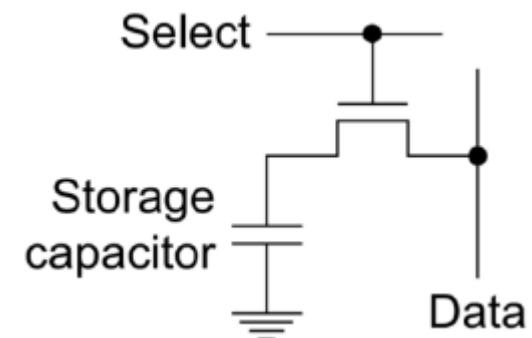
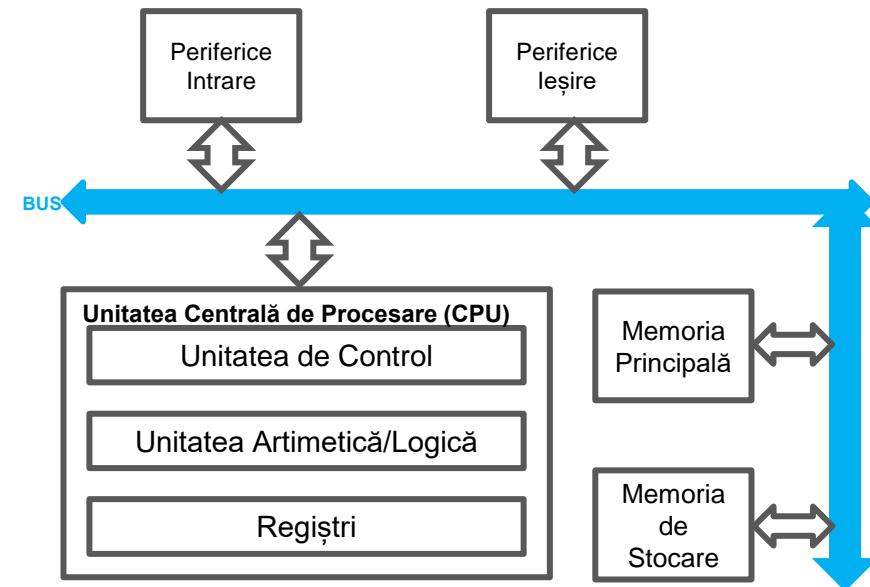


## PROCESOR

Producător procesor	Intel®
Tip procesor	i9
Model procesor	9880H
Arhitectură	Coffee Lake
Număr nuclei	8
Frecvență nominală	2.3 GHz
Cache	16384 KB
Frecvență Turbo Boost	4.8 GHz
Tehnologie procesor	14 nm
Procesor grafic integrat	Intel® UHD Graphics 630

# ARHITECTURA DE BAZĂ

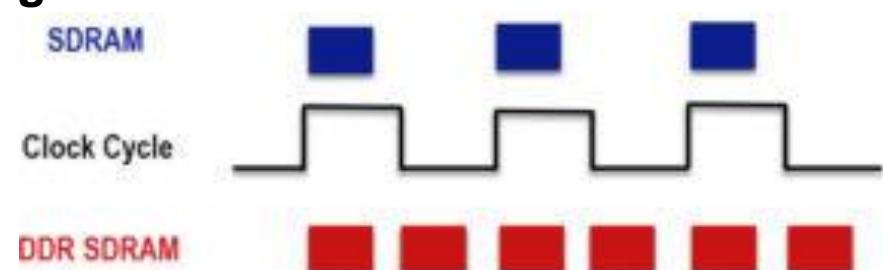
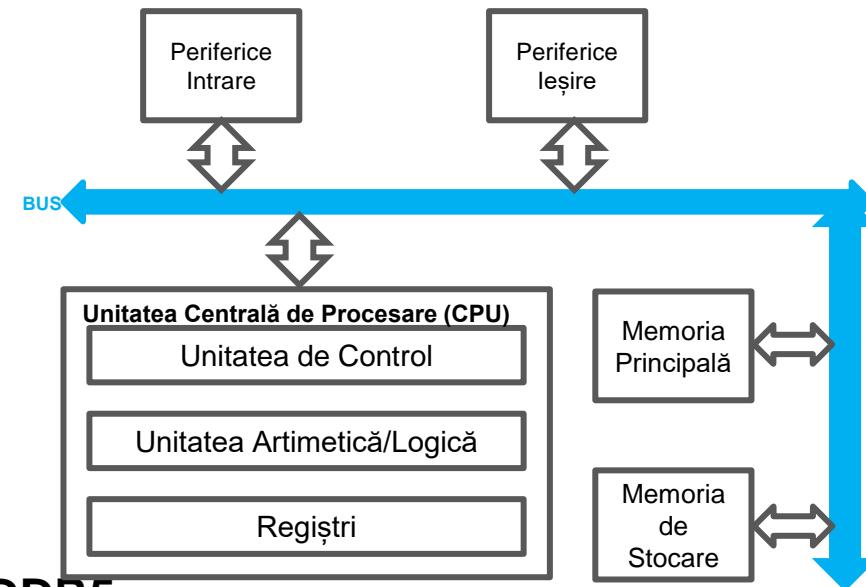
- Memoria Principală
  - conține cod și date
  - este volatilă
- Static RAM (SRAM)
  - bazată pe flip-flops
  - rapid
  - scump
  - registrii din CPU sunt de același tip
- Dynamic RAM (DRAM)
  - fiecare bit este reprezentat de o combinație tranzistor + condensator
  - condensatoarele suferă de leakage (scurgeri de tensiune)
  - DRAM trebuie actualizat o dată fiecare câteva zeci de ms



de ce este DRAM mai ieftin decât SRAM?  
are DRAM niște dezavantaje în comparație cu SRAM?

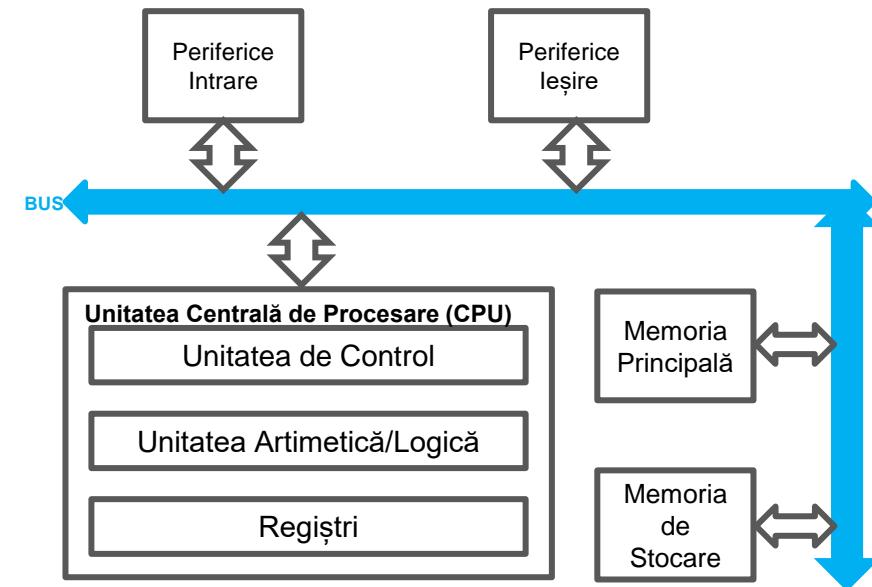
# ARHITECTURA DE BAZĂ

- Memoria Principală
  - conține cod și date
  - este volatilă
- DDR RAM
  - Double Data Rate RAM
  - DDR1/DDR2/DDR3/DDR4/DDR5
  - performanța este definită de:
    - capacitate
    - dacă au un sistem intern de corectarea erorilor (ECC)
    - timpi de acces (în cât timp de la comanda de citire de biți din RAM avem datele disponibile?, timpul de refresh)
    - consumul de energie



# ARHITECTURA DE BAZĂ

- Memoria Principală
  - conține cod și date
  - este volatilă
- ce avem azi?



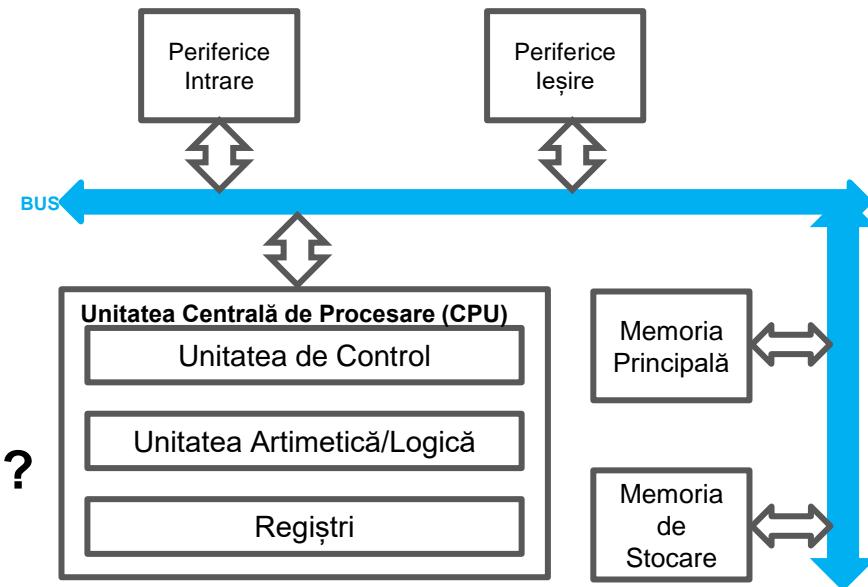
DDR4 Specs MHz

Friendly name	Industry name	Peak Transfer Rate	Data transfers/second (in millions)
DDR4-2400	PC4-19200	19200 MB/s	2400
DDR4-2666	PC4-21300	21300 MB/s	2666
DDR4-2933	PC4-23400	23400 MB/s	2933
DDR4-3000	PC4-24000	24000 MB/s	3000
DDR4-3200	PC4-25600	25600 MB/s	3200
DDR4-3600	PC4-28800	28800 MB/s	3600
DDR4-4000	PC4-32000	32000 MB/s	4000
DDR4-4400	PC4-35200	35200 MB/s	4400

fiecare transferă 64  
biți deodată

# ARHITECTURA DE BAZĂ

- **Memoria Principală**
  - **conține cod și date**
  - **este volatilă**
- **ce ne interesează la memorie?**

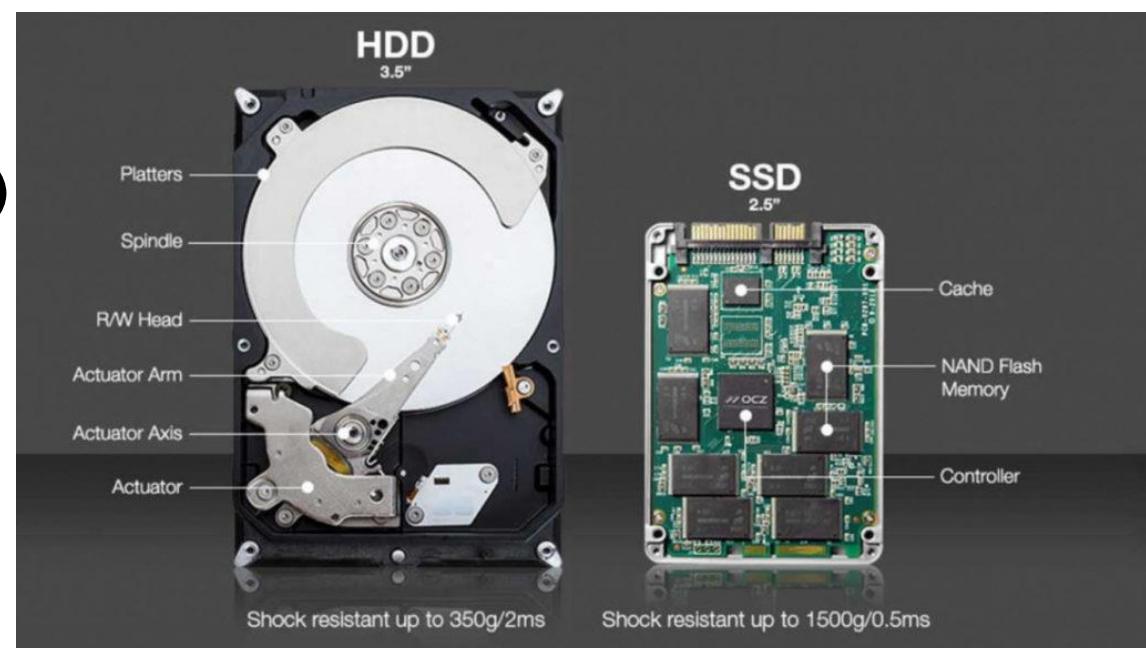
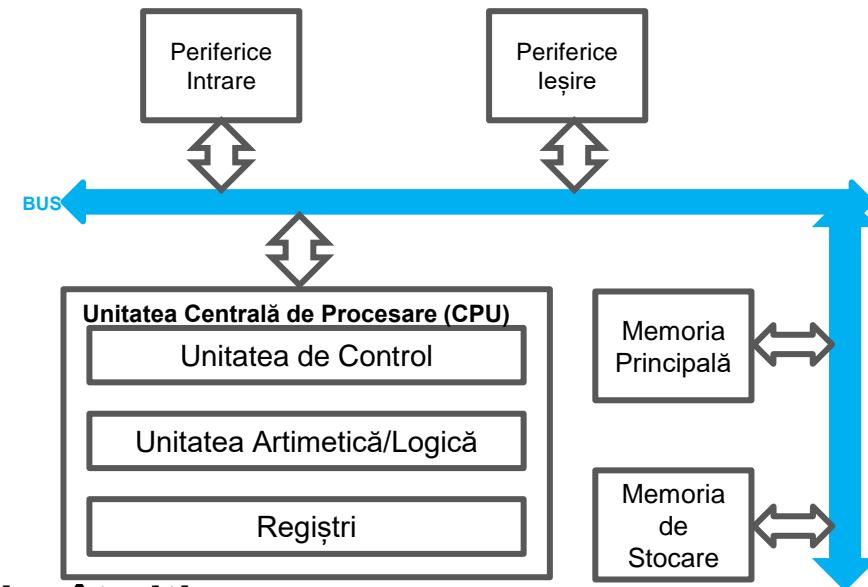


## MEMORIE

Capacitate memorie	32 GB
Tip memorie	DDR4
Numar sloturi	4
Sloturi ocupate	2
Frecvență	2666 MHz
Capacitate memorie maxima suportată	128 GB

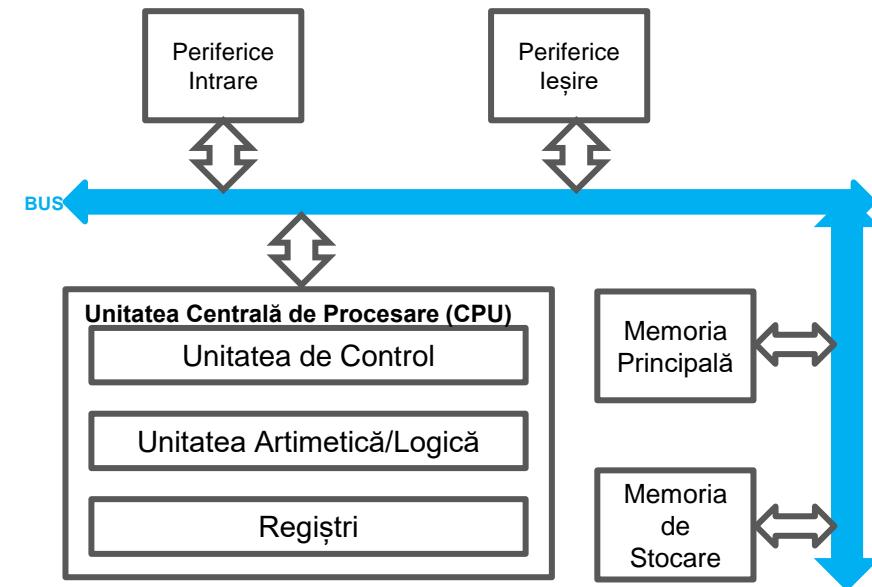
# ARHITECTURA DE BAZĂ

- Memoria de Stocare
  - conține cod și date
  - este nevolatilă
- SSD (Solid State Disks)
  - e memorie flash, rapidă
  - azi, e scumpă
  - scrierea e mult mai lentă decât citirea
- HDD (Hard Disks)
  - mecanic

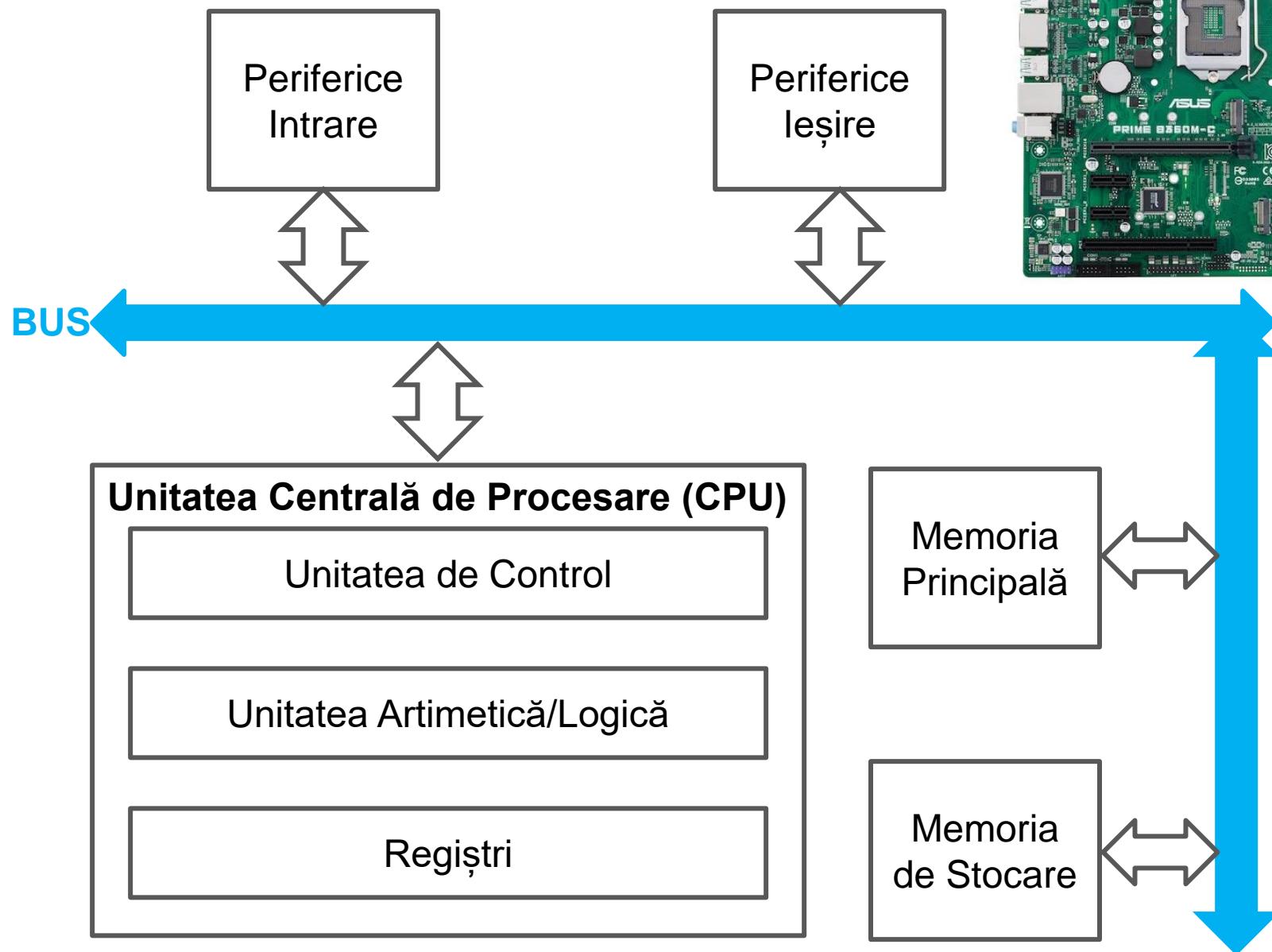


# ARHITECTURA DE BAZĂ

- **BUS**
  - **conectează CPU/memorie**
  - **proprietați**
    - **capacitatea (bandwidth)**
    - **viteza (MHz)**



# ARHITECTURA DE BAZĂ



un astfel de sistem poate executa doar *cod mașină*

# **CE AM FĂCUT ASTĂZI**

- arhitectura de bază a calculatoarelor
- **Instruction Set Architecture (ISA)**

# **DATA VIITOARE ...**

- continuăm discuția despre arhitectura de bază a calculatoarelor
- de la cod sursă la cod mașină

# LECTURĂ SUPLIMENTARĂ

- PH book
  - 2.5 Representing Instructions in the Computer
  - 4.1 – 4.4 The Processor
- Ben Eater, Designing a 7-segment hex decoder,  
<https://www.youtube.com/watch?v=7zffjsXqATg>
- Ben Eater, Using an EEPROM to replace combinational logic,  
<https://www.youtube.com/watch?v=BA12Z7gQ4P0>
- Crash Course Computer Science (o descriere grafică intuitivă, corectă):
  - How Computers Calculate - the ALU,  
<https://www.youtube.com/watch?v=1l5ZMmrOfnA&list=PLH2l6uzC4UEW0s7-KewFLBC1D0l6XRfy&index=6>
  - Registers and RAM,  
<https://www.youtube.com/watch?v=fpnE6UAfbtU&list=PLH2l6uzC4UEW0s7-KewFLBC1D0l6XRfy&index=7>
  - The Central Processing Unit (CPU),  
<https://www.youtube.com/watch?v=FZGugFqdr60&list=PLH2l6uzC4UEW0s7-KewFLBC1D0l6XRfy&index=8>
  - Instructions & Programs,  
<https://www.youtube.com/watch?v=zItgXvg6r3k&list=PLH2l6uzC4UEW0s7-KewFLBC1D0l6XRfy&index=9>



# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x07**

**DE LA COD SURSĂ LA EXECUȚIE**

Cristian Rusu

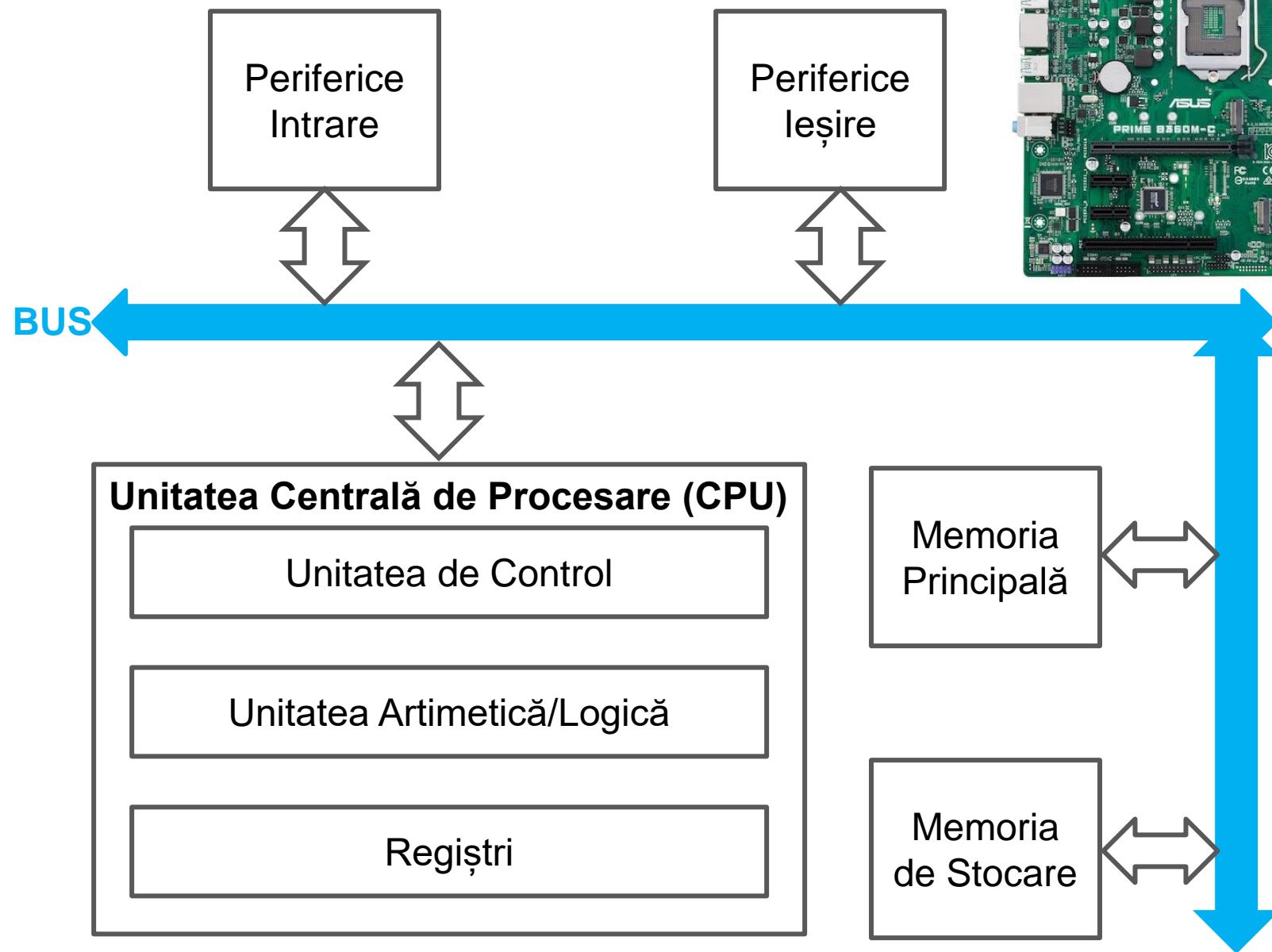
# **DATA TRECUTĂ**

- arhitectura de bază a calculatoarelor
- **Instruction Set Architecture (ISA)**

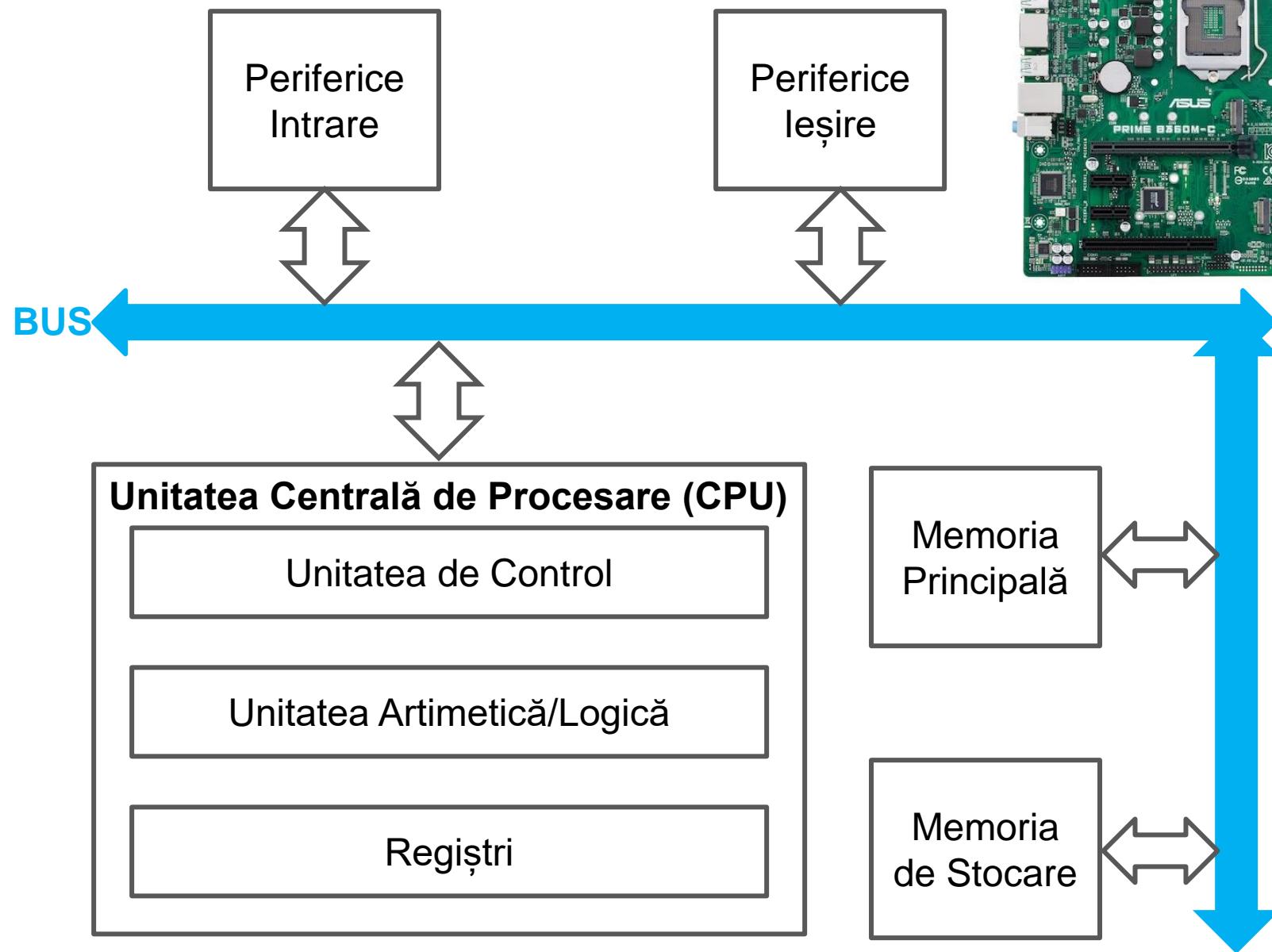
# CUPRINS

- scurt review arhitectura de bază a calculatoarelor
- de la cod sursă la cod mașină
  - software cracking
  - executarea datelor

# ARHITECTURA DE BAZĂ

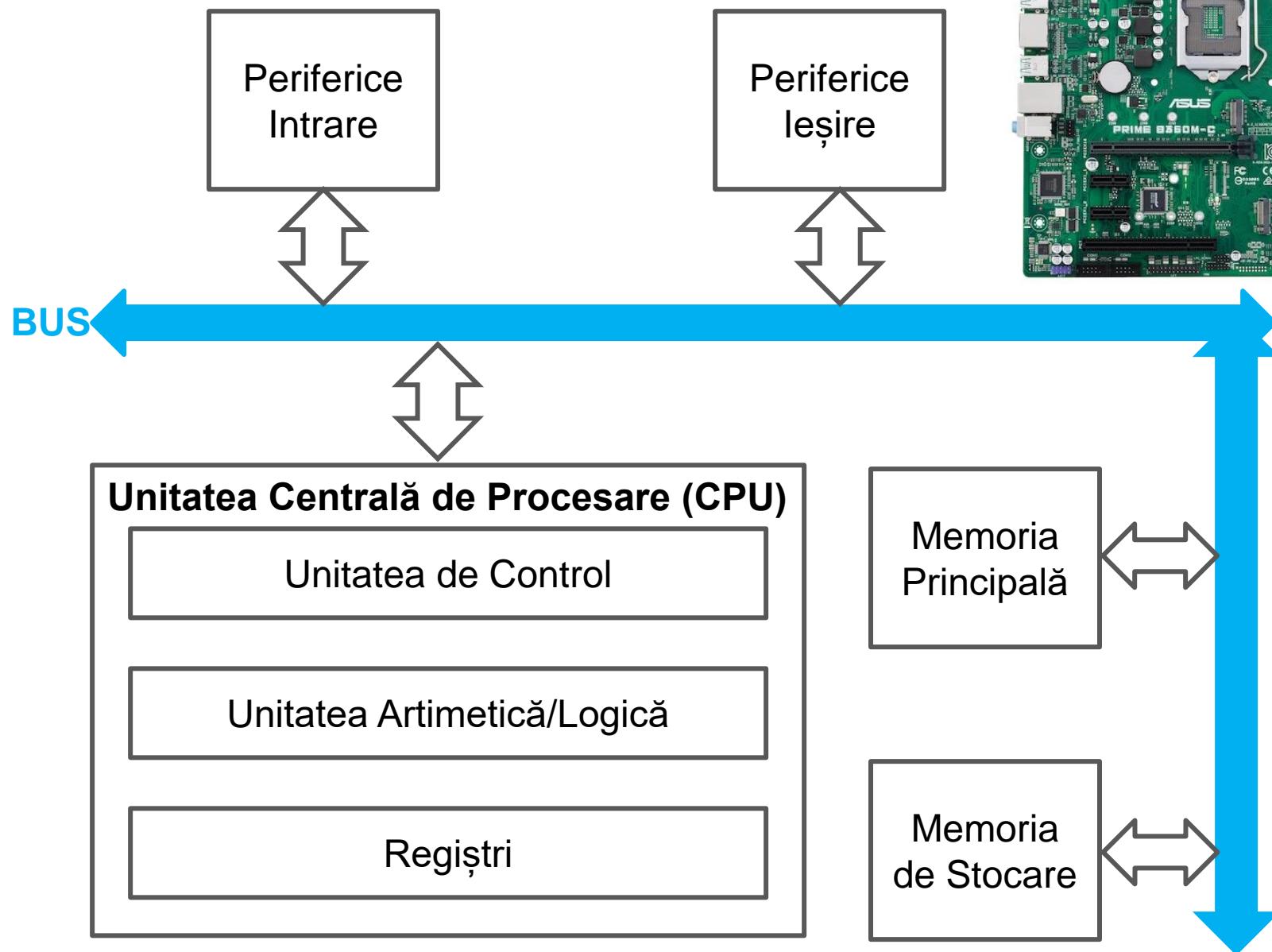


# ARHITECTURA DE BAZĂ



comunicarea cu perifericele se face de obicei prin buffer-e în memorie

# ARHITECTURA DE BAZĂ

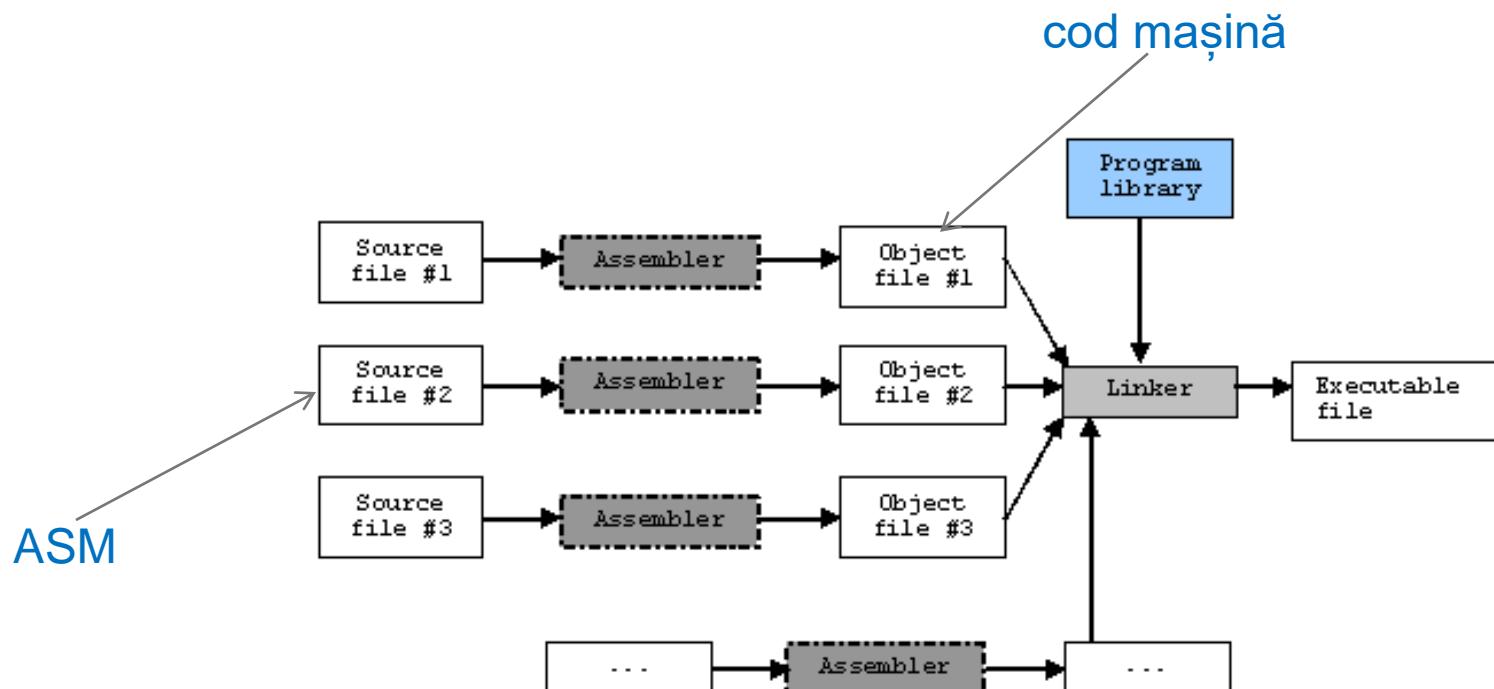


un astfel de sistem poate executa doar *cod mașină*

# DE LA COD SURSĂ LA EXECUȚIE

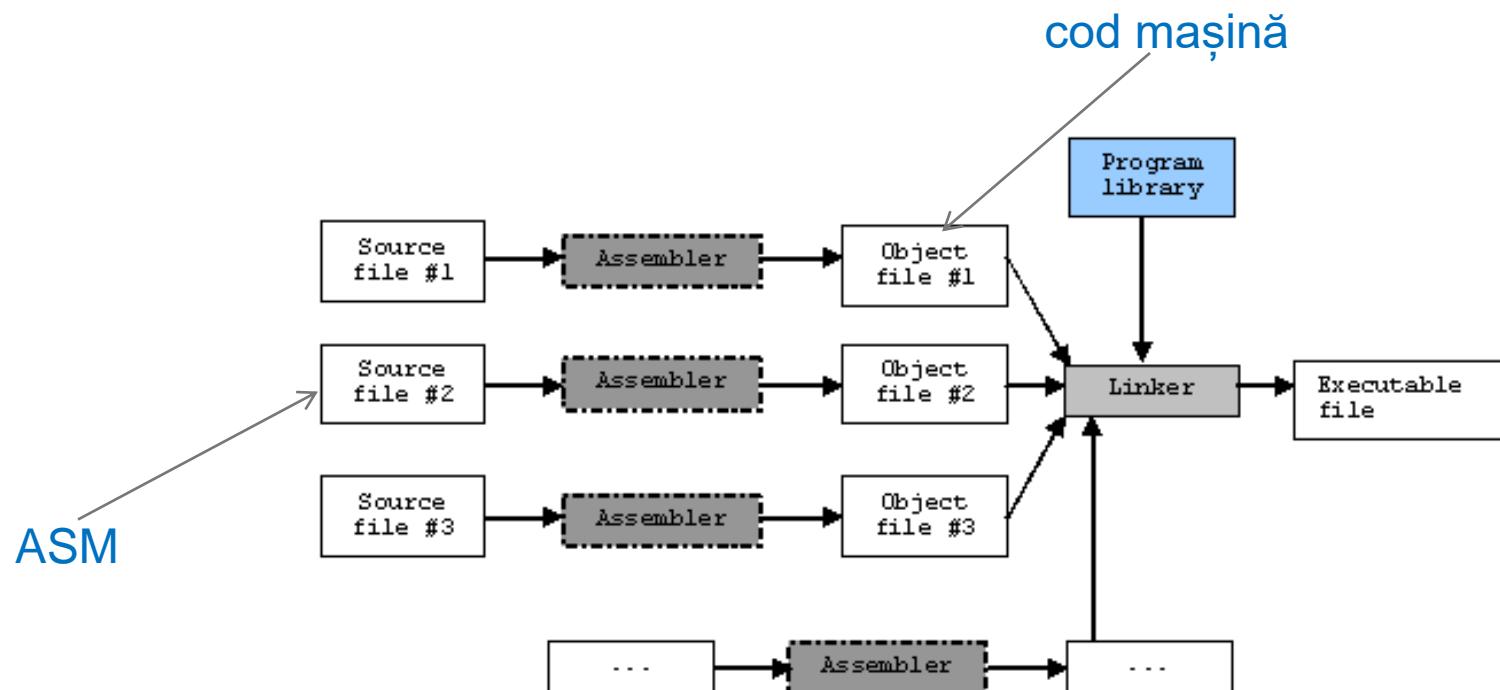
- cod mașină (machine code)

- instrucțiuni binare executate direct de CPU
- CPU poate executa doar cod mașină (orice altceva e tradus în CM)
- cum obține cod mașină?
  - din cod sursă
  - codul sursă este generic
  - codul mașină e specific pentru Assembler, CPU, OS



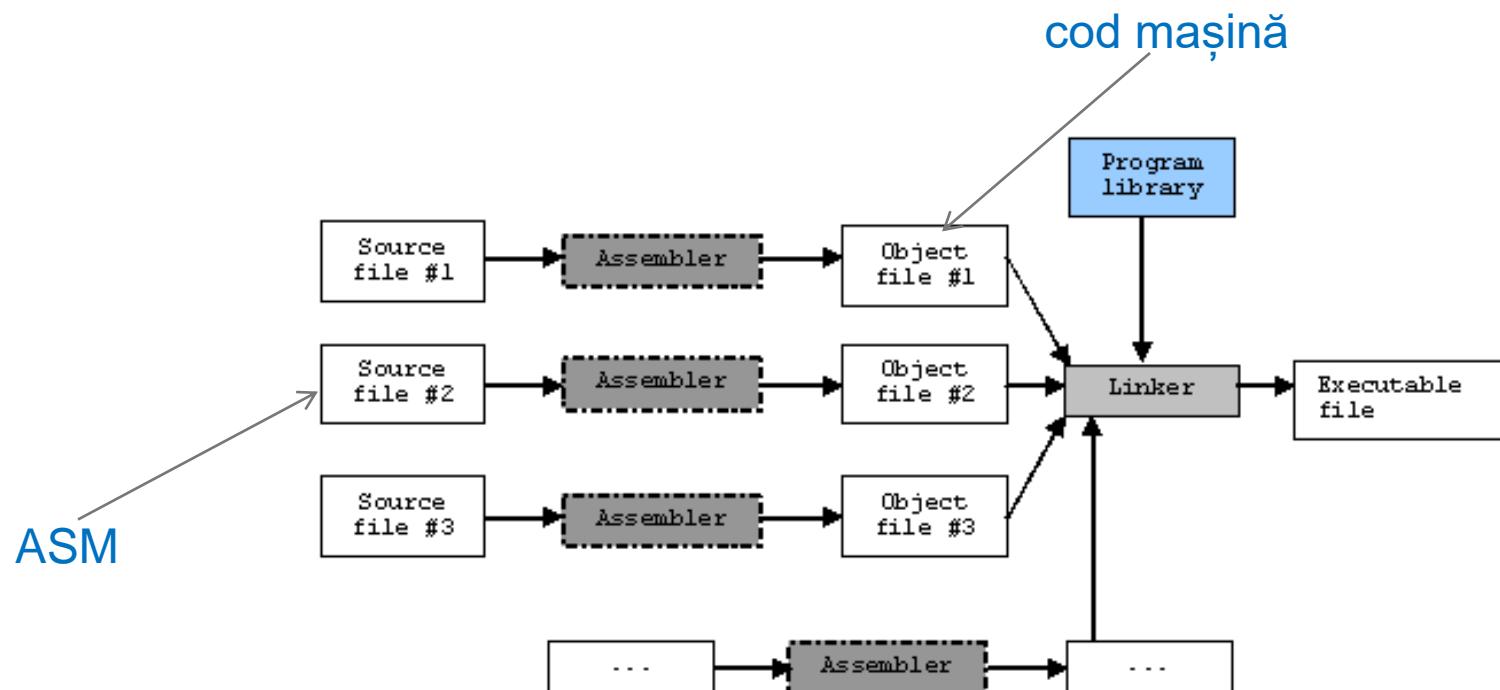
# DE LA COD SURSĂ LA EXECUȚIE

- cod mașină (machine code)
  - la laborator, primul vostru program ASM a fost:
    - as --32 program\_exit.asm -o program\_exit.o
    - ld -m elf\_i386 program\_exit.o -o program\_exit
    - ./program\_exit



# DE LA COD SURSĂ LA EXECUȚIE

- cod mașină (machine code)
  - la laborator, pentru programele ASM unde ați folosit scanf/printf:
    - as --32 matrix.asm -o matrix.o
    - gcc -m32 matrix.o -o matrix
    - ./matrix



# DE LA COD SURSĂ LA EXECUȚIE

- în general (nu doar pentru Assembly)

Source Code

```
int main()
{
    printf("Hello world!");
}
```

Assembly

```
.main PROC
    push ebp
    mov ebp, esp
    push OFFSET $SG2985
    call DWORD PTR __imp_printf
    add esp, 4
    xor eax, eax
    pop ebp
    ret 0
.main ENDP
_TEXT ENDS
END
```

Object File

```
68 00 30 40 00 FF 15 90
50 C3 B8 40 5A 00 00 66
33 C0 EB 34 80 00 3C 00
50 45 00 00 75 EA B8 00
40 00 75 DC 33 C0 83 B9
81 E8 00 40 00 0F 95 C0
15 7C 20 40 00 59 6A FF
34 20 40 00 A3 88 33 40
30 40 00 89 01 B0 00 38
89 01 E8 27 05 00 00 E8
40 00 00 75 0C 68 E4 12
59 E8 48 05 00 00 83 30
FF FF 15 44 28 40 00 59
E8 D4 04 00 00 A1 58 30
00 FF 35 54 30 40 00 A3
```

Binary File

```
68 00 30 40 00 FF 15 90
50 C3 B8 40 5A 00 00 66
33 C0 EB 34 80 00 3C 00
50 45 00 00 75 EA B8 00
40 00 75 DC 33 C0 83 B9
81 E8 00 40 00 0F 95 C0
15 7C 20 40 00 59 6A FF
34 20 40 00 A3 88 33 40
30 40 00 89 01 B0 00 38
89 01 E8 27 05 00 00 E8
40 00 00 75 0C 68 E4 12
00 68 B4 20 40 00 68 A7
59 59 85 C0 74 17 C7 45
00 00 E9 DE 00 00 00 89
33 40 00 75 18 68 A0 20
77 04 00 00 59 59 C7 05
```

Process

```
68 00 30 40 00 FF 15 90
50 C3 B8 40 5A 00 00 66
33 C0 EB 34 80 00 3C 00
50 45 00 00 75 EA B8 00
40 00 75 DC 33 C0 83 B9
81 E8 00 40 00 0F 95 C0
15 7C 20 40 00 59 6A FF
34 20 40 00 A3 88 33 40
30 40 00 89 01 B0 00 38
89 01 E8 27 05 00 00 E8
40 00 00 75 0C 68 E4 12
59 E8 48 05 00 00 83 30
FF FF 15 44 28 40 00 59
E8 D4 04 00 00 A1 58 30
00 FF 35 54 30 40 00 A3
```

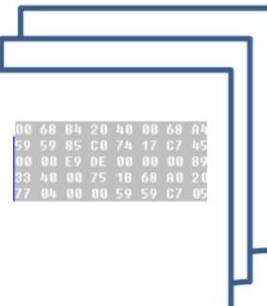
Compile

Assemble

Link

Load

Libraries



# DE LA COD SURSĂ LA EXECUȚIE

cod sursă: main.c

```
#include <stdio.h>

int main()
{
    printf("hello\n");
    return 42;
}
```

gcc -S -o main.asm main.c

cod sursă, assembly main.s

```
.LC0:
    .string "hello"
    .text
    .globl main
    .type main, @function

main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq %rsp, %rbp
    .cfi_def_cfa_register 6
    leaq .LC0(%rip), %rdi
    call puts@PLT
    movl $42, %eax
    popq %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

gcc -o main main.c

cod mașină, main (hexdump)

0000000	457f 464c 0102 0001 0000 0000 0000 0000
0000010	0003 003e 0001 0000 1060 0000 0000 0000
0000020	0040 0000 0000 0000 3978 0000 0000 0000
0000030	0000 0000 0040 0038 000d 0040 001f 001e
0000040	0006 0000 0004 0000 0040 0000 0000 0000
0000050	0040 0000 0000 0000 0040 0000 0000 0000
0000060	02d8 0000 0000 0000 02d8 0000 0000 0000

# DE LA COD SURSĂ LA EXECUȚIE

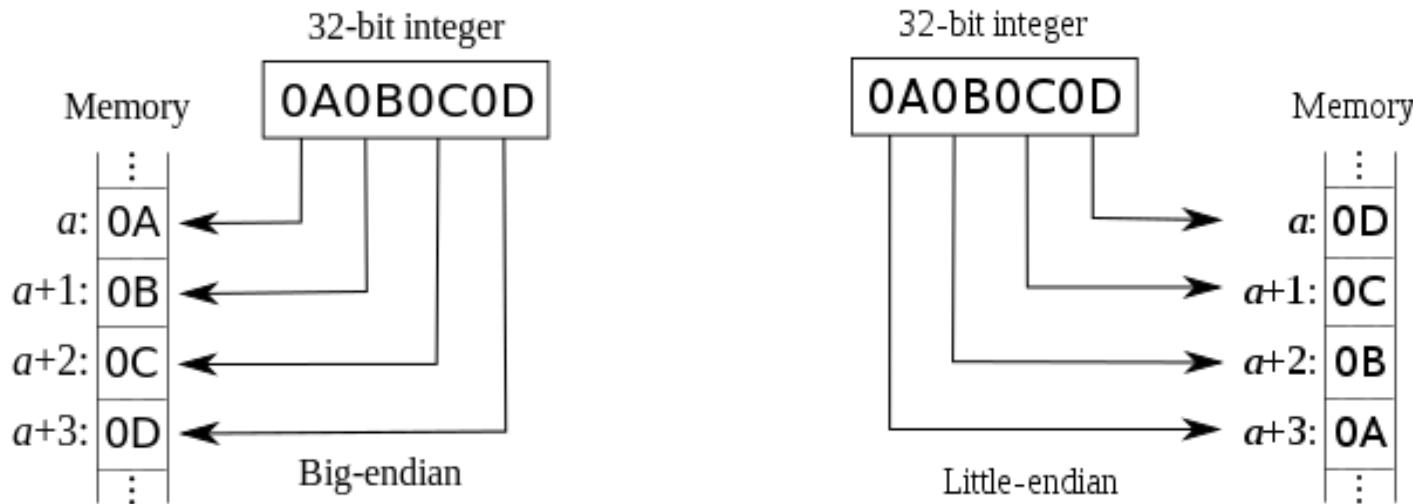
- **objdump main**

```
0000000000001149 <main>:  
1149: f3 0f 1e fa        endbr64  
114d: 55                 push    %rbp  
114e: 48 89 e5           mov     %rsp,%rbp  
1151: 48 8d 3d ac 0e 00 00  lea     0xeac(%rip),%rdi      # 2004 <_IO_stdin_used+0x4>  
1158: e8 f3 fe ff ff    callq   1050 <puts@plt>  
115d: b8 2a 00 00 00    mov     $0x2a,%eax  
1162: 5d                 pop    %rbp  
1163: c3                 retq  
1164: 66 2e 0f 1f 84 00 00  nopw   %cs:0x0(%rax,%rax,1)  
116b: 00 00 00             
116e: 66 90               xchg   %ax,%ax
```

# DE LA COD SURSĂ LA EXECUȚIE

- objdump main

```
0000000000000001149 <main>:  
1149: f3 0f 1e fa          endbr64  
114d: 55                  push    %rbp  
114e: 48 89 e5            mov     %rsp,%rbp  
1151: 48 8d 3d ac 0e 00 00 lea     0xeac(%rip),%rdi      # 2004 <_IO_stdin_used+0x4>  
1158: e8 f3 fe ff ff      callq   1050 <puts@plt>  
115d: b8 2a 00 00 00      mov     $0x2a,%eax  
1162: 5d                  pop    %rbp  
1163: c3                  retq  
1164: 66 2e 0f 1f 84 00 00 nopw   %cs:0x0(%rax,%rax,1)  
116b: 00 00 00  
116e: 66 90                xchg   %ax,%ax
```



de asemenea, observați că instrucțiunile nu sunt codate cu aceeași lungime

# ARHITECTURA SETULUI DE INSTRUCȚIUNI

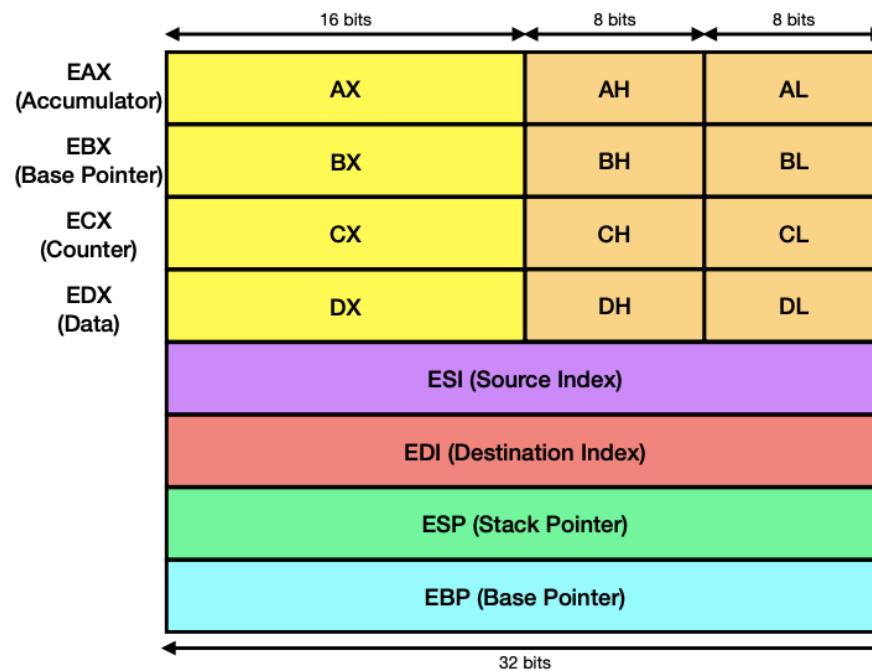
- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - registri
    - instrucțiuni
    - tipuri de date
    - metode de adresare a memoriei

# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- Instruction Set Arhitecture (ISA)

- structura sintactică și semantică a limbajului Assembly
  - regiștri
  - instrucțiuni
  - tipuri de date
  - metode de adresare a memoriei

4 bits = 1 nibble  
8 bits = 1 byte  
16 bits = 1 word  
32 bits = 1 dword  
64 bits = 1 qword



## FLAGS

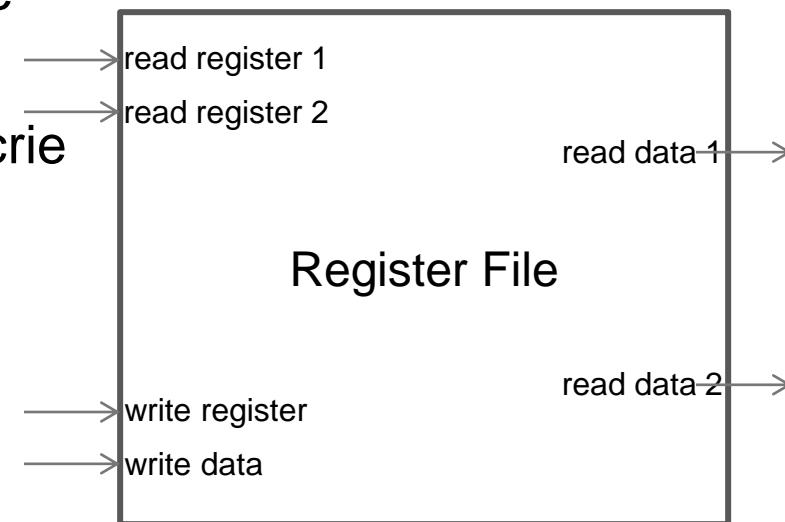
**Instruction Pointer (IP)**: următoarea instrucțiune care trebuie executată

**Stack Pointer (ESP)**: adresa stivei

**YMM** (pentru AVX) / **XMM** (pentru SSE): regiștri pentru operații pe vectori

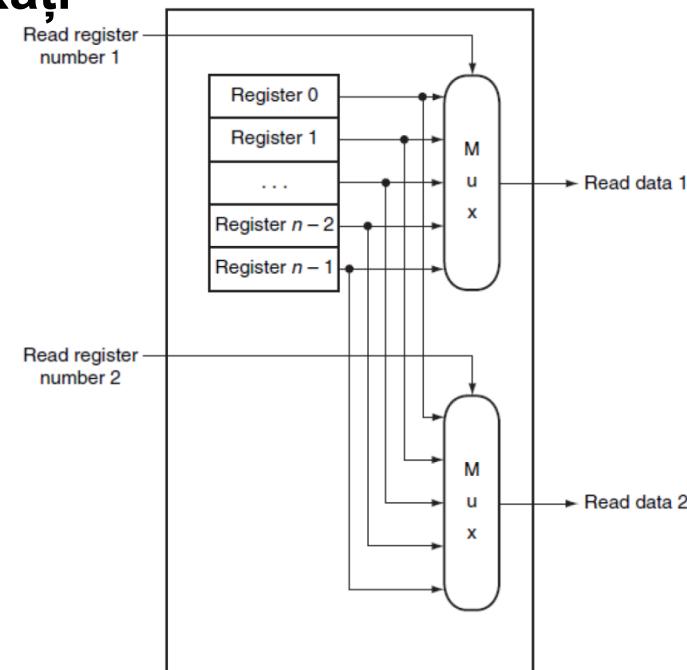
# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - regiștri
    - instrucțiuni
    - tipuri de date
    - metode de adresare a memoriei
- **În general, regiștrii sunt grupați și indexați**
  - read register 1 / 2: indecsii de citire
  - read data 1 / 2: datele citite
  - write register: indexul în care se scrie
  - write data: datele care se scriu



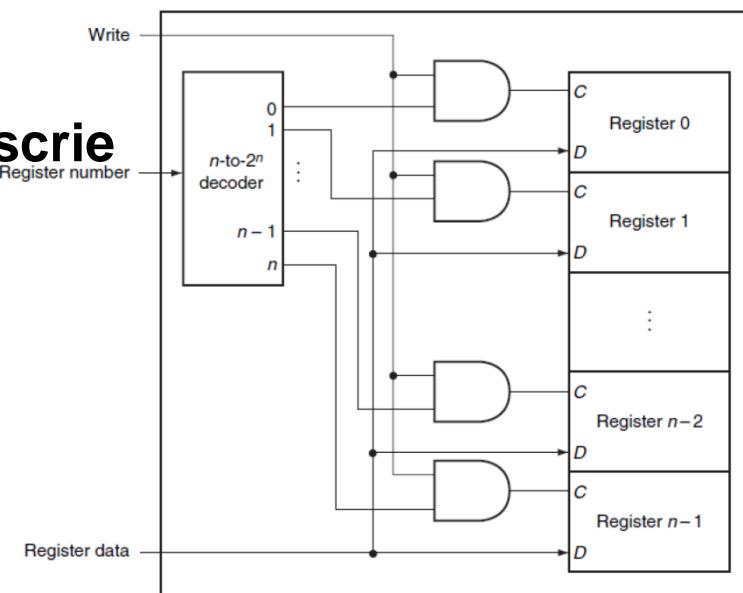
# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - regiștri
    - instrucțiuni
    - tipuri de date
    - metode de adresare a memoriei
- **În general, regiștrii sunt grupați și indexați**
  - **read register 1 / 2: indecșii de citire**
  - **read data 1 / 2: datele citite**
  - **write register: indexul în care se scrie**
  - **write data: datele care se scriu**



# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- Instruction Set Arhitecture (ISA)
  - structura sintactică și semantică a limbajului Assembly
    - regiștri
    - instrucțiuni
    - tipuri de date
    - metode de adresare a memoriei
- În general, regiștrii sunt grupați și indexați
  - read register 1 / 2: indecsii de citire
  - read data 1 / 2: datele citite
  - **write register: indexul în care se scrie**
  - **write data: datele care se scriu**



# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - regiștri
    - **instructiuni**
    - tipuri de date
    - metode de adresare a memoriei
  - <opcode> <listă operanzi>
    - add op1, op2 ( $op2 \leftarrow op2 + op1$ )
    - **Categorii de instructiuni**
      - **transferul datelor**: mov, cmov, movq, movs, movz, push, pop
      - **aritmetică și logică**: add, sub, mul, imul, div, idiv, sal, sar, shl, shr, and, or, not, xor, test, cmp
      - **controlul programului**: call, ret, j\*

# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- Instruction Set Arhitecture (ISA)
  - structura sintactică și semantică a limbajului Assembly
    - registri
    - instrucțiuni
    - **tipuri de date**
    - metode de adresare a memoriei

C declaration	Intel data type	GAS suffix	x86-64 Size (Bytes)
char	Byte	b	1
short	Word	w	2
int	Double word	l	4
unsigned	Double word	l	4
long int	Quad word	q	8
unsigned long	Quad word	q	8
char *	Quad word	q	8
float	Single precision	s	4
double	Double precision	d	8
long double	Extended precision	t	16

# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - registri
    - instrucțiuni
    - tipuri de date
    - metode de adresare a memoriei
  - **adresare imediată:**
    - imediat: mov \$172, %rdi
    - cu registru: mov %rcx, %rdi
    - cu memorie: mov 0x172, %rdi
  - **adresare indirectă**
    - indirect prin registru: mov (%rax), %rdi
    - indirect indexat: mov 172(%rax), %rdi
    - indirect bazat pe IP: mov 172(%rip), %rdi
  - **cazul cel mai general:** mov 172(%rdi, %rdx, 8), %rax
    - Base + Index\*Scale + Displacement
    - îl aveți explicit detaliat în suportul de laborator

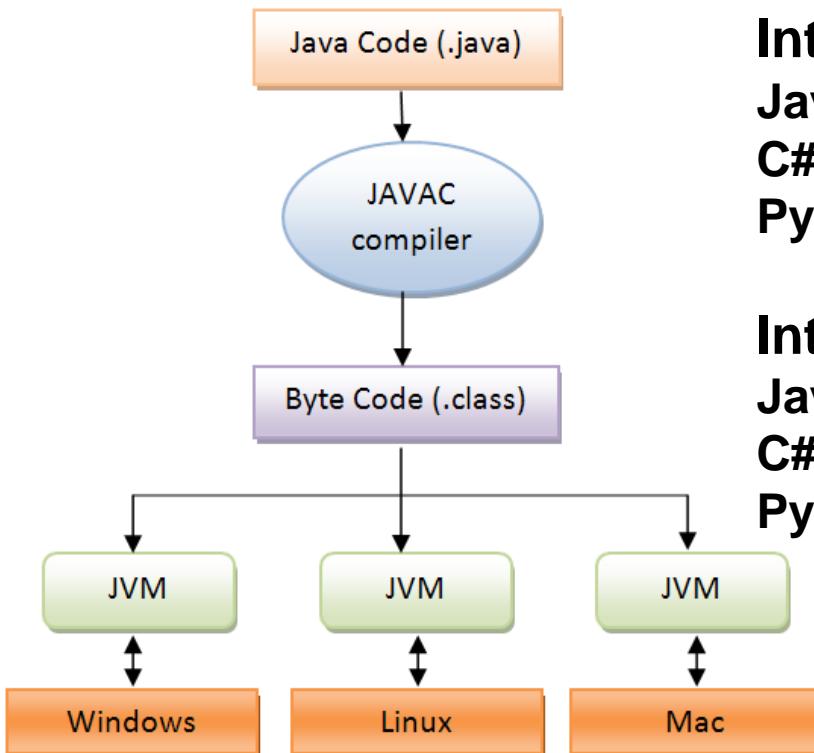
# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- câteva exemple în Assembly

- mov
- push
- call
- cmp
- add
- pop
- lea
- test
- je
- xor
- jmp
- jne
- ret
- inc/sub

# DE LA COD SURSĂ LA EXECUȚIE

- **excepție de la regulă**
  - bytecode (cod interpretat): instrucțiunile sunt executate de un interpretor care apoi le trimită la CPU



## Interpreted code:

**Java:** java byte-code

**C#:** Common Intermediate Language (CIL)

**Python:** python byte-code (fișiere .pyc)

## Interpreter:

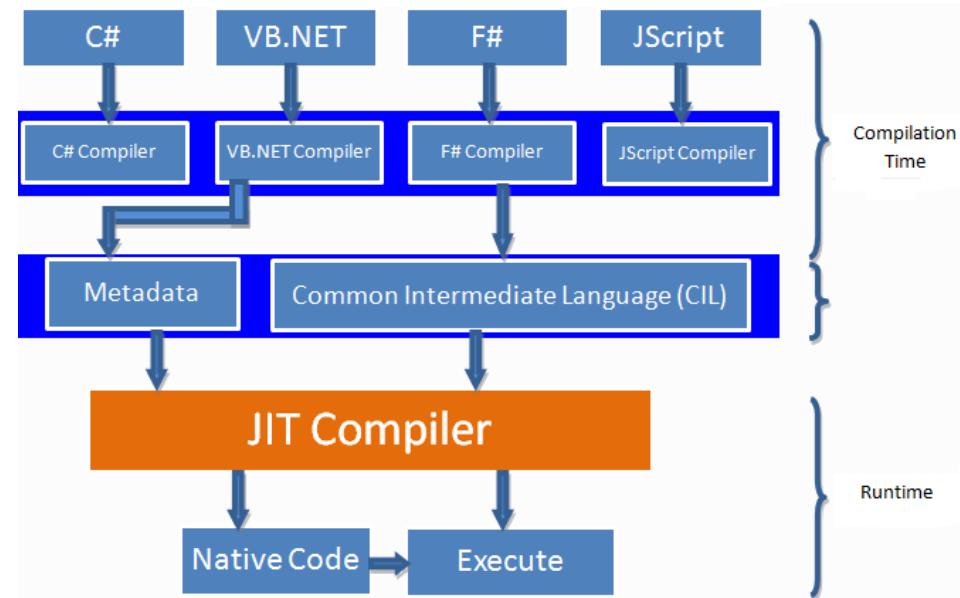
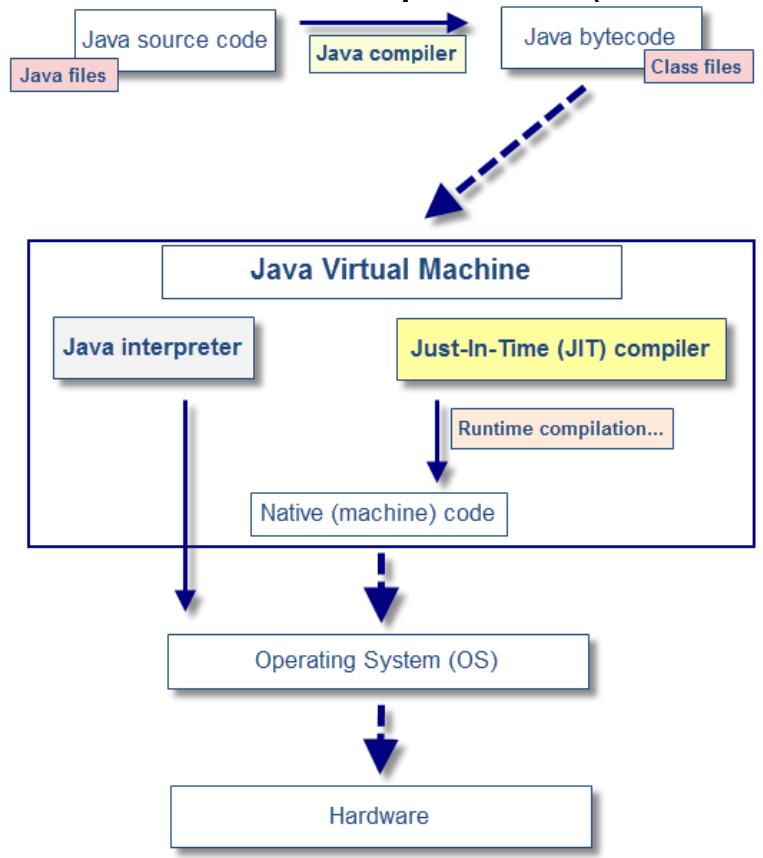
**Java:** Java VM

**C#:** Common Language Runtime (CLR) în .NET

**Python:** python Virtual Machine

# DE LA COD SURSĂ LA EXECUȚIE

- **excepție de la regulă**
  - bytecode (cod interpretat): instrucțiunile sunt executate de un interpretor care apoi le trimită la CPU
  - totul e lent pentru că mai este un pas de procesare
  - JIT compilation (Just-In-Time compilation) ajută



# UN EXEMPLU

- următorul program simplu verifică o cheie de licență

```
#include <string.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    if(argc==2) {
        printf("Checking License: %s\n", argv[1]);
        if(strcmp(argv[1], "AAAA-Z10N-42-OK")==0) {
            printf("Access Granted!\n");
        } else {
            printf("WRONG!\n");
        }
    } else {
        printf("Usage: <key>\n");
    }
    return 0;
}
```

# UN EXEMPLU

- **gdb checklicense**

verifică dacă ceva este egal cu 2

call la strcmp  
apoi jne

din nou call la puts  
avem asta în cod?

```
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
0x0000000000000740 <+0>:    push   rbp
0x0000000000000741 <+1>:    mov    rbp,rs
0x0000000000000744 <+4>:    sub    rs,0x10
0x0000000000000748 <+8>:    mov    DWORD PTR [rbp-0x4],edi
0x000000000000074b <+11>:   mov    QWORD PTR [rbp-0x10],rsi
0x000000000000074f <+15>:   cmp    DWORD PTR [rbp-0x4],0x2
0x0000000000000753 <+19>:   jne    0x7ae <main+110>
0x0000000000000755 <+21>:   mov    rax,QWORD PTR [rbp-0x10]
0x0000000000000759 <+25>:   add    rax,0x8
0x000000000000075d <+29>:   mov    rax,QWORD PTR [rax]
0x0000000000000760 <+32>:   mov    rsi,rax
0x0000000000000763 <+35>:   lea    rdi,[rip+0xea]      # 0x854
0x000000000000076a <+42>:   mov    eax,0x0
0x000000000000076f <+47>:   call   0x5e0 <printf@plt>
0x0000000000000774 <+52>:   mov    rax,QWORD PTR [rbp-0x10]
0x0000000000000778 <+56>:   add    rax,0x8
0x000000000000077c <+60>:   mov    rax,QWORD PTR [rax]
0x000000000000077f <+63>:   lea    rsi,[rip+0xec]      # 0x872
0x0000000000000786 <+70>:   mov    rdi,rax
0x0000000000000789 <+73>:   call   0x5f0 <strcmp@plt>
0x000000000000078e <+78>:   test   eax,eax
0x0000000000000790 <+80>:   jne    0x7a0 <main+96>
0x0000000000000792 <+82>:   lea    rdi,[rip+0xe2]      # 0x87b
0x0000000000000799 <+89>:   call   0x5d0 <puts@plt>
0x000000000000079e <+94>:   jmp    0x7ba <main+122>
0x00000000000007a0 <+96>:   lea    rdi,[rip+0xe4]      # 0x88b
0x00000000000007a7 <+103>:  call   0x5d0 <puts@plt>
0x00000000000007ac <+108>:  jmp    0x7ba <main+122>
0x00000000000007ae <+110>:  lea    rdi,[rip+0xe4]      # 0x899
0x00000000000007b5 <+117>:  call   0x5d0 <puts@plt>
0x00000000000007ba <+122>:  mov    eax,0x0
0x00000000000007bf <+127>:  leave 
0x00000000000007c0 <+128>:  ret

End of assembler dump.
(gdb) █
```

# UN EXEMPLU

- gdb checklicense

```
#include <string.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    if(argc==2) {
        printf("Checking License: %s\n", argv[1]);
        if(strcmp(argv[1], "AAAA-Z10N-42-OK")==0) {
            printf("Access Granted!\n");
        } else {
            printf("WRONG!\n");
        }
    } else {
        printf("Usage: <key>\n");
    }
    return 0;
}
```

```
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
0x0000000000000740 <+0>: push   rbp
0x0000000000000741 <+1>: mov    rbp,rsi
0x0000000000000744 <+4>: sub    rsp,0x10
0x0000000000000748 <+8>: mov    DWORD PTR [rbp-0x4],edi
0x000000000000074b <+11>: mov    QWORD PTR [rbp-0x10],rsi
0x000000000000074f <+15>: cmp    DWORD PTR [rbp-0x4],0x2
0x0000000000000753 <+19>: ine    0x7ae <main+110>
0x0000000000000755 <+21>: mov    rax,QWORD PTR [rbp-0x10]
0x0000000000000759 <+25>: add    rax,0x8
0x000000000000075d <+29>: mov    rax,QWORD PTR [rax]
0x0000000000000760 <+32>: mov    rsi,rax
0x0000000000000763 <+35>: lea    rdi,[rip+0xea]      # 0x854
0x000000000000076a <+42>: mov    eax,0x0
0x000000000000076f <+47>: call   0x5e0 <printf@plt>
0x0000000000000774 <+52>: mov    rax,QWORD PTR [rbp-0x10]
0x0000000000000778 <+56>: add    rax,0x8
0x000000000000077c <+60>: mov    rax,QWORD PTR [rax]
0x000000000000077f <+63>: lea    rsi,[rip+0xec]      # 0x872
0x0000000000000786 <+70>: mov    rdi,rax
0x0000000000000789 <+73>: call   0x5f0 <strcmp@plt>
0x000000000000078e <+78>: test   eax,eax
0x0000000000000790 <+80>: jne    0x7a0 <main+96>
0x0000000000000792 <+82>: lea    rdi,[rip+0xe2]      # 0x87b
0x0000000000000799 <+89>: call   0x5d0 <puts@plt>
0x000000000000079e <+94>: jmp    0x7ba <main+122>
0x00000000000007a0 <+96>: lea    rdi,[rip+0xe4]      # 0x88b
0x00000000000007a7 <+103>: call   0x5d0 <puts@plt>
0x00000000000007ac <+108>: jmp    0x7ba <main+122>
0x00000000000007ae <+110>: lea    rdi,[rip+0xe4]      # 0x899
0x00000000000007b5 <+117>: call   0x5d0 <puts@plt>
0x00000000000007ba <+122>: mov    eax,0x0
0x00000000000007bf <+127>: leave 
0x00000000000007c0 <+128>: ret

End of assembler dump.
(gdb) █
```

# UN EXEMPLU

- **informațiile executabilului**
  - file checklicense
- **hex viewer**
  - hexdump -C checklicense
- **hex editor**
  - hexeditor checklicense
- **scoate toate string-urile din fișier**
  - strings checklicense
- **dump al obiectelor din fișier**
  - objdump -x checklicense
- **analiză binară avansată**
  - radare2 (r2)
  - ghidra

# UN EXEMPLU

- objdump -d checklicense

```
0000000000000740 <main>:  
740: 55                      push   %rbp  
741: 48 89 e5                mov    %rsp,%rbp  
744: 48 83 ec 10              sub    $0x10,%rsp  
748: 89 7d fc                mov    %edi,-0x4(%rbp)  
74b: 48 89 75 f0              mov    %rsi,-0x10(%rbp)  
74f: 83 7d fc 02              cmpl   $0x2,-0x4(%rbp)  
753: 75 59                  jne    7ae <main+0x6e>  
755: 48 8b 45 f0              mov    -0x10(%rbp),%rax  
759: 48 83 c0 08              add    $0x8,%rax  
75d: 48 8b 00                mov    (%rax),%rax  
760: 48 89 c6                mov    %rax,%rsi  
763: 48 8d 3d ea 00 00 00      lea    0xea(%rip),%rdi      # 854 <_IO_stdin_used+0x4>  
76a: b8 00 00 00 00            mov    $0x0,%eax  
76f: e8 6c fe ff ff          callq 5e0 <printf@plt>  
774: 48 8b 45 f0              mov    -0x10(%rbp),%rax  
778: 48 83 c0 08              add    $0x8,%rax  
77c: 48 8b 00                mov    (%rax),%rax  
77f: 48 8d 35 ec 00 00 00      lea    0xec(%rip),%rsi      # 872 <_IO_stdin_used+0x22>  
786: 48 89 c7                mov    %rax,%rdi  
789: e8 62 fe ff ff          callq 5f0 <strcmp@plt>  
78e: 85 c0                  test   %eax,%eax  
790: 75 0e                  jne    7a0 <main+0x60>  
792: 48 8d 3d e2 00 00 00      lea    0xe2(%rip),%rdi      # 87b <_IO_stdin_used+0x2b>  
799: e8 32 fe ff ff          callq 5d0 <puts@plt>  
79e: eb 1a                  jmp    7ba <main+0x7a>  
7a0: 48 8d 3d e4 00 00 00      lea    0xe4(%rip),%rdi      # 88b <_IO_stdin_used+0x3b>  
7a7: e8 24 fe ff ff          callq 5d0 <puts@plt>  
7ac: eb 0c                  jmp    7ba <main+0x7a>  
7ae: 48 8d 3d e4 00 00 00      lea    0xe4(%rip),%rdi      # 899 <_IO_stdin_used+0x49>  
7b5: e8 16 fe ff ff          callq 5d0 <puts@plt>  
7ba: b8 00 00 00 00            mov    $0x0,%eax  
7bf: c9                      leaveq  
7c0: c3                      retq  
7c1: 66 2e 0f 1f 84 00 00      nopw   %cs:0x0(%rax,%rax,1)  
7c8: 00 00 00  
7cb: 0f 1f 44 00 00            nopl   0x0(%rax,%rax,1)
```

# UN EXEMPLU

- hexeditor checklicense

The screenshot shows a hex editor interface with two panes. The left pane displays a memory dump in hex and ASCII formats. The right pane shows the corresponding ASCII characters. A vertical scroll bar is visible between the two panes.

Address	Hex Value	ASCII
00000790	75 0E 48 8D 3D E2 00 00 00 00 E8 32 FE FF FF EB 0C 48 8D	u.H.=.....2..... H.=.....\$.....H.
000007A0	48 8D 3D E4 00 00 00 E8 16 FE FF FF B8 00 00 00 00 C9	=..... .f.....D..
000007B0	3D E4 00 00 00 E8 16 FE FF FF B8 00 00 00 00 00 C9	AWAVA..AUATL.%.. .UH...SI..I.
000007C0	C3 66 2E 0F 1F 84 00 00 00 00 00 0F 1F 44 00 00	.L).H...H..... .H..t 1.....
000007D0	41 57 41 56 41 89 FF 41 55 41 54 4C 8D 25 F6 05	L..L..D..A..H.. .H9.u.H...[]A\A]
000007E0	20 00 55 48 8D 2D F6 05 20 00 53 49 89 F6 49 89	A^A..f..... ....H..H.....
000007F0	D5 4C 29 E5 48 83 EC 08 48 C1 FD 03 E8 9F FD FF	....Checking the license: %s ...
00000800	FF 48 85 ED 74 20 31 DB 0F 1F 84 00 00 00 00 00 00	..ABCDEFGH.Acce s granted!.Acce s denied.Usage:
00000810	4C 89 EA 4C 89 F6 44 89 FF 41 FF 14 DC 48 83 C3	<key>.....;<... .....X..
00000820	01 48 39 DD 75 EA 48 83 C4 08 5B 5D 41 5C 41 5D	....h..X..
00000830	41 5E 41 5F C3 90 66 2E 0F 1F 84 00 00 00 00 00 00	....(.....
00000840	F3 C3 00 00 48 83 EC 08 48 83 C4 08 C3 00 00 00	0.....
00000850	01 00 02 00 43 68 65 63 6B 69 6E 67 20 74 68 65	.zR..x
00000860	20 6C 69 63 65 6E 73 65 3A 20 25 73 20 2E 2E 2E	.....+.....
00000870	0A 00 41 42 43 44 45 46 47 48 00 41 63 63 65 73	.zR..x
00000880	73 20 67 72 61 6E 74 65 64 21 00 41 63 63 65 73	\$.....@.. .F..J..w..?;*
00000890	73 20 64 65 6E 69 65 64 00 55 73 61 67 65 3A 20	*3\$".....D
000008A0	3C 6B 65 79 3E 00 00 00 01 1B 03 3B 3C 00 00 00	.....\.....
000008B0	06 00 00 00 18 FD FF FF 88 00 00 00 58 FD FF FF	.A....C... .....
000008C0	B0 00 00 00 68 FD FF FF 58 00 00 00 98 FE FF FF	D.... ...8...e...
000008D0	C8 00 00 00 28 FF FF FF E8 00 00 00 98 FF FF FF	.B....B....E..
000008E0	30 01 00 00 00 00 00 00 14 00 00 00 00 00 00 00	B.(..H..0..H.8..M
000008F0	01 7A 52 00 01 78 10 01 1B 0C 07 08 90 01 07 10	.@r.8A.0A.(B..B.
00000900	14 00 00 00 1C 00 00 00 08 FD FF FF 2B 00 00 00	.B..B.
00000910	00 00 00 00 00 00 00 00 14 00 00 00 00 00 00 00	.....
00000920	01 7A 52 00 01 78 10 01 1B 0C 07 08 90 01 00 00	.....
00000930	24 00 00 00 1C 00 00 00 88 FC FF FF 40 00 00 00	.....
00000940	00 0E 10 46 0E 18 4A 0F 0B 77 08 80 00 3F 1A 3B	.....
00000950	2A 33 24 22 00 00 00 00 14 00 00 00 44 00 00 00	.....
00000960	A0 FC FF FF 08 00 00 00 00 00 00 00 00 00 00 00	.....
00000970	1C 00 00 00 5C 00 00 00 C8 FD FF FF 81 00 00 00	.....
00000980	00 41 0E 10 86 02 43 0D 06 02 7C 0C 07 08 00 00	.....
00000990	44 00 00 00 7C 00 00 00 38 FE FF FF 65 00 00 00	.....
000009A0	00 42 0E 10 8F 02 42 0E 18 8E 03 45 0E 20 8D 04	.....
000009B0	42 0E 28 8C 05 48 0E 30 86 06 48 0E 38 83 07 4D	.....
000009C0	0E 40 72 0E 38 41 0E 30 41 0E 28 42 0E 20 42 0E	.....
000009D0	18 42 0E 10 42 0E 08 00 14 00 00 00 C4 00 00 00	.....
000009E0	60 FE FF FF 02 00 00 00 00 00 00 00 00 00 00 00	.....
000009F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000AA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000A10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000A20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000A30	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000A40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

schimbăm JNE?  
care este noul OPCODE  
pentru noua instrucțiune?

# UN EXEMPLU

- hexeditor checklicense

The screenshot shows a hex editor interface with two panes. The left pane displays a memory dump in hex and ASCII formats. The right pane shows the corresponding ASCII text. A vertical scroll bar is visible between the panes.

Address	Hex	ASCII
00000790	75 0E 48 8D 3D E2 00 00 00 00 E8 32 FE FF FF EB 1A	u.H.=.....2.....
000007A0	48 8D 3D E4 00 00 00 E8 24 FE FF FF EB 0C 48 8D	H.=.....\$.....H.
000007B0	3D E4 00 00 00 E8 16 FE FF FF B8 00 00 00 00 C9	=.....
000007C0	C3 66 2E 0F 1F 84 00 00 00 00 00 0F 1F 44 00 00	.f.....D..
000007D0	41 57 41 56 41 89 FF 41 55 41 54 4C 8D 25 F6 05	AWAVA..AUATL.%..
000007E0	20 00 55 48 8D 2D F6 05 20 00 53 49 89 F6 49 89	.UH.-...SI..I..
000007F0	D5 4C 29 E5 48 83 EC 08 48 C1 FD 03 E8 9F FD FF	.L).H...H.....
00000800	FF 48 85 ED 74 20 31 DB 0F 1F 84 00 00 00 00 00 00	.H..t 1.....
00000810	4C 89 EA 4C 89 F6 44 89 FF 41 FF 14 DC 48 83 C3	L..L..D..A..H..
00000820	01 48 39 DD 75 EA 48 83 C4 08 5B 5D 41 5C 41 5D	.H9.u.H...[]A\A]
00000830	41 5E 41 5F C3 90 66 2E 0F 1F 84 00 00 00 00 00 00	A^A ..f.....
00000840	F3 C3 00 00 48 83 EC 08 48 83 C4 08 C3 00 00 00	....H..H.....
00000850	01 00 02 00 43 68 65 63 6B 69 6E 67 20 74 68 65	....Checking the
00000860	20 6C 69 63 65 6E 73 65 3A 20 25 73 20 2E 2E 2E	license: %s ...
00000870	0A 00 41 42 43 44 45 46 47 48 00 41 63 63 65 73	..ABCDEFGH.Acce
00000880	73 20 67 72 61 6E 74 65 64 21 00 41 63 63 65 73	s granted!.Acce
00000890	73 20 64 65 6E 69 65 64 00 55 73 61 67 65 3A 20	s denied.Usage:
000008A0	3C 6B 65 79 3E 00 00 00 01 1B 03 3B 3C 00 00 00	<key>.....;<...
000008B0	06 00 00 00 18 FD FF FF 88 00 00 00 58 FD FF FF	.....X..
000008C0	B0 00 00 00 68 FD FF FF 58 00 00 00 98 FE FF FF	....h..X..
000008D0	C8 00 00 00 28 FF FF FF E8 00 00 00 98 FF FF FF	....(.....
000008E0	30 01 00 00 00 00 00 00 14 00 00 00 00 00 00 00 00	0.....
000008F0	01 7A 52 00 01 78 10 01 1B 0C 07 08 90 01 07 10	.zR..x
00000900	14 00 00 00 1C 00 00 00 08 FD FF FF 2B 00 00 00	.....+.....
00000910	00 00 00 00 00 00 00 00 14 00 00 00 00 00 00 00 00	.....
00000920	01 7A 52 00 01 78 10 01 1B 0C 07 08 90 01 00 00	.zR..x
00000930	24 00 00 00 1C 00 00 00 88 FC FF FF 40 00 00 00	\$.....@..
00000940	00 0E 10 46 0E 18 4A 0F 0B 77 08 80 00 3F 1A 3B	...F..J..w..?;*
00000950	2A 33 24 22 00 00 00 00 14 00 00 00 44 00 00 00	*3\$".....D..
00000960	A0 FC FF FF 08 00 00 00 00 00 00 00 00 00 00 00	.....
00000970	1C 00 00 00 5C 00 00 00 C8 FD FF FF 81 00 00 00	....\.....
00000980	00 41 0E 10 86 02 43 0D 06 02 7C 0C 07 08 00 00	.A....C... .....
00000990	44 00 00 00 7C 00 00 00 38 FE FF FF 65 00 00 00	D.... ...8...e...
000009A0	00 42 0E 10 8F 02 42 0E 18 8E 03 45 0E 20 8D 04	.B....B....E..
000009B0	42 0E 28 8C 05 48 0E 30 86 06 48 0E 38 83 07 4D	B.(..H..0..H.8..M
000009C0	0E 40 72 0E 38 41 0E 30 41 0E 28 42 0E 20 42 0E	.@r.8A.0A.(B..B.
000009D0	18 42 0E 10 42 0E 08 00 14 00 00 00 C4 00 00 00	.B..B..
000009E0	60 FE FF FF 02 00 00 00 00 00 00 00 00 00 00 00	.....
000009F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000AA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000A10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000A20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000A30	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000A40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

At the bottom, there is a menu bar with options: G Help, ^C Exit (No Save), ^T goTo Offset, ^X Exit and Save, ^W Search, ^U Undo, ^L Redraw, ^E.

schimbați voi și alte lucruri:

- parola
- textul afișat

# UN EXEMPLU

- ce am făcut?
  - am modificat, permanent, fișierul binar
  - cum ne putem da seama că un fișier a fost modificat?

## Kali Linux Downloads

### Download Kali Linux Images

We generate fresh Kali Linux image files every few months, which we make available for download. This page provides the links to download Kali Linux in its latest official release. For a release history, check our Kali Linux Releases page. Please note: You can find unofficial, untested weekly releases at <http://cdimage.kali.org/kali-weekly/>. Downloads are **rate limited to 5 concurrent connections**.

Image Name	Torrent	Version	Size	SHA256Sum
Kali Linux 64-Bit (Installer)	<a href="#">Torrent</a>	2020.4	4.1G	50492d761e400c2b5e22c8f253dd6f75c27e4bc84e33c2eff272476a0588fb02
Kali Linux 64-Bit (Live)	<a href="#">Torrent</a>	2020.4	3.3G	4d764a2ba67f41495c17247184d24b7f9ac9a7c57415bbbed663402aec78952b

# EXECUȚIA DATELOR

- fie următorul program foarte simplu (shellcode.c)

```
#include      <stdio.h>
#include      <stdlib.h>
#include      <unistd.h>
#include      <string.h>
#include      <errno.h>

int main()
{
    int e;
    char *argv[] = { "/bin/ls", "-l", NULL };

    e = execve("/bin/ls", argv, NULL);
    if (e == -1)
        fprintf(stderr, "Error: %s\n", strerror(errno));
    return 0;
}
```

# EXECUȚIA DATELOR

- același program în Assembly

```
.text
.globl _start

_start:
    xor %eax, %eax
    push %eax
    push $0x68732f2f
    push $0x6e69622f
    mov %esp, %ebx
    push %eax
    push %ebx
    mov %esp, %ecx
    mov $0xb, %al
    int $0x80
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

```
root@kali:~# objdump -d shellcode
shellcode:      file format elf32-i386
```

Disassembly of section .text:

08048054 <_start>:		
8048054:           31 c0	xor	%eax, %eax
8048056:           50	push	%eax
8048057:           68 2f 2f 73 68	push	\$0x68732f2f
804805c:           68 2f 62 69 6e	push	\$0x6e69622f
8048061:           89 e3	mov	%esp, %ebx
8048063:           50	push	%eax
8048064:           53	push	%ebx
8048065:           89 e1	mov	%esp, %ecx
8048067:           b0 0b	mov	\$0xb, %al
8048069:           cd 80	int	\$0x80
804806b:           b8 01 00 00 00	mov	\$0x1, %eax
8048070:           bb 00 00 00 00	mov	\$0x0, %ebx
8048075:           cd 80	int	\$0x80

# EXECUȚIA DATELOR

- un program echivalent

```
#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
    fprintf(stdout,"Length: %d\n",strlen(shellcode));
    (*(void(*)()) shellcode)();
    return 0;
}
```

ce se întamplă aici?  
afişați shellcode-ul pe ecran

# EXECUȚIA DATELOR

- un program echivalent

```
#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
    fprintf(stdout,"Length: %d\n",strlen(shellcode));
    (*(void(*)()) shellcode)();
    return 0;
}
```

aceste programe nu mai pot rula pe sisteme de operare moderne

- Data Execution Prevention (DEP) e activ

# CE AM FĂCUT ASTĂZI

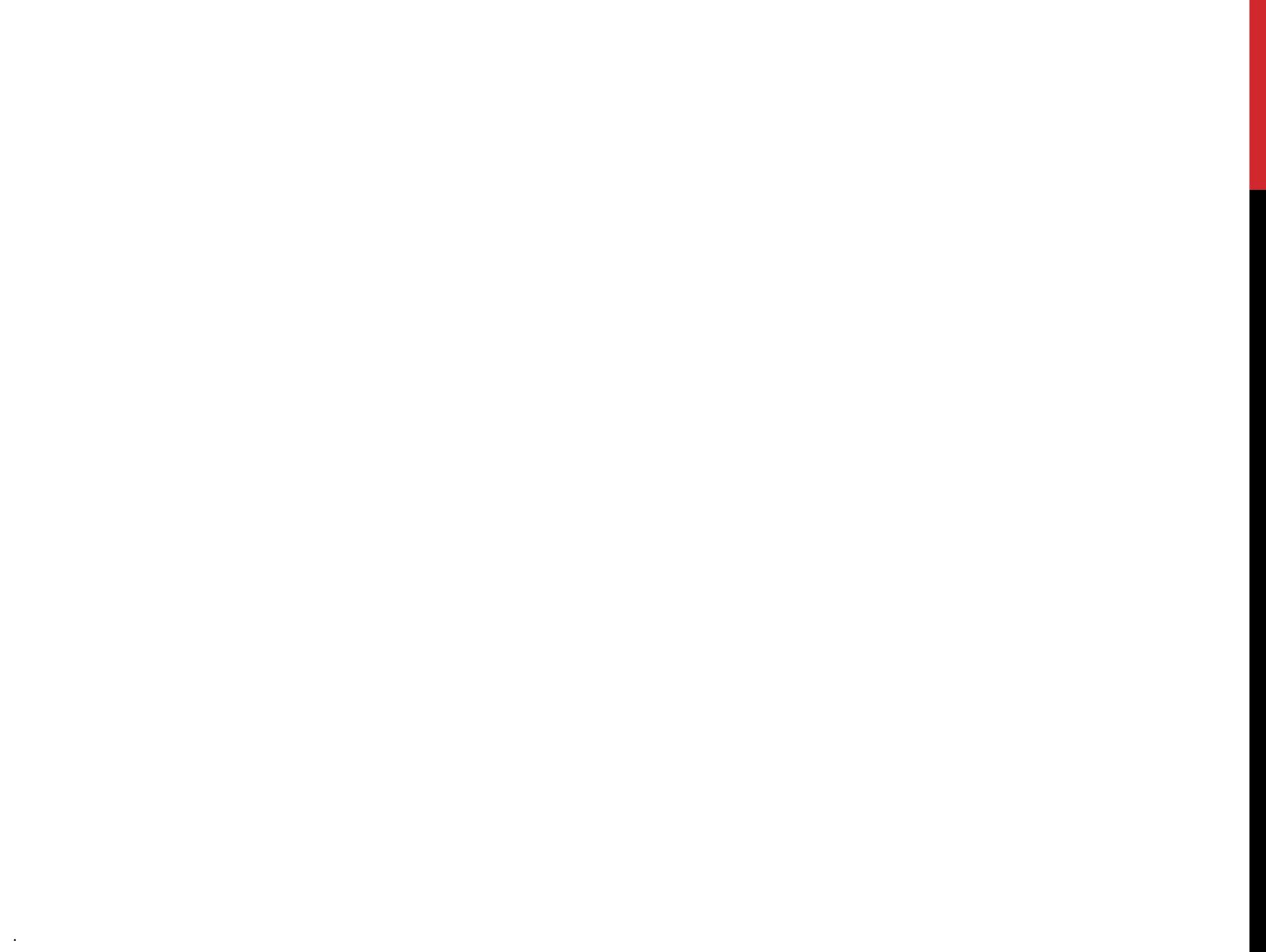
- Instruction Set Architecture (ISA)
- de la cod sursă la cod mașină
- un exemplu simplu de *software cracking* și *shellcode execution*

# **DATA VIITOARE ...**

- acoperim concepte mai complexe care aduc performanță sporită
  - pipelining
  - branch prediction
  - out of order execution
- sisteme multi-procesor
- performanța calculatoarelor

# LECTURĂ SUPLIMENTARĂ

- PH book
  - 1.3 Below Your Program
  - 1.4 Under the Covers
  - 2.15 Advanced Material: Compiling C and Interpreting Java



# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x08**

**PIPELINING, BRANCH PREDICTION, OUT OF  
ORDER EXECUTION**

Cristian Rusu

# DATA TRECUTĂ

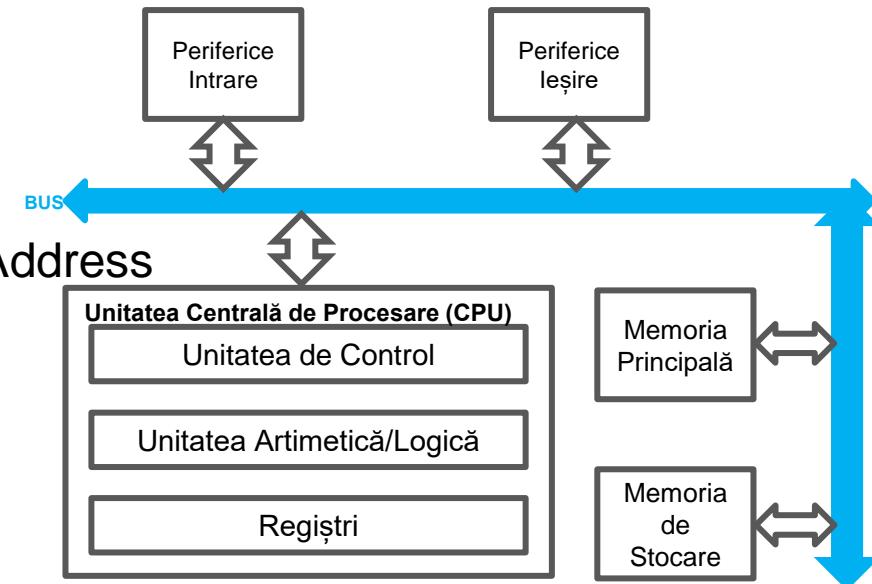
- **Instruction Set Architecture (ISA)**
- **de la cod sursă la cod mașină**
  - sper că v-ați amintit demo-ul de la Cursul 0x00 unde am executat jocul Doom în browser
- **un exemplu simplu de *software cracking* și *shellcode execution***
- **dacă sunteți pasionați de securitate (binary exploitation):**
  - C, ASM (stiva, call-ul procedurilor)
  - cursul de SO (anul II, semestrul I)
  - J. Erickson, Hacking: The Art of Exploitation
  - exerciții:
    - protostar
    - Capture The Flag (CTF)

# CUPRINS

- **trei mecanisme pentru execuție cu viteză sporită**
  - pipelining (conductă de date)
  - branch prediction (predicția salturilor)
  - out of order execution (execuția în ordine arbitrară)

# ARHITECTURA DE BAZĂ

- CPU execută instrucțiuni:
  - IF – Instruction Fetch
  - ID – Instruction Decode
    - COA – Calculate Operand Address
    - FO – Fetch Operand
    - FOs – Fetch Operands
  - EX – Execution
  - MEM – Memory Access
  - WB – Write Back



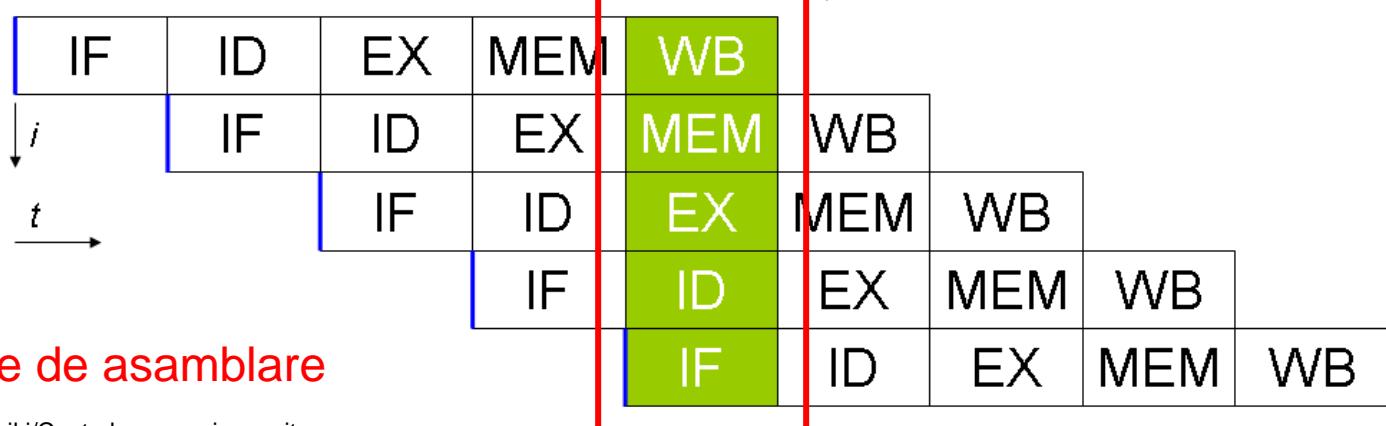
# PIPELINING

- este un tip de paralelism la nivel de instrucțiune (Instruction Level Parallelism - ILP)
  - asta pentru că e paralelism diferit față de “task/process level parallelism” (thread-uri și procese)
- de la :



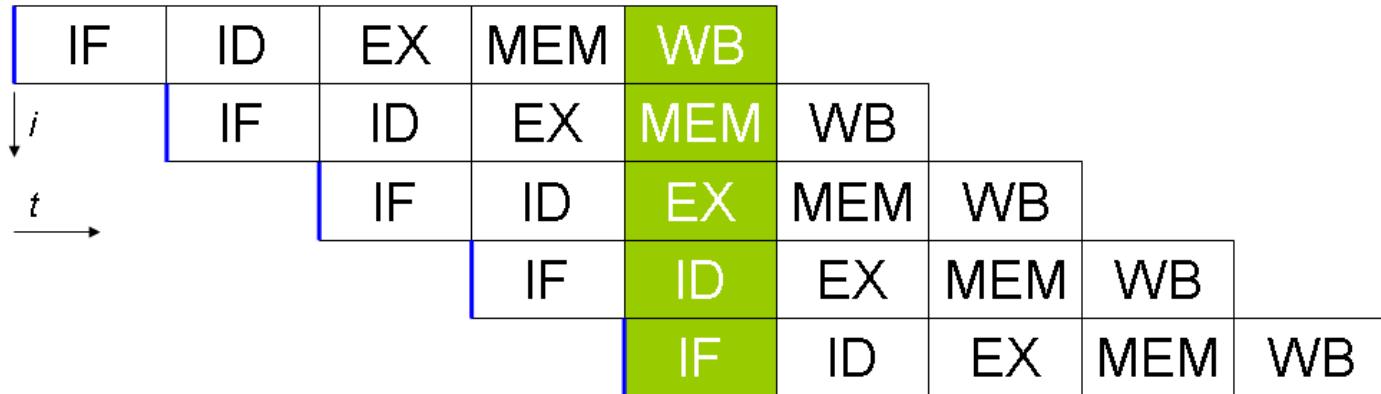
- vrem să ajungem, ideal, la:

eficiență maximă



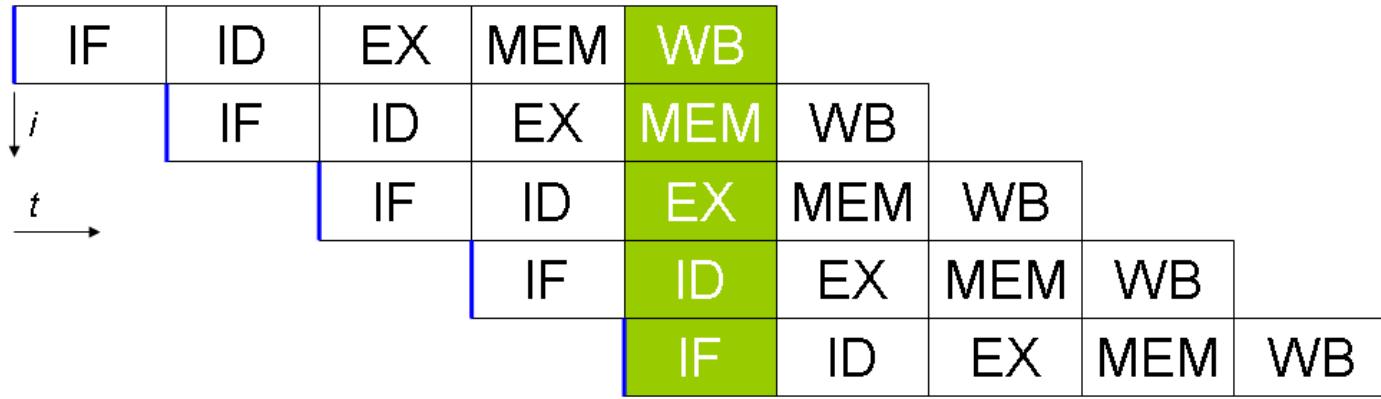
ideea: o linie de asamblare

# PIPELINING



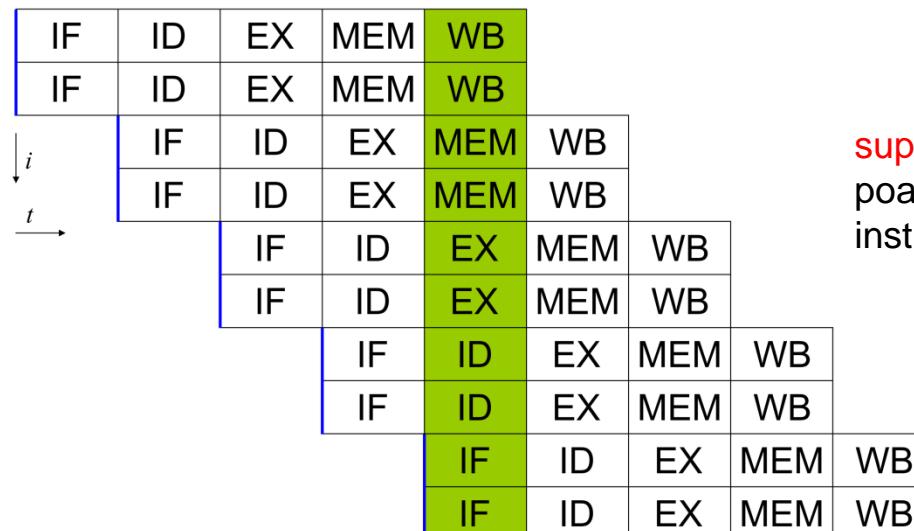
- imaginea arată bine, din păcate nu putem face aşa ceva mereu
  - pipeline stalls (întârzieri în conducta de date)
  - aceste evenimente se numesc hazards (erori)
    - **structural hazards**: o unitate de calcul este deja utilizată
      - două instrucțiuni încearcă să acceseze aceeași unitate
    - **data hazards**: datele nu sunt pregătite pentru utilizare
      - o instrucțiune depinde de rezultatul unei instrucțiuni precedente
    - **control hazards**: nu știm următoarea instrucțiune
      - din cauza unor instrucțiuni de jump nu știm instrucțiunea următoare

# PIPELINING: STRUCTURAL HAZARDS



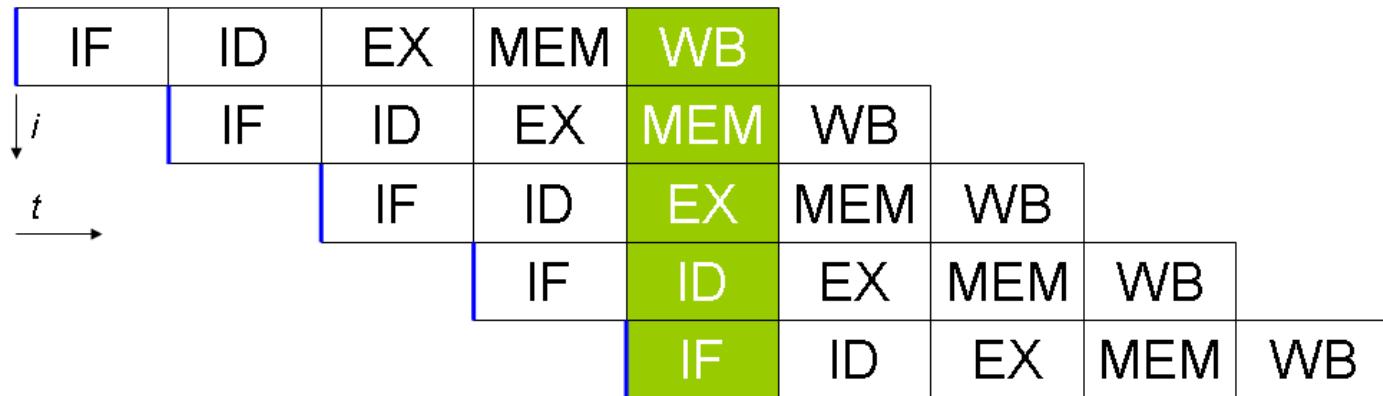
- **pipeline stalls (întârzieri în conducta de date)**

- structural hazards
  - două instrucțiuni încearcă să acceseze aceeași unitate



superscalar, hardware dublat,  
poate termina mai mult de o  
instrucție într-un ciclu

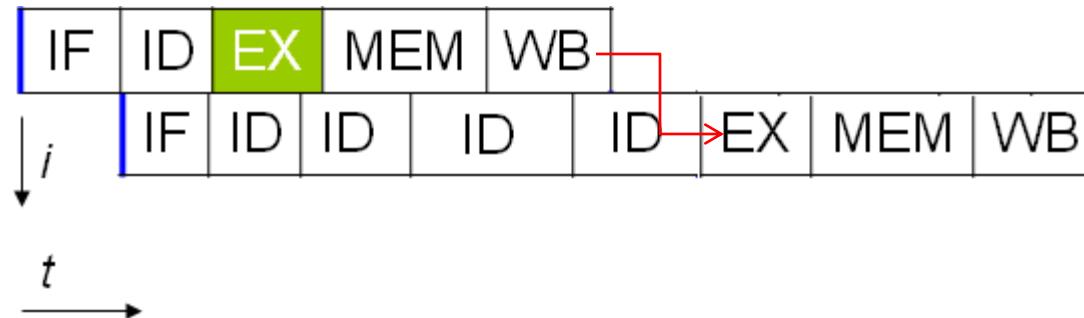
# PIPELINING: DATA HAZARDS



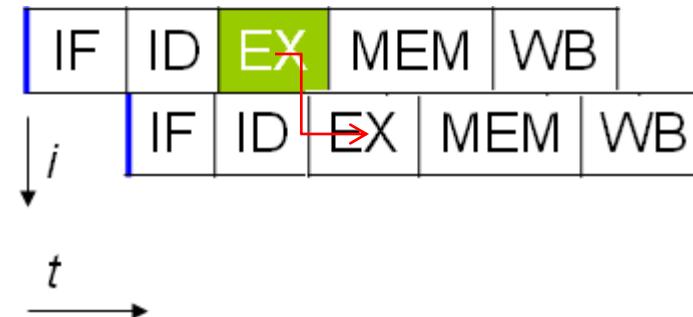
- **pipeline stalls (întârzieri în conducta de date)**
  - data hazards
    - o instrucțiune depinde de rezultatul unei instrucțiuni anterioare
    - **True Dependence (Read After Write – RAW)**
      - add %ebx, **%eax**
      - sub **%eax**, %ecx
    - **Anti-dependence (Write After Read – WAR)**
      - add **%ebx**, %eax
      - sub %ecx, **%ebx**
    - **Output Dependence (Write After Write – WAW)**
      - mov \$0x10, **%eax**
      - mov \$0x01, **%eax**

# PIPELINING: DATA HAZARDS

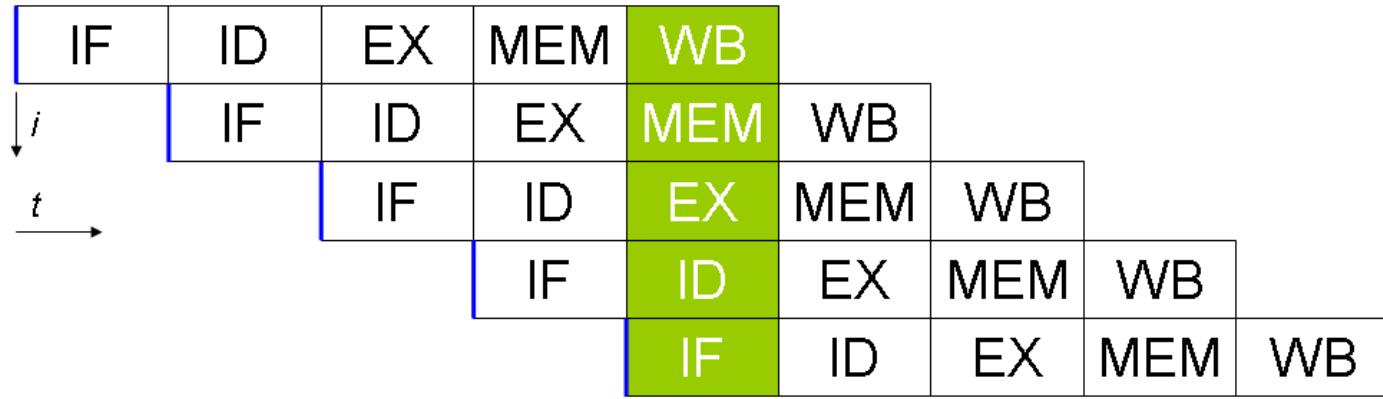
- pipeline stalls (întârzieri în conducta de date)
  - data hazards
    - True Dependence (Read After Write – RAW)
      - add %ebx, %eax
      - sub %eax, %ecx



- posibilă soluție *bypassing*



# PIPELINING: DATA HAZARDS

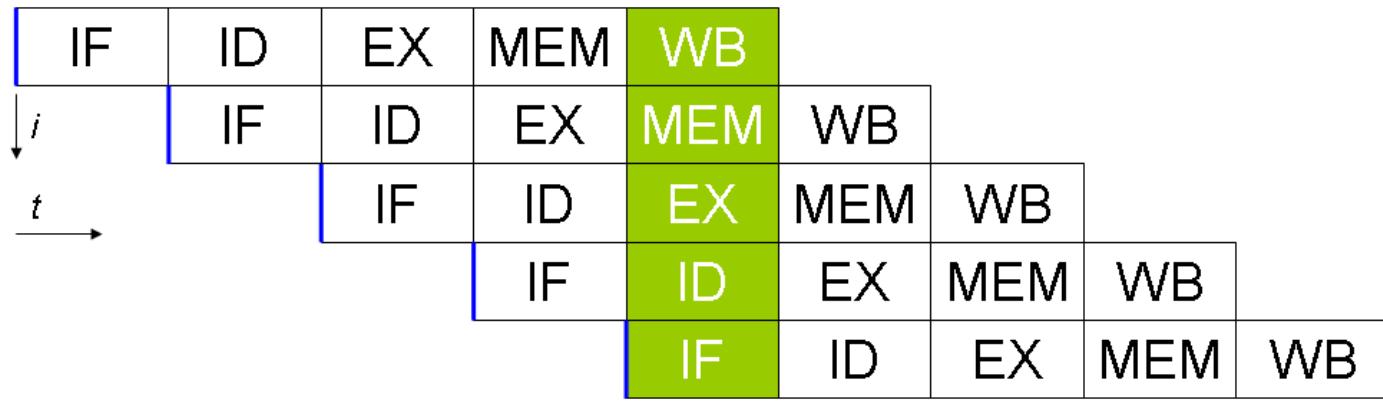


- suplimentar, situația e complicată de faptul că instrucțiuni diferite au nevoie de un număr diferit de cicli de CPU

operația	instrucțiuni	# cicli
operații întregi/biți	add, sub, and, or, xor, sar, sal, lea, etc.	1
înmulțirea întregilor	mul, imul	3
împărțirea întregilor	div, idiv	deinde (20–80)
adunare floating point	addss, addsd	3
înmulțire floating point	mulss, mulsd	5
împărțire floating point	divss, divsd	deinde (20–80)
fused-multiply-add floating point	vfmass, vfmasd	5

ce facem în aceste situații?

# PIPELINING: DATA HAZARDS



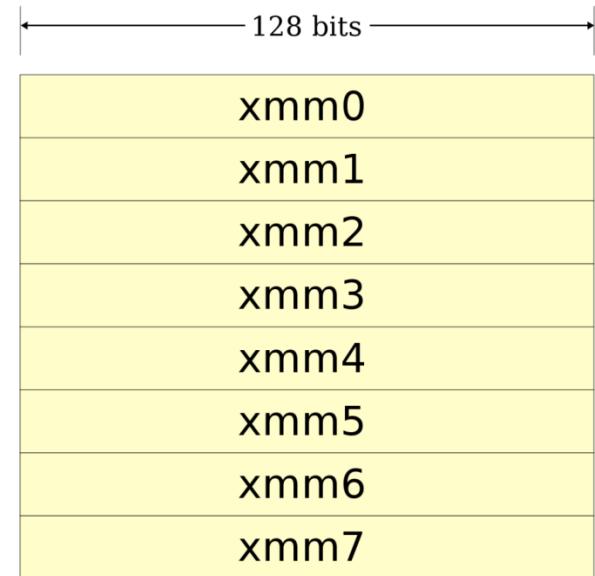
- suplimentar, situația e complicată de faptul că instrucțiuni diferite au nevoie de un număr diferit de cicli de CPU

operația	instrucțiuni	# cicli
operații întregi/biți	add, sub, and, or, xor, sar, sal, lea, etc.	1
înmulțirea întregilor	mul, imul	3
împărțirea întregilor	div, idiv	deinde (20–80)
adunare floating point	addss, addsd	3
înmulțire floating point	mulss, mulsd	5
împărțire floating point	divss, divsd	deinde (20–80)
fused-multiply-add floating point	vfmass, vfmasd	5

avem registri speciali

# PIPELINING: DATA HAZARDS

- pentru instrucțiuni “dificile” avem registri speciali
  - xmm?
  - xmm8-xmm15 există doar pe x64
  - original, registri suportau
    - 4 înmulțiri pe 32-bit FP
  - datorită SSE2, operații cu
    - două numere 64-bit FP
    - două numere întregi pe 64-bit
    - 4 numere pe 32 de biți
    - 8 numere pe 16 biți
    - 16 numere pe 8 biți
  - Streaming SIMD Extensions
    - exemplu:  $\text{result.x} = \text{v1.x} + \text{v2.x}$ ,  $\text{result.y} = \text{v1.y} + \text{v2.y}$ ,  $\text{result.z} = \text{v1.z} + \text{v2.z}$ ,  $\text{result.w} = \text{v1.w} + \text{v2.w}$
    - devine:
      - `movaps v1, xmm0 # xmm0 = v1.w | v1.z | v1.y | v1.x`
      - `movaps v2, xmm1 # xmm1 = v2.w | v2.z | v2.y | v2.x`
      - `addps xmm1, xmm0 # xmm0 = v1.w + v2.w | v1.z + v2.z | v1.y + v2.y | v1.x + v2.x`



# PIPELINING: DATA HAZARDS

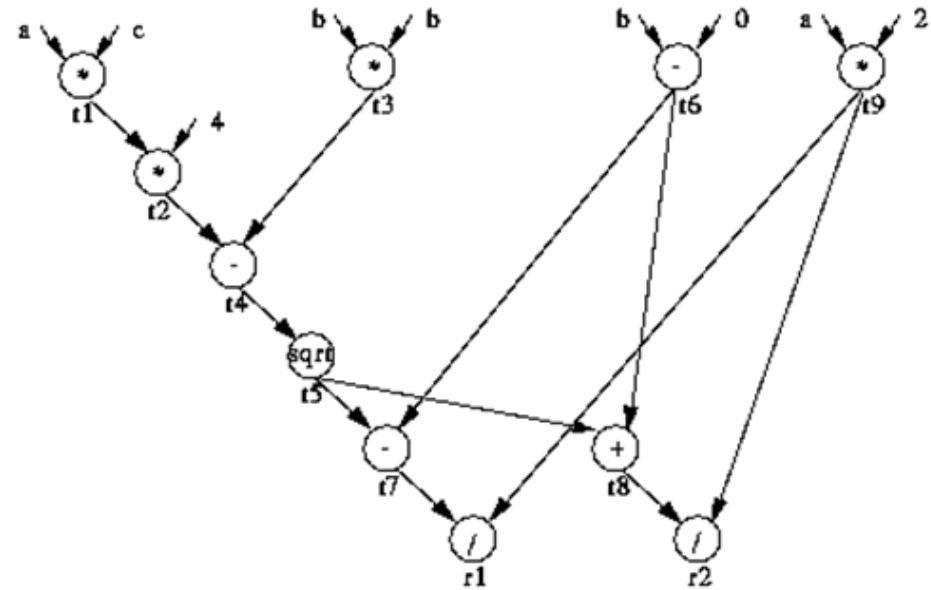
- redenumirea regiștrilor (*register renaming*)
- considerăm codul (unde folosim doar registrul R1)
  - $R1 = \text{data}[1024]$
  - $R1 = R1 + 2$
  - $\text{data}[1032] = R1$
  - $R1 = m[2048]$
  - $R1 = R1 + 4$
  - $m[2056] = R1$
- ce observați? ce putem face aici?

# PIPELINING: DATA HAZARDS

- redenumirea regiștrilor (*register renaming*)
- considerăm codul (unde folosim doar registrul R1)
  - $R1 = \text{data}[1024]$
  - $R1 = R1 + 2$
  - $\text{data}[1032] = R1$
  - $R1 = m[2048]$
  - $R1 = R1 + 4$
  - $m[2056] = R1$
- echivalent, dar mai bine pentru pipelining
  - $R1 = m[1024]$
  - $R1 = R1 + 2$
  - $m[1032] = R1$
  - $R2 = m[2048]$
  - $R2 = R2 + 4$
  - $m[2056] = R2$
- ultimele 3 instrucțiuni se pot executa acum în paralel cu primele 3

# PIPELINING: DATA HAZARDS

- ce se întâmplă în procesoarele moderne?
  - ce citesc câteva sute de instrucțiuni la un moment dat
  - în hardware, se realizează un graf (*data-flow graph*) de dependențe între aceste instrucțiuni
  - exemplu: quad(a, b, c)
    - $t_1 = a * c;$
    - $t_2 = 4 * t_1;$
    - $t_3 = b * b;$
    - $t_4 = t_3 - t_2;$
    - $t_5 = \sqrt{t_4};$
    - $t_6 = -b;$
    - $t_7 = t_6 - t_5;$
    - $t_8 = t_6 + t_5;$
    - $t_9 = 2 * a;$
    - $r_1 = t_7/t_9;$
    - $r_2 = t_8/t_9;$

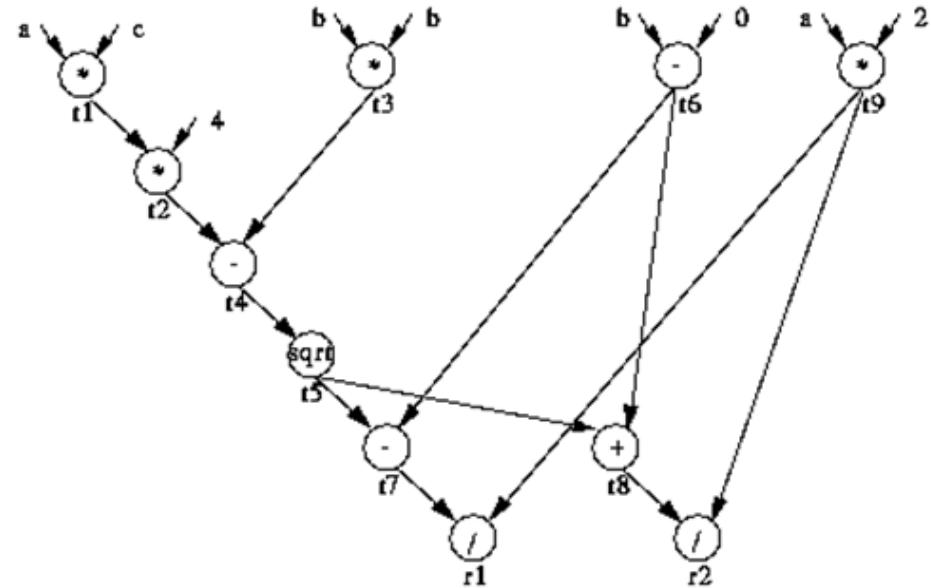


cum executăm eficient acest graf?

# PIPELINING: DATA HAZARDS

- ce se întâmplă în procesoarele moderne?
  - ce citesc câteva sute de instrucțiuni la un moment dat
  - în hardware, se realizează un graf (*data-flow graph*) de dependențe între aceste instrucțiuni
- exemplu: quad(a, b, c)
  - $t_1 = a * c; t_3 = b * b; t_6 = -b; t_9 = 2 * a;$
  - $t_2 = 4 * t_1;$
  - $t_4 = t_3 - t_2;$
  - $t_5 = \sqrt{t_4};$
  - $t_7 = t_6 - t_5; t_8 = t_6 + t_5;$
  - $r_1 = t_7/t_9; r_2 = t_8/t_9;$

6 vs. 11 pași



out of order execution

# PIPELINING: BRANCH PREDICTION

- considerați următoarea secvență de cod Assembly

```
f2:  
    pushq  %rbx  
    xorl  %ebx, %ebx  
.L3:  
    movl  %ebx, %edi  
    addl  $1, %ebx  
    call  callfunc  
    cmpl  $10, %ebx  
    jne   .L3  
    popq  %rbx  
    ret
```

- ce face codul de mai sus?

# PIPELINING: BRANCH PREDICTION

- considerați următoarea secvență de cod Assembly

```
f2:  
    pushq  %rbx  
    xorl  %ebx, %ebx  
.L3:  
    movl  %ebx, %edi  
    addl  $1, %ebx  
    call  callfunc  
    cmpl  $10, %ebx  
    jne   .L3  
    popq  %rbx  
    ret
```

- ce face codul de mai sus?
  - for loop, apeleaza funcția *callfunc*
  - unde e problema pentru pipelining?

# PIPELINING: BRANCH PREDICTION

- considerați următoarea secvență de cod Assembly

```
f2:  
    pushq  %rbx  
    xorl  %ebx, %ebx  
.L3:  
    movl  %ebx, %edi  
    addl  $1, %ebx  
    call  callfunc  
    cmpl  $10, %ebx  
→jne   .L3  
    popq  %rbx  
    ret
```

- ce face codul de mai sus?
  - for loop, apeleaza funcția *callfunc*
  - unde e problema pentru pipelining?
    - avem două posibilități pentru următoarea instrucție:
      - popq %rbx
      - movl %ebx, %edi

# PIPELINING: BRANCH PREDICTION

- o potențială soluție: branch prediction (predicția saltului)

```
f2:  
    pushq  %rbx  
    xorl  %ebx, %ebx  
.L3:  
    movl  %ebx, %edi  
    addl  $1, %ebx  
    call  callfunc  
    cmpl  $10, %ebx  
→jne   .L3  
    popq  %rbx  
    ret
```

- în general predicția este binară (dacă sari sau nu, *then* sau *else*)
- ce propuneți voi?

# PIPELINING: BRANCH PREDICTION

- o potențială soluție: branch prediction (predicția saltului)

```
f2:  
    pushq  %rbx  
    xorl  %ebx, %ebx  
.L3:  
    movl  %ebx, %edi  
    addl  $1, %ebx  
    call  callfunc  
    cmpl  $10, %ebx  
→jne   .L3  
    popq  %rbx  
    ret
```

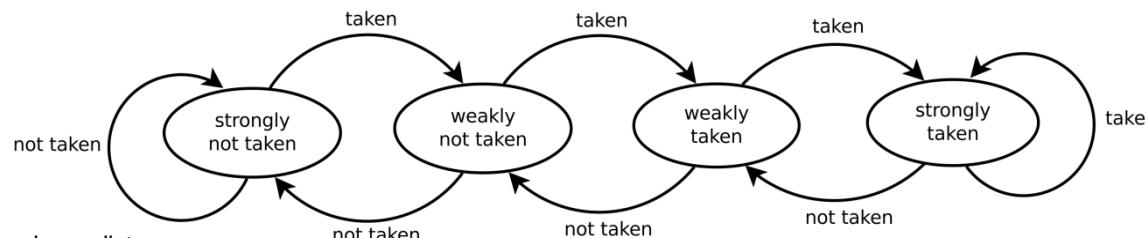
- în general predicția este binară (dacă sari sau nu, *then* sau *else*)
- ce propuneți voi?
  - predicție fixă: mereu sare / mereu nu sare
  - predicția de la pasul anterior: ce a făcut instrucțiunea ultima dată
  - predicția cu istoric: ce face de obicei instrucțiunea
  - execuție speculativă (eager execution): calculează cu și fără salt

# PIPELINING: BRANCH PREDICTION

- o potențială soluție: branch prediction (predicția saltului)

```
f2:  
    pushq  %rbx  
    xorl  %ebx, %ebx  
.L3:  
    movl  %ebx, %edi  
    addl  $1, %ebx  
    call  callfunc  
    cmpl  $10, %ebx  
→jne .L3  
    popq  %rbx  
    ret
```

- în general predicția este binară (dacă sari sau nu, *then* sau *else*)
- ce propuneți voi?
  - predicția cu istoric: ce face de obicei instrucțiunea



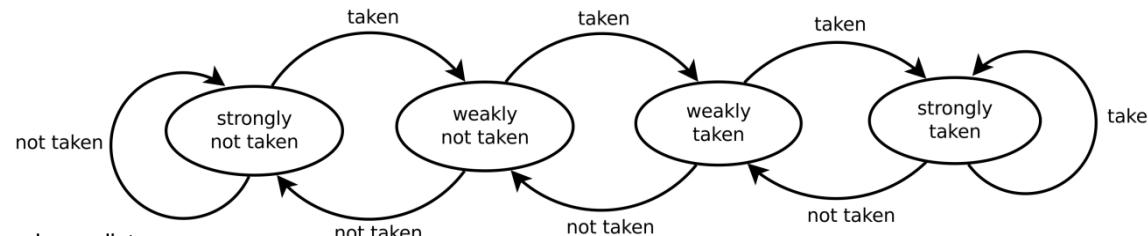
ce e asta?

# PIPELINING: BRANCH PREDICTION

- o potențială soluție: branch prediction (predicția saltului)

```
f2:  
    pushq  %rbx  
    xorl  %ebx, %ebx  
.L3:  
    movl  %ebx, %edi  
    addl  $1, %ebx  
    call  callfunc  
    cmpl  $10, %ebx  
→jne .L3  
    popq  %rbx  
    ret
```

- în general predicția este binară (dacă sari sau nu, *then* sau *else*)
- ce propuneți voi?
  - predicția cu istoric: ce face de obicei instrucțiunea



ce e asta?  
counter 2 biți

# PIPELINING

- când complicăm hardware și software pot apărea probleme
- în special probleme de securitate
  - meltdown, spectre
  - aceste două atacuri exploatează execuția speculativă și sistemul ierarhic al memoriei (cache-ul)
- când complicăm hardware, e cu atât mai rău
  - soluția: trebuie înlocuit hardware-ul
  - soluția: sistemul de operare trebuie să ia în considerare problema
    - *totul* va fi mai lent

# **CE AM FĂCUT ASTĂZI**

- pipelining
- branch prediction
- out of order execution

# **DATA VIITOARE ...**

- sisteme multi-procesor
- ierarhia memoriei, caching
- performanța calculatoarelor

# LECTURĂ SUPLIMENTARĂ

- **PH book**
  - 4.5 An Overview of Pipelining
  - 4.6 Pipelined Datapath and Control
  - 4.7 Data Hazards: Forwarding versus Stalling
  - 4.8 Control Hazards
  - 4.9 Exceptions
  - 4.10 Parallelism via Instructions
- **Crash Course Computer Science**
  - Advanced CPU Designs,  
<https://www.youtube.com/watch?v=rtAIC5J1U40>



# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x09**

**SISTEME MULTI-PROCESOR, IERARHIA MEMORIEI**

Cristian Rusu

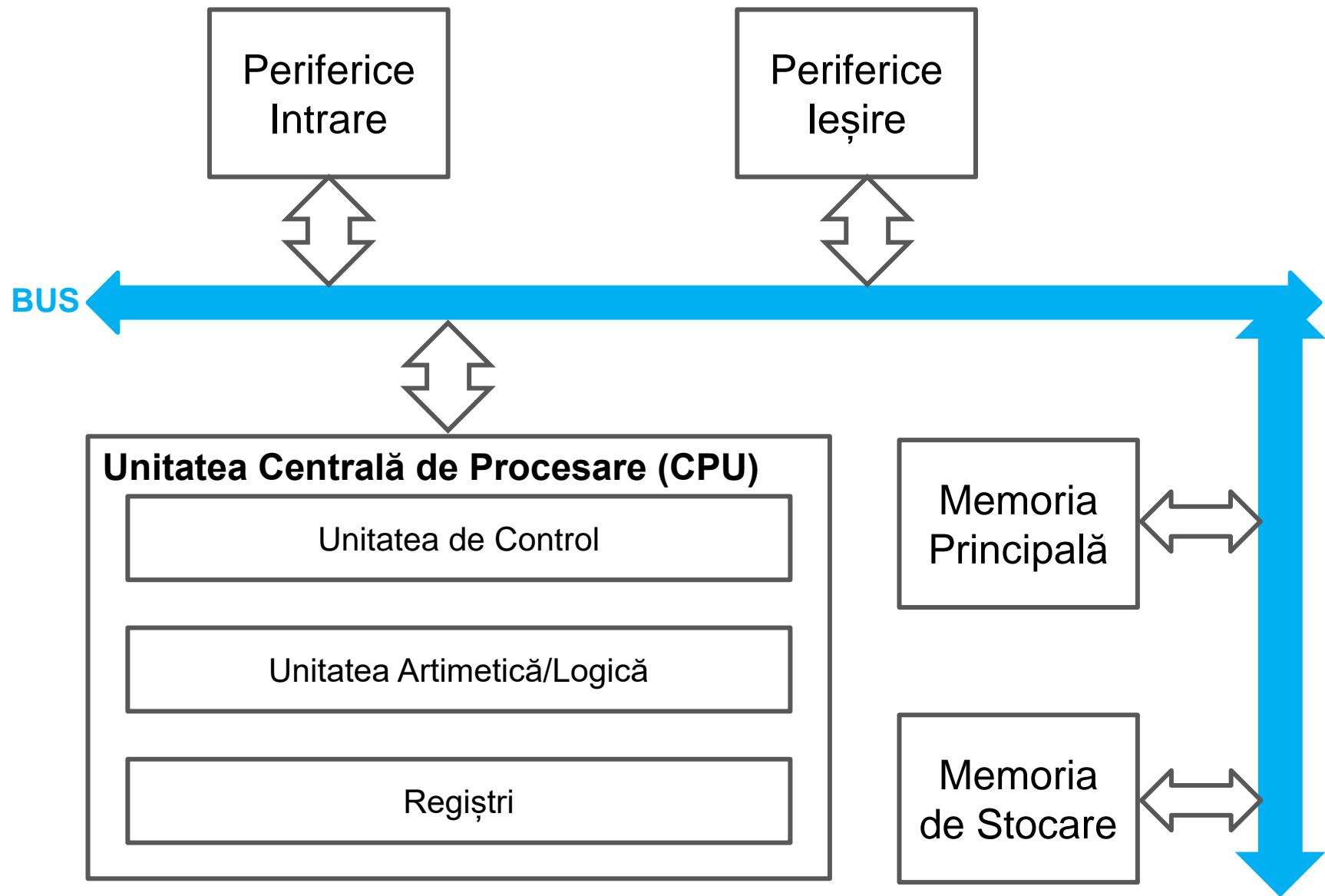
# **DATA TRECUTĂ**

- pipelining
- branch prediction
- out of order execution

# CUPRINS

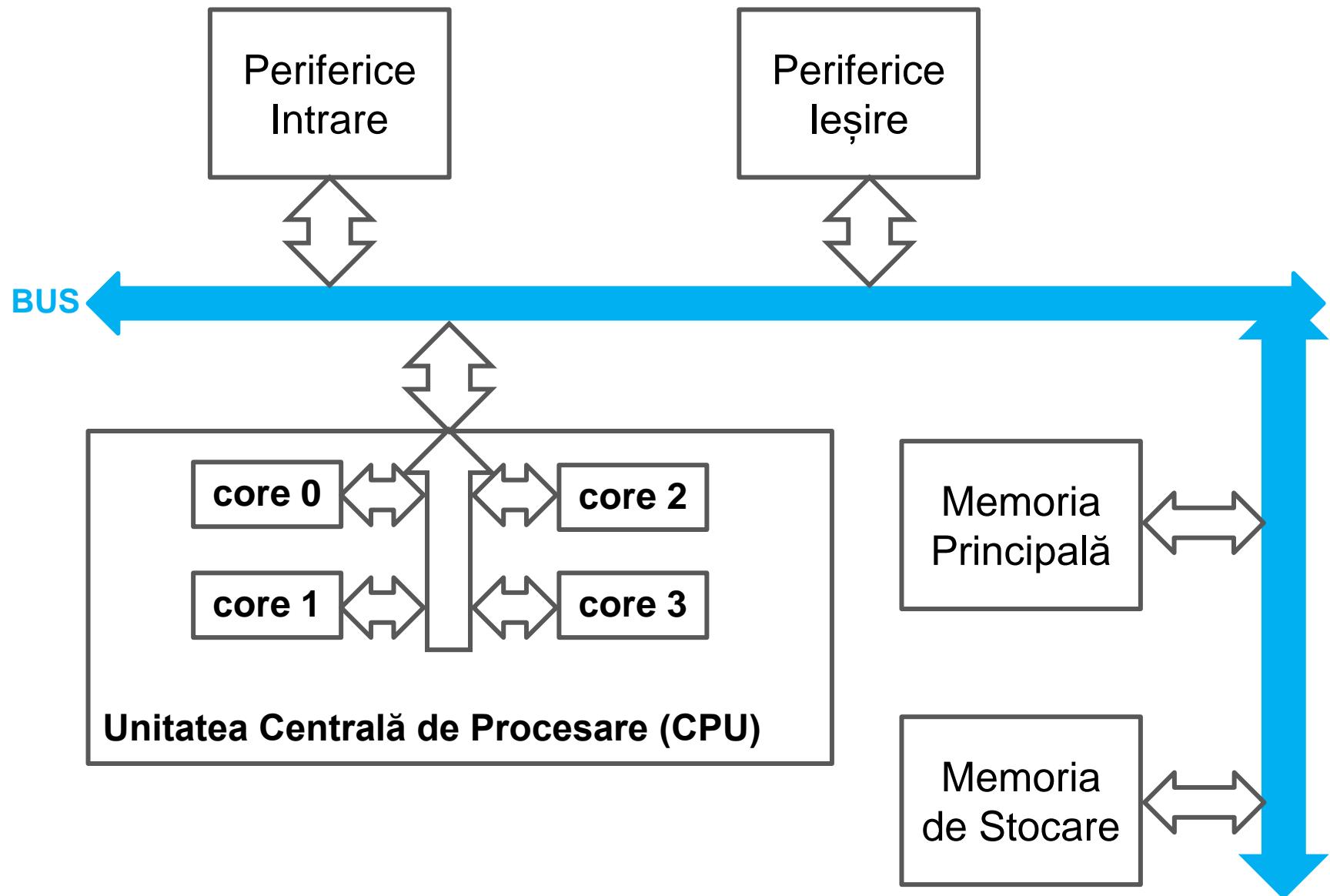
- sisteme multi-procesor
- ierarhia memoriei
- caching

# ARHITECTURA DE BAZĂ



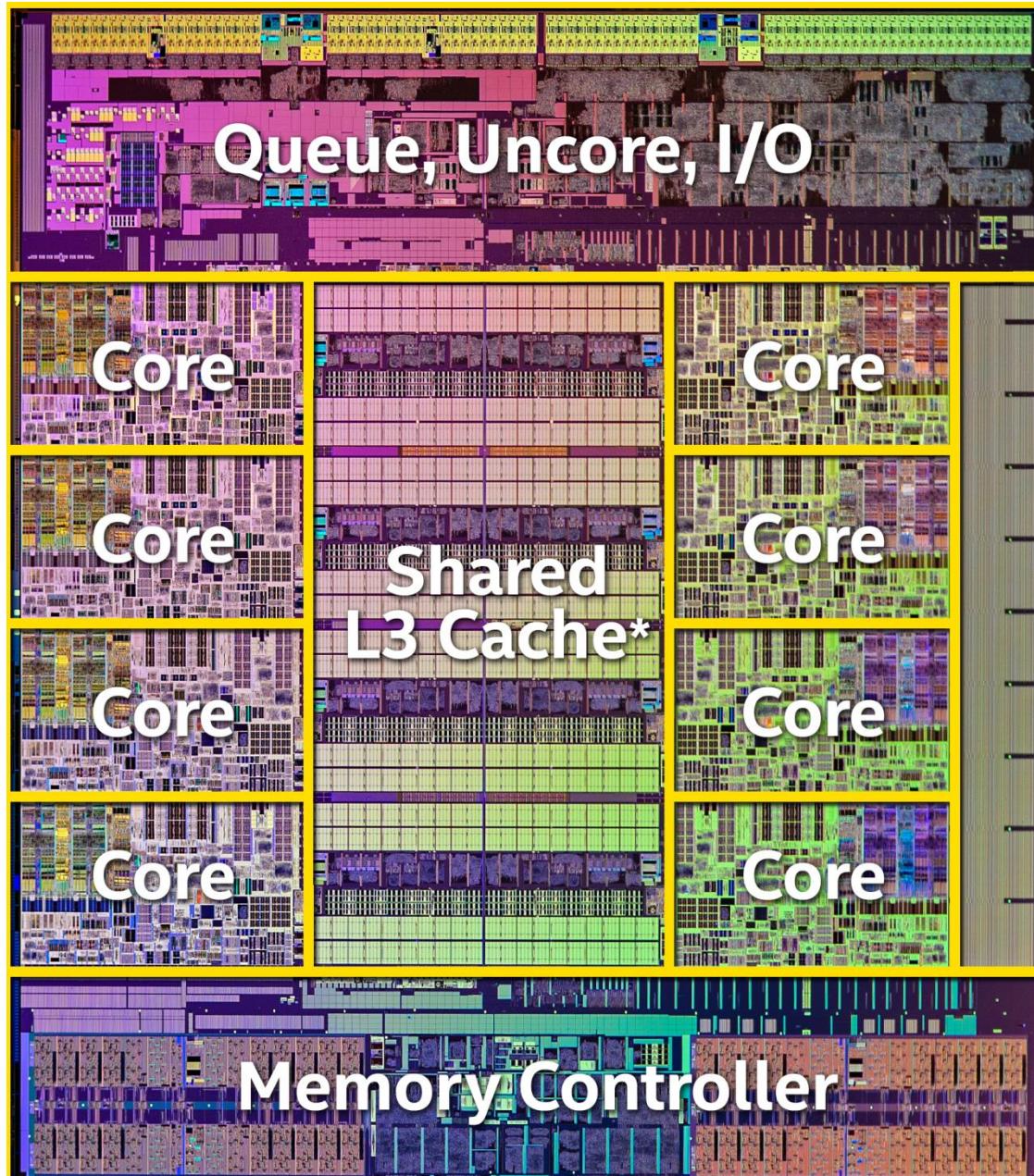
rularea simultană a programelor este “simulată”

# ARHITECTURA MULTI-CORE

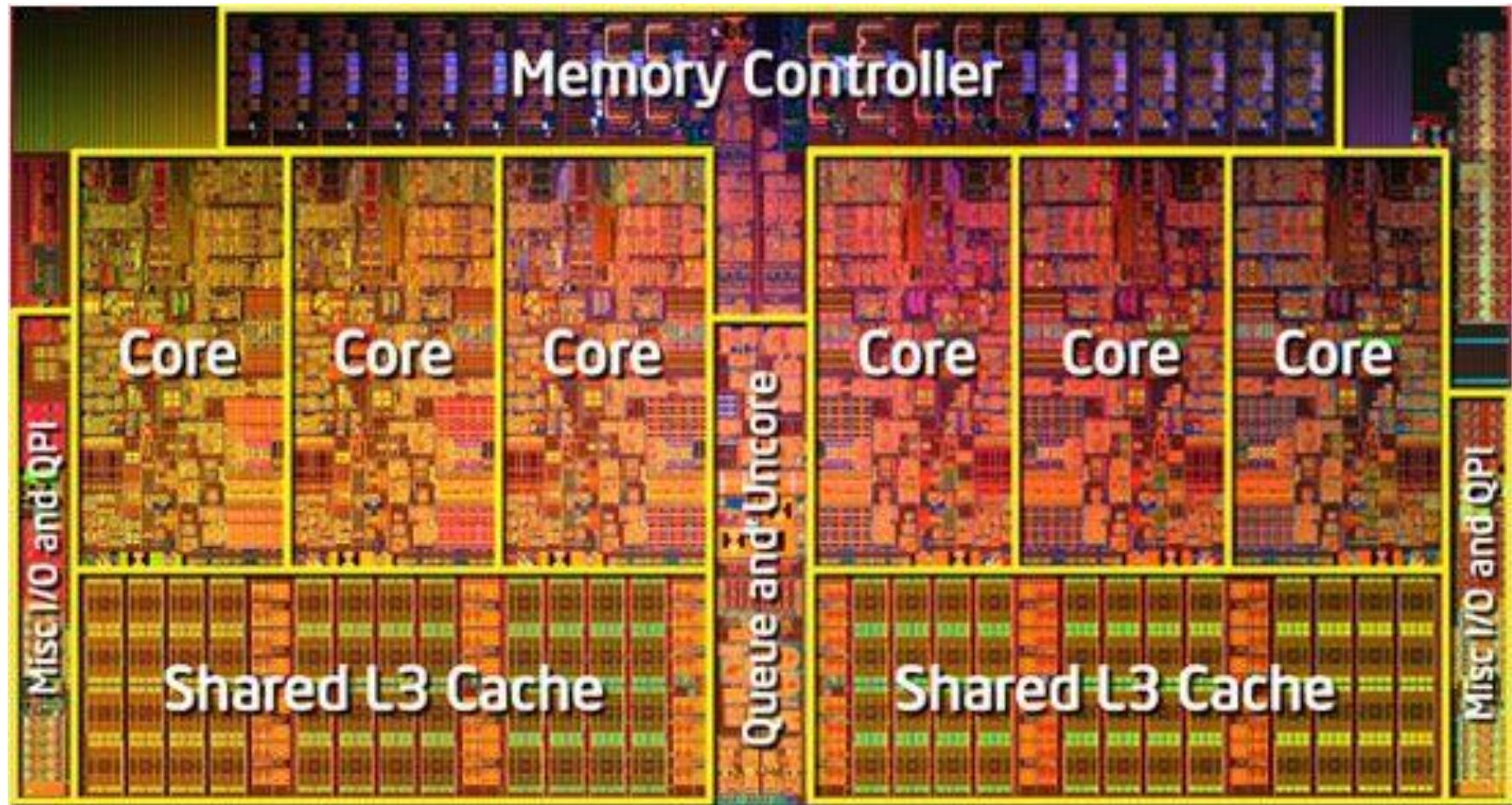


rularea simultană a programelor este “reală”

# ARHITECTURA MULTI-CORE



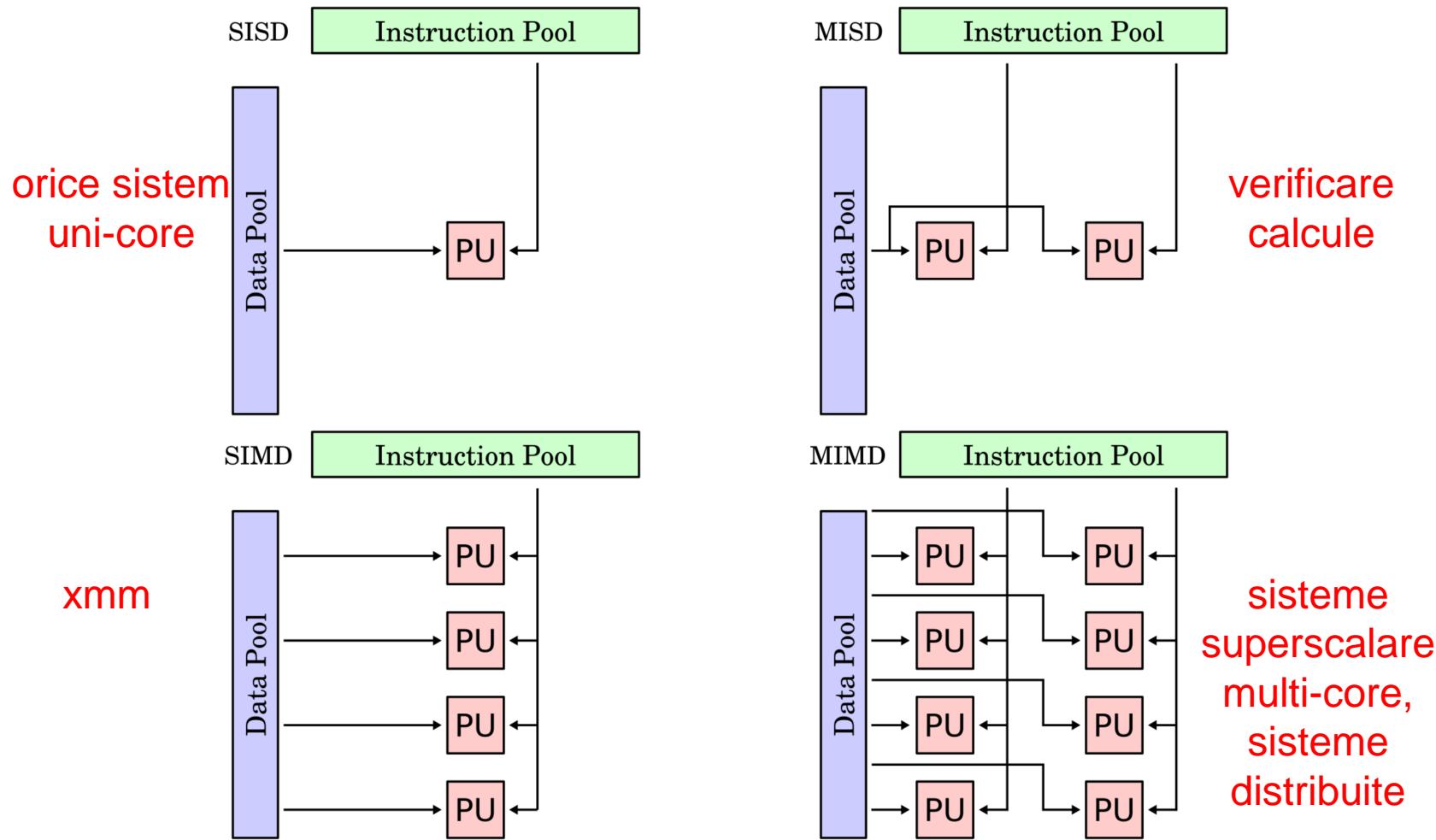
# ARHITECTURA MULTI-CORE



e bine că avem multe core-uri, noua problemă: să aducem datele la core-uri

# ARHITECTURA MULTI-CORE

- taxonomia Flynn



# ARHITECTURA MULTI-CORE

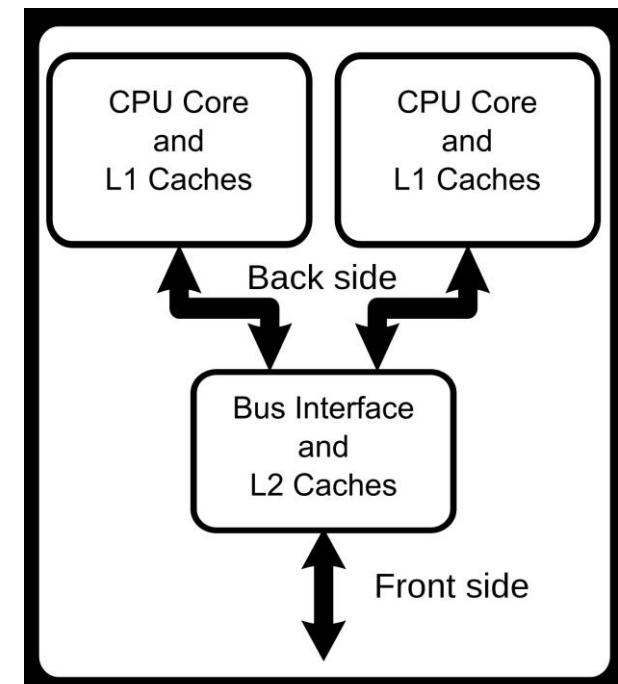
- **problema:**
  - sigur toate core-urile vor instrucțiuni
  - probabil toate core-urile vor să acceseze memoria
  - din când în când core-uri vor să facă operații I/O
  - toate comunică pe același bus
    - conflicte de acces
    - coadă de priorități pentru acces
- **soluția:**
  - fiecare core are “o memorie” locală cu care să poată comunica rapid
  - ierarhizarea memoriei

# TIPURI DE PARALELISM

- **la nivel de biți**
  - modificarea dimensiunii “cuvintelor” procesorului (procesoarea pe 16, 32 și 64 de biți)
  - cu câți biți poate sistemul de calcul să opereze
- **la nivel de instrucții**
  - pipelines
- **la nivel de task-uri**
  - multi-thread
  - multi-process
- **la nivel de blocuri**
  - vectorizarea operațiilor
  - operații pe blocuri

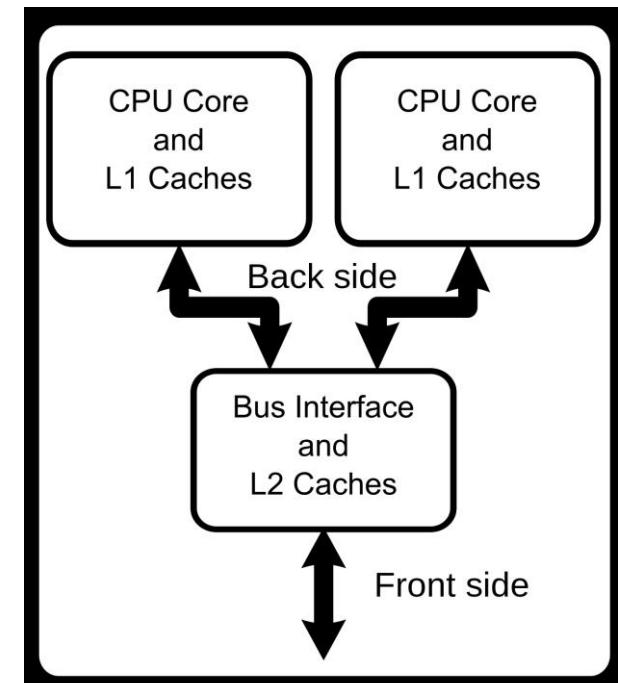
# TIPURI DE PARALELISM

- în trecut 1 procesor = 1 unitate de calcul
- de câteva decenii 1 procesor = mai multe (2,4,...) unități de calcul
  - sisteme multi-core
  - unitățile de calcul au resurse proprii: registrii, ALU, FPU, cache...
  - dar unele resurse sunt împărtășite între toate unitățile de calcul (cache L3, controller-ul de memorie)
- avantaje: coerența cache-ului
- dezavantaj: software special



# TIPURI DE PARALELISM

- **în trecut 1 procesor = 1 unitate de calcul**
- **de câteva decenii 1 procesor = mai multe (2,4,...) unități de calcul**
  - sisteme multi-thread: Hyper-Threading, Chip Multi-threading
  - unitățile de calcul au resurse proprii: regiștri (și cam atât)
  - restul resurselor de calcul sunt împărtășite de thread-uri
  - threadurile sunt blocate dacă resursele de calcul sunt ocupate
- avantaj: dacă resursa este disponibilă
- dezavantaj: competiție pentru resurse
- hyper-threading = logical cores

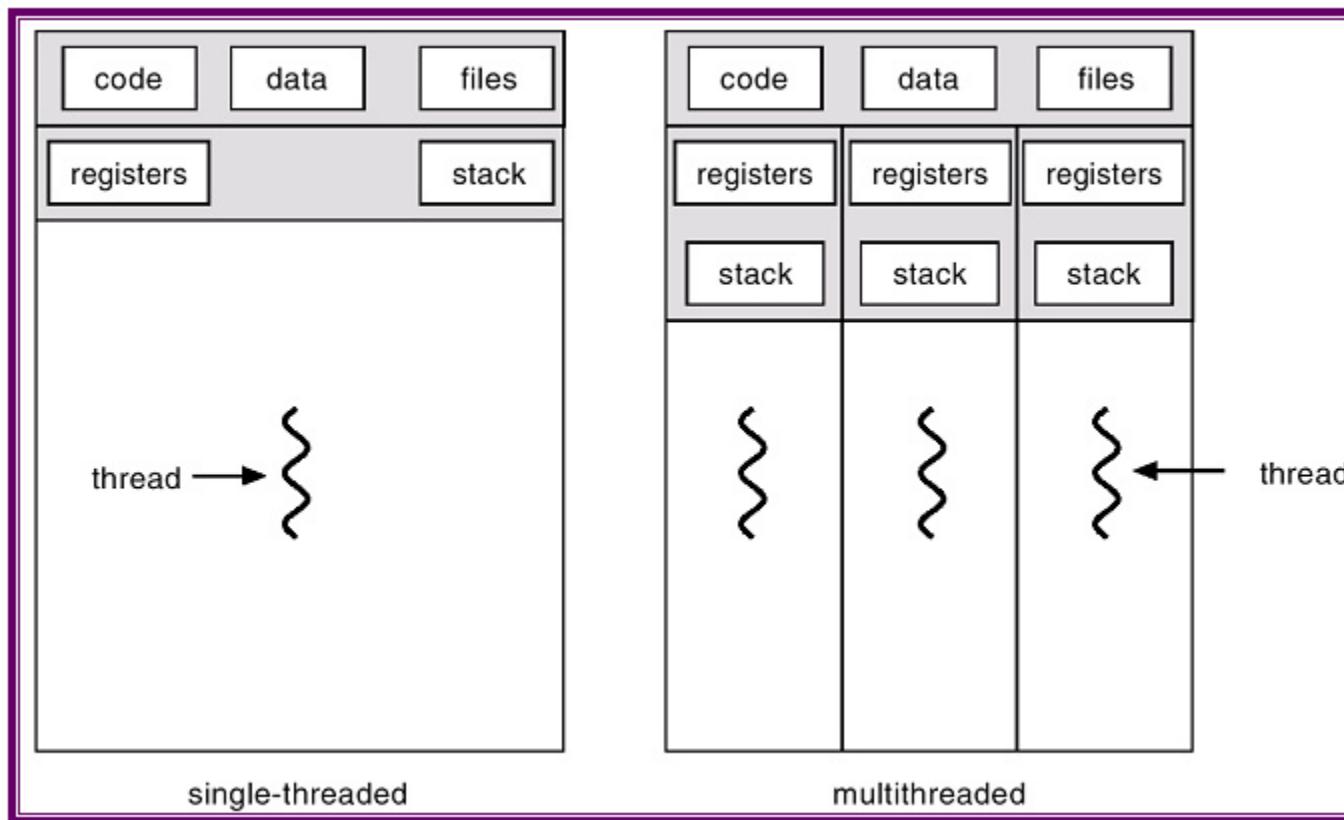


# TIPURI DE PARALELISM

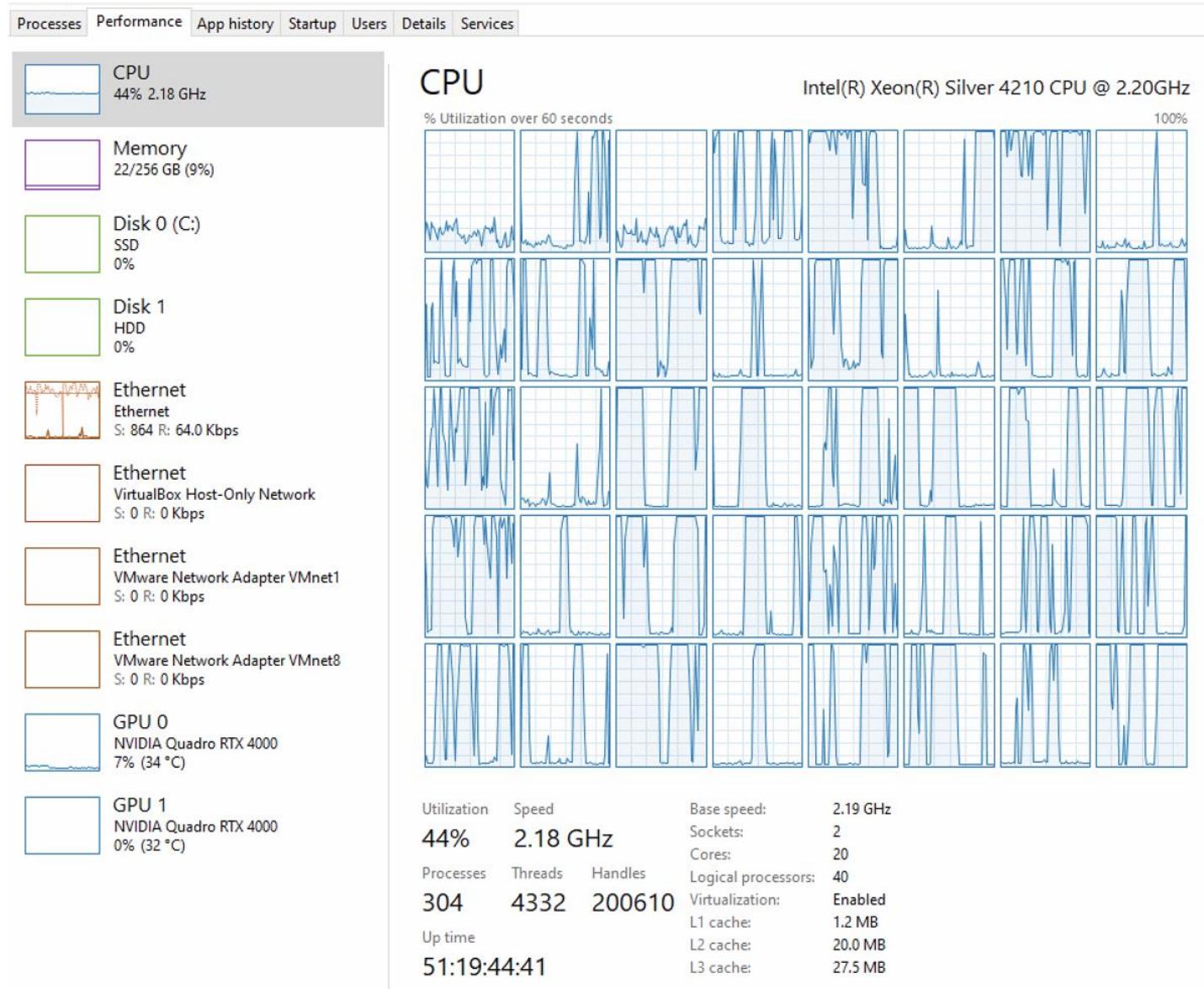
- **Instruction-level parallelism (ILP)**
  - Instruction Pipelining
  - Register Renaming
  - Speculative Execution
  - Branch Prediction
  - Value Prediction
  - Memory Dependence Prediction
  - Cache Latency Prediction
  - Out-of-order Execution
  - Dataflow Analysis/Execution

# TIPURI DE PARALELISM

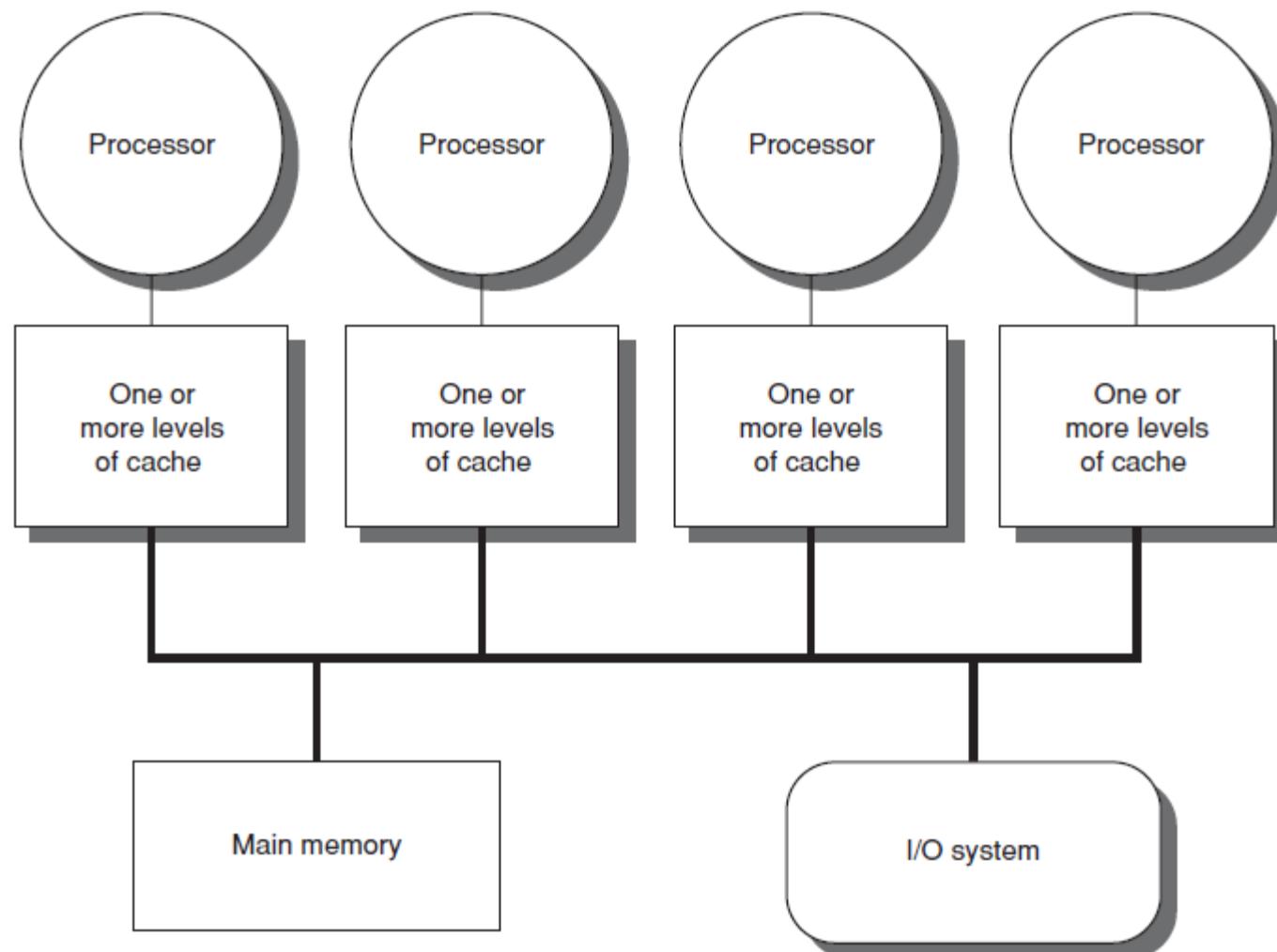
- Task parallelism (ILP)
  - multi-thread
  - multi-process



# TIPURI DE PARALELISM

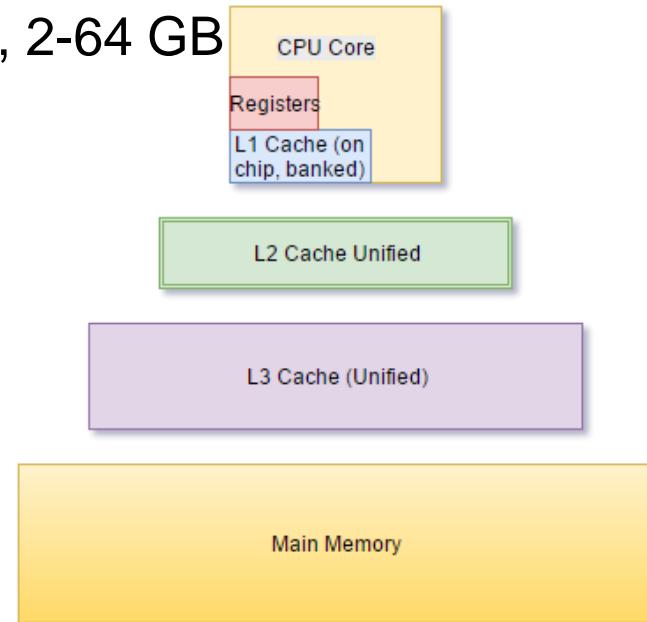


# IERARHIZAREA MEMORIEI

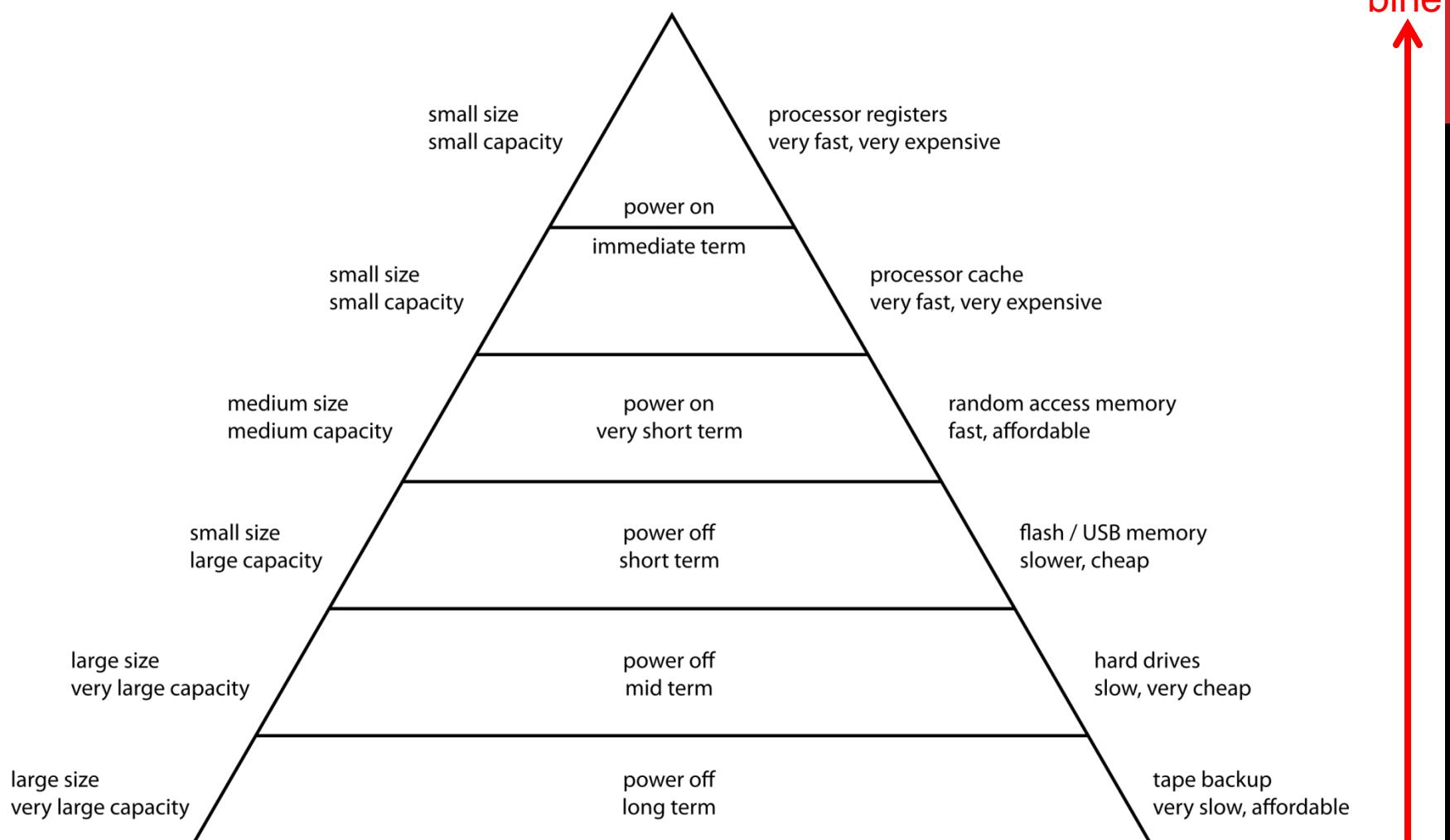


# IERARHIZAREA MEMORIEI

- tipuri de memorie
  - regiștrii procesorului: acces imediat, 100-1000 bytes
  - cache L0: acces foarte rapid, 5-20 kbytes
  - cache L1 (cache instrucțiuni și date): 700GB/s, 100-500 kbytes
  - cache L2: 200GB/s, 500-1000 kbytes
  - cache L3 (de obicei partajat): 100GB/s, 1-5 MB
- memoria principală RAM: 100-500 MB/s, 2-64 GB
- disc HD/SSD: 10-100 MB/s, 1TB

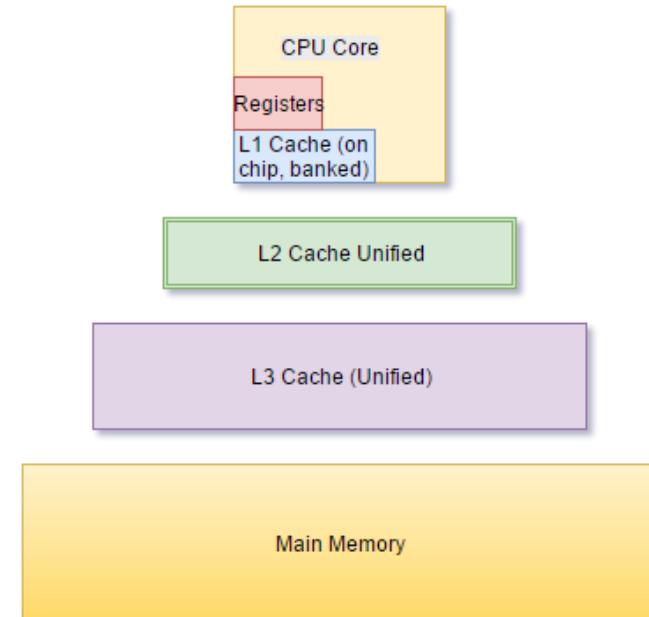


# IERARHIZAREA MEMORIEI



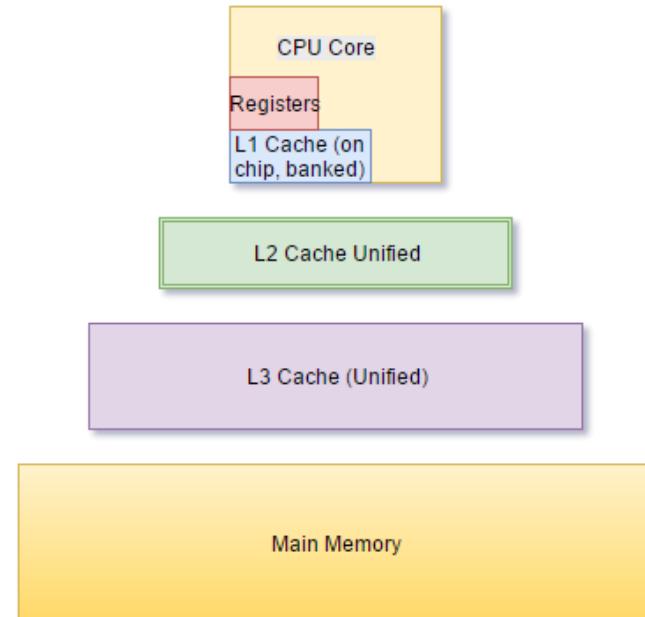
# IERARHIZAREA MEMORIEI

- de ce e bine să avem cache?
- presupunem că timpii de acces sunt:
  - în memoria principală: 50 ns
  - în L1: 1 ns (dar există o probabilitate de 10% ca în L1 să nu găsim ceea ce căutăm, i.e., 10% miss rate)
  - în L2: 5 ns cu 1% miss rate
  - în L3: 10 ns cu 0.2% miss rate
- să presupunem că vrem să accesăm o bucată de memorie, cât ne costă ca timp dacă:
  - verificăm în RAM:



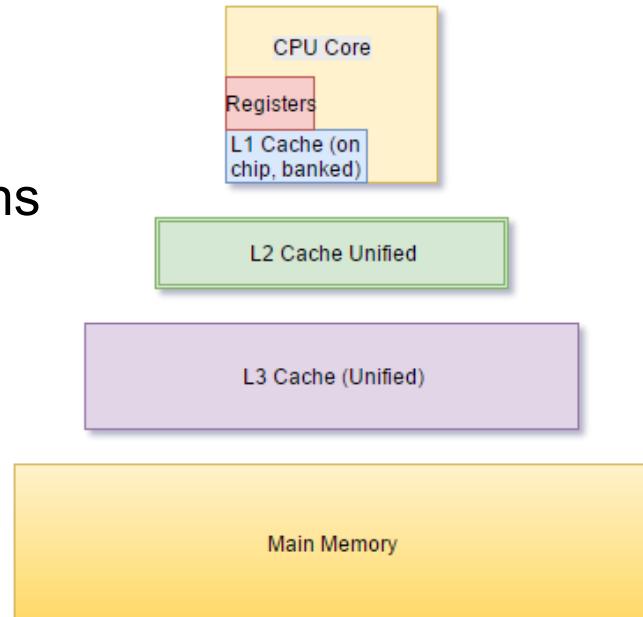
# IERARHIZAREA MEMORIEI

- de ce e bine să avem cache?
- presupunem că timpii de acces sunt:
  - în memoria principală: 50 ns
  - în L1: 1 ns (dar există o probabilitate de 10% ca în L1 să nu găsim ceea ce căutăm, i.e., 10% miss rate)
  - în L2: 5 ns cu 1% miss rate
  - în L3: 10 ns cu 0.2% miss rate
- să presupunem că vrem să accesăm o bucată de memorie, cât ne costă ca timp dacă:
  - verificăm în RAM: 50 ns
  - verificăm în L1:



# IERARHIZAREA MEMORIEI

- de ce e bine să avem cache?
- presupunem că timpii de acces sunt:
  - în memoria principală: 50 ns
  - în L1: 1 ns (dar există o probabilitate de 10% ca în L1 să nu găsim ceea ce căutăm, i.e., 10% miss rate)
  - în L2: 5 ns cu 1% miss rate
  - în L3: 10 ns cu 0.2% miss rate
- să presupunem că vrem să accesăm o bucată de memorie, cât ne costă ca timp dacă:
  - verificăm în RAM: 50 ns
  - verificăm în L1:  $1\text{ ns} + (0.1 \times 50\text{ ns}) = 6\text{ ns}$
  - verificăm în L2:



# IERARHIZAREA MEMORIEI

- de ce e bine să avem cache?
- presupunem că timpii de acces sunt:
  - în memoria principală: 50 ns
  - în L1: 1 ns (dar există o probabilitate de 10% ca în L1 să nu găsim ceea ce căutăm, i.e., 10% miss rate)
  - în L2: 5 ns cu 1% miss rate
  - în L3: 10 ns cu 0.2% miss rate
- să presupunem că vrem să accesăm o bucată de memorie, cât ne costă ca timp dacă:
  - verificăm în RAM: 50 ns
  - verificăm în L1:  $1\text{ ns} + (0.1 \times 50\text{ ns}) = 6\text{ ns}$
  - verificăm în L2:  $1\text{ ns} + (0.1 \times (5\text{ ns} + (0.01 \times 50\text{ ns}))) = 1.55\text{ ns}$
  - verificăm în L3:

# IERARHIZAREA MEMORIEI

- de ce e bine să avem cache?
- presupunem că timpii de acces sunt:
  - în memoria principală: 50 ns
  - în L1: 1 ns (dar există o probabilitate de 10% ca în L1 să nu găsim ceea ce căutăm, i.e., 10% miss rate)
  - în L2: 5 ns cu 1% miss rate
  - în L3: 10 ns cu 0.2% miss rate
- să presupunem că vrem să accesăm o bucată de memorie, cât ne costă ca timp dacă:
  - verificăm în RAM: 50 ns
  - verificăm în L1:  $1\text{ ns} + (0.1 \times 50\text{ ns}) = 6\text{ ns}$
  - verificăm în L2:  $1\text{ ns} + (0.1 \times (5\text{ ns} + (0.01 \times 50\text{ ns}))) = 1.55\text{ ns}$
  - verificăm în L3:  $1\text{ ns} + (0.1 \times (5\text{ ns} + (0.01 \times (10\text{ ns} + (0.002 \times 50\text{ ns})))))) = 1.5101\text{ ns}$

observați trade-off-ul între viteza de acces și probabilitatea de miss rate  
cât este miss rate în RAM?

# IERARHIZAREA MEMORIEI

- de ce e bine să avem cache?
- presupunem că timpii de acces sunt:
  - în memoria principală: 50 ns
  - în L1: 1 ns (dar există o probabilitate de 10% ca în L1 să nu găsim ceea ce căutăm, i.e., 10% miss rate)
  - în L2: 5 ns cu 1% miss rate
  - în L3: 10 ns cu 0.2% miss rate
- să presupunem că vrem să accesăm o bucată de memorie, cât ne costă ca timp dacă:
  - verificăm în RAM: 50 ns
  - verificăm în L1:  $1\text{ ns} + (0.1 \times 50\text{ ns}) = 6\text{ ns}$
  - verificăm în L2:  $1\text{ ns} + (0.1 \times (5\text{ ns} + (0.01 \times 50\text{ ns}))) = 1.55\text{ ns}$
  - verificăm în L3:  $1\text{ ns} + (0.1 \times (5\text{ ns} + (0.01 \times (10\text{ ns} + (0.002 \times 50\text{ ns})))) = 1.5101\text{ ns}$

observați trade-off-ul între viteza de acces și probabilitatea de miss rate

cât este miss rate în RAM? 0% (în RAM sigur avem informația)  
cu ce dimensiune are legatură miss rate?

# IERARHIZAREA MEMORIEI

- de ce e bine să avem cache?
- presupunem că timpii de acces sunt:
  - în memoria principală: 50 ns
  - în L1: 1 ns (dar există o probabilitate de 10% ca în L1 să nu găsim ceea ce căutăm, i.e., 10% miss rate)
  - în L2: 5 ns cu 1% miss rate
  - în L3: 10 ns cu 0.2% miss rate
- să presupunem că vrem să accesăm o bucată de memorie, cât ne costă ca timp dacă:
  - verificăm în RAM: 50 ns
  - verificăm în L1:  $1\text{ ns} + (0.1 \times 50\text{ ns}) = 6\text{ ns}$
  - verificăm în L2:  $1\text{ ns} + (0.1 \times (5\text{ ns} + (0.01 \times 50\text{ ns}))) = 1.55\text{ ns}$
  - verificăm în L3:  $1\text{ ns} + (0.1 \times (5\text{ ns} + (0.01 \times (10\text{ ns} + (0.002 \times 50\text{ ns})))))) = 1.5101\text{ ns}$

observați trade-off-ul între viteza de acces și probabilitatea de miss rate

cât este miss rate în RAM? 0% (în RAM sigur avem informația)

cu ce dimensiune are legatură miss rate? cu dimensiunea memoriei

# IERARHIZAREA MEMORIEI

- de ce e bine să avem cache?
- presupunem că timpii de acces sunt:
  - în memoria principală: 50 ns
  - în L1: 1 ns (dar există o probabilitate de 10% ca în L1 să nu găsim ceea ce căutăm, i.e., 10% miss rate)
  - în L2: 5 ns cu 1% miss rate
  - în L3: 10 ns cu 0.2% miss rate
- să presupunem că vrem să accesăm o bucată de memorie, cât ne costă ca timp dacă:
  - verificăm în RAM: 50 ns
  - verificăm în L1:  $1\text{ ns} + (0.1 \times 50\text{ ns}) = 6\text{ ns}$
  - cu un miss rate de 10% merită să avem cache L1
  - pentru ce probabilitate miss rate nu mai merită cache L1?

# IERARHIZAREA MEMORIEI

- de ce e bine să avem cache?
- presupunem că timpii de acces sunt:
  - în memoria principală: 50 ns
  - în L1: 1 ns (dar există o probabilitate de 10% ca în L1 să nu găsim ceea ce căutăm, i.e., 10% miss rate)
  - în L2: 5 ns cu 1% miss rate
  - în L3: 10 ns cu 0.2% miss rate
- să presupunem că vrem să accesăm o bucată de memorie, cât ne costă ca timp dacă:
  - verificăm în RAM: 50 ns
  - verificăm în L1:  $1\text{ ns} + (0.1 \times 50\text{ ns}) = 6\text{ ns}$
  - cu un miss rate de 10% merită să avem cache L1
  - pentru ce probabilitate miss rate nu mai merită cache L1?
    - $p = 49 / 50 = 98\%$

# IERARHIZAREA MEMORIEI

- de ce e bine să avem cache?
- presupunem că timpii de acces sunt:
  - în memoria principală: 50 ns
  - în L1: 1 ns (dar există o probabilitate de 10% ca în L1 să nu găsim ceea ce căutăm, i.e., 10% miss rate)
  - în L2: 5 ns cu 1% miss rate
  - în L3: 10 ns cu 0.2% miss rate
- Exemplu:
  - memoria RAM este 1 GB
  - memoria cache L1 este 128 kbytes
  - memoria RAM este de aproximativ 8000 de ori mai multă decât memoria cache L1, deci cum putem avea miss rate 10%?
  - ne bazăm pe **principiul de localizare**

# CACHE

- **principiul de localizare**
  - presupunem că avem în RAM un vector de 1000 elemente



- cache L1 este gol



- cache L2 este gol



**citim elementul  $a_0$  (avem nevoie să procesăm vectorul): elementul nu e în L1, nu e în L2, trebuie să mergem în RAM, dar după ce îl citim din RAM copiem un segment din vector în L1 și L2**

# CACHE

- **principiul de localizare**
  - presupunem că avem în RAM un vector de 1000 elemente



- cache L1 conține  $a_0$  și următoarele elemente din vector



- cache L2 conține  $a_0$  și mai multe elemente din vector



**citim elementul  $a_1$ : este deja în L1 (cache hit!)**

# CACHE

- **principiul de localizare**
  - presupunem că avem în RAM un vector de 1000 elemente



- cache L1 conține a<sub>0</sub> și următoarele elemente din vector



- cache L2 conține a<sub>0</sub> și mai multe elemente din vector



**citim elementul a<sub>2</sub>: este deja în L1 (cache hit!)**

# CACHE

- **principiul de localizare**
  - presupunem că avem în RAM un vector de 1000 elemente



- cache L1 conține a<sub>0</sub> și următoarele elemente din vector



- cache L2 conține a<sub>0</sub> și mai multe elemente din vector



**citim elementul a<sub>3</sub>: este deja în L1 (cache hit!)**

# CACHE

- **principiul de localizare**
  - presupunem că avem în RAM un vector de 1000 elemente



- cache L1 conține  $a_0$  și următoarele elemente din vector



- cache L2 conține  $a_0$  și mai multe elemente din vector



**citim elementul  $a_4$ :** nu este în L1 (cache miss!), dar este în L2, deci îl citim de acolo și actualizăm în L1

# CACHE

- **principiul de localizare**

- presupunem că avem în RAM un vector de 1000 elemente



- cache L1 conține a<sub>4</sub> și următoarele elemente din vector



- cache L2 conține a<sub>0</sub> și mai multe elemente din vector



**citim elementul a<sub>5</sub>: este deja în L1 (cache hit!)**

**... când nu mai găsim nici în L2, mergem din nou în RAM și citim un nou subset din vector (a<sub>10</sub> ... a<sub>19</sub>)**

# CACHE

- **principiul de localizare**
  - presupunem că avem în RAM un vector de 1000 elemente



- cache L1 conține a<sub>4</sub> și următoarele elemente din vector



- cache L2 conține a<sub>0</sub> și mai multe elemente din vector



**miss rate pentru cache L1?**

**miss rate pentru cache L2?**

# CACHE

- **principiul de localizare**
  - presupunem că avem în RAM un vector de 1000 elemente



- cache L1 conține a<sub>4</sub> și următoarele elemente din vector



- cache L2 conține a<sub>0</sub> și mai multe elemente din vector



**miss rate pentru cache L1? 25%**

**miss rate pentru cache L2? 10%**

# CACHE

- **principiul de localizare**
  - presupunem că avem în RAM un vector de 1000 elemente



- cache L1 conține  $a_4$  și următoarele elemente din vector



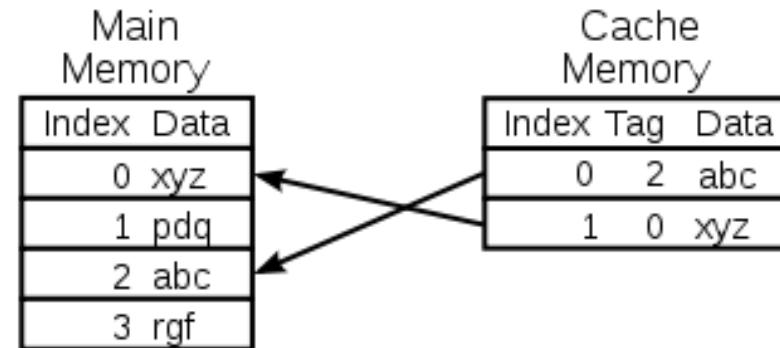
- cache L2 conține  $a_0$  și mai multe elemente din vector



**aici am accesat secvențial vectorul, dacă îl accesăm aleator, totul este pierdut (deci chiar dacă memoria se numește RAM, nu e deloc bine să accesăm datele complet Random)**

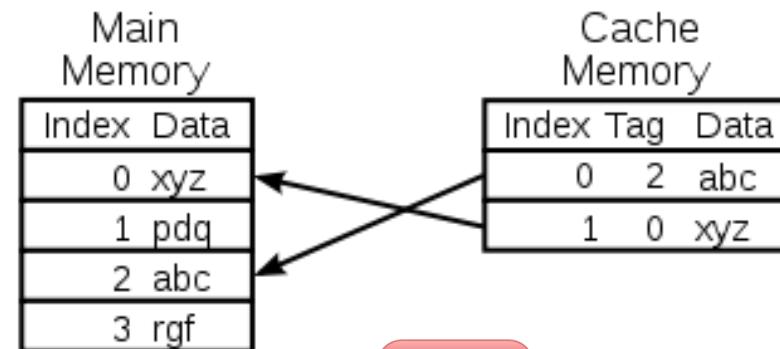
# CACHE

- corespondență dintre locația din cache și locația din RAM
  - este realizată printr-un tabel

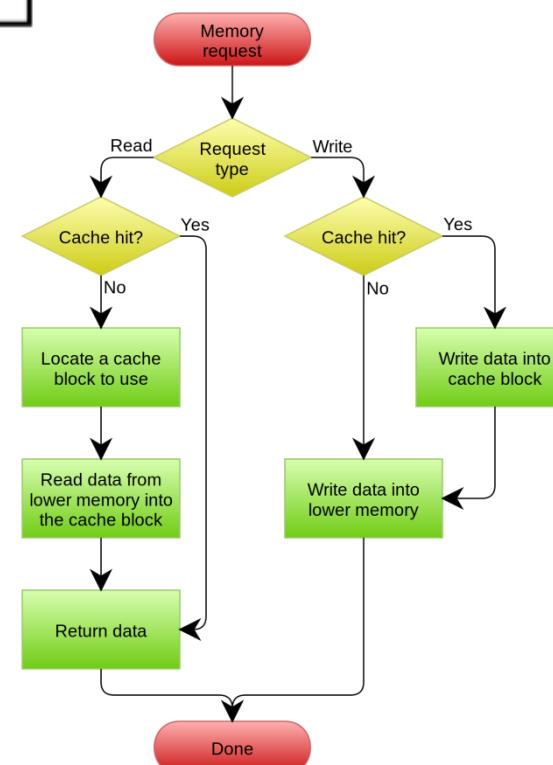


# CACHE

- corespondență dintre locația din cache și locația din RAM
  - este realizată printr-un tabel

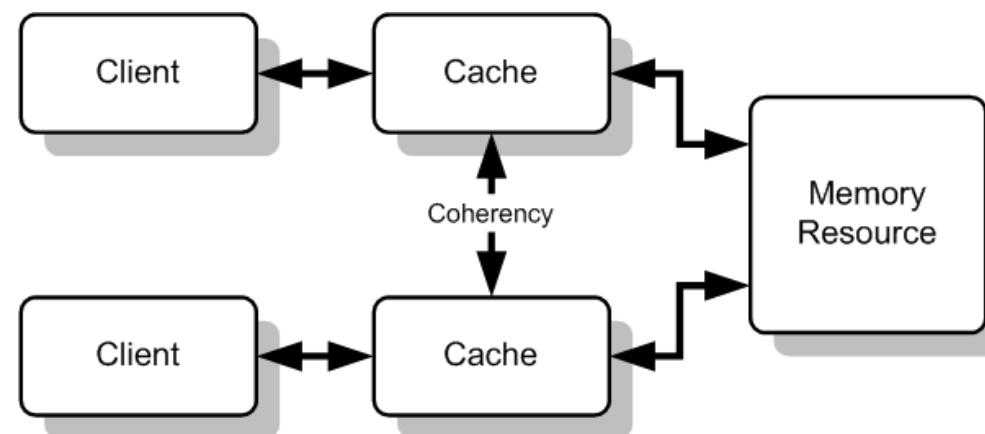


- algoritmul general:



# CACHE

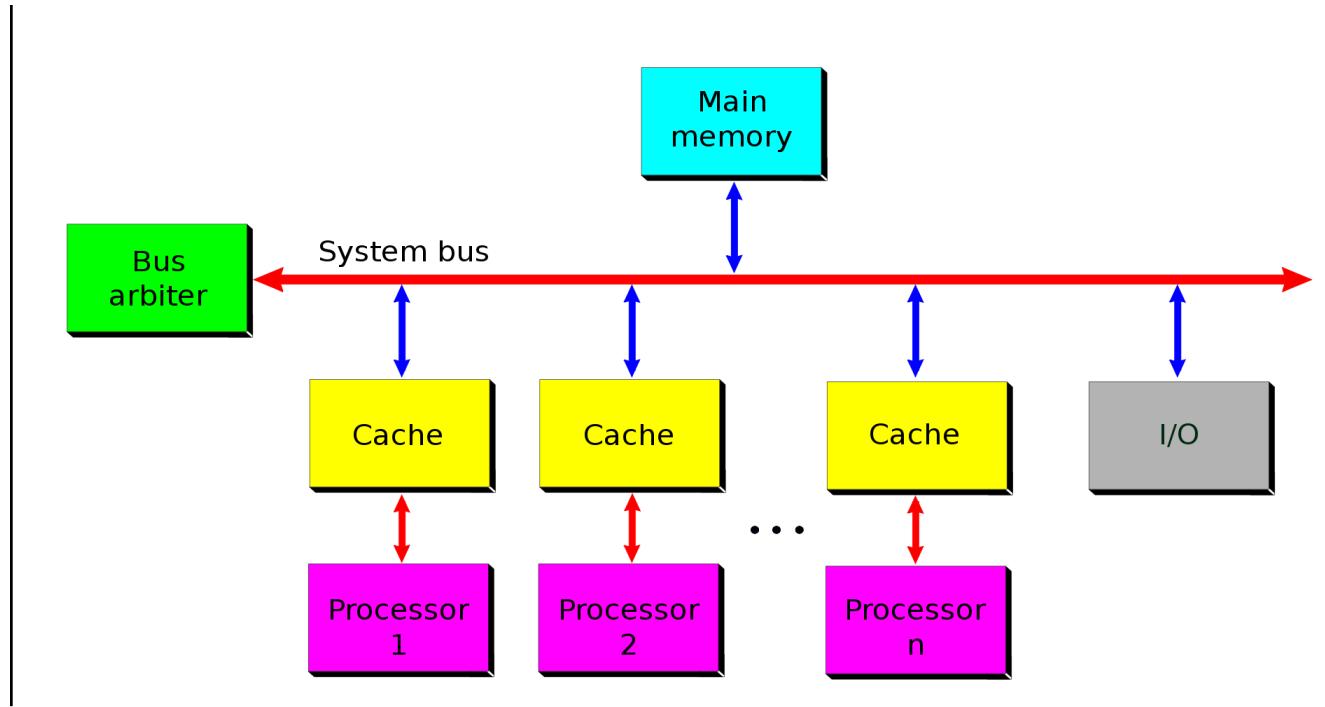
- corespondența dintre locația din cache și locația din RAM
  - la citire nu sunt niciodată probleme
  - la sciere lucrurile se complică
    - rezultatul este scris în cache
    - și celelalte memorii trebuie să fie anunțate de noua valoare
      - L1/L2/L3/RAM etc.
    - situația se complică și mai mult dacă sunt cache-uri diferite pentru fiecare core pe care îl avem: cache coherence (protocol pentru consistența tuturor cache-urilor)



# IERARHIZAREA MEMORIEI

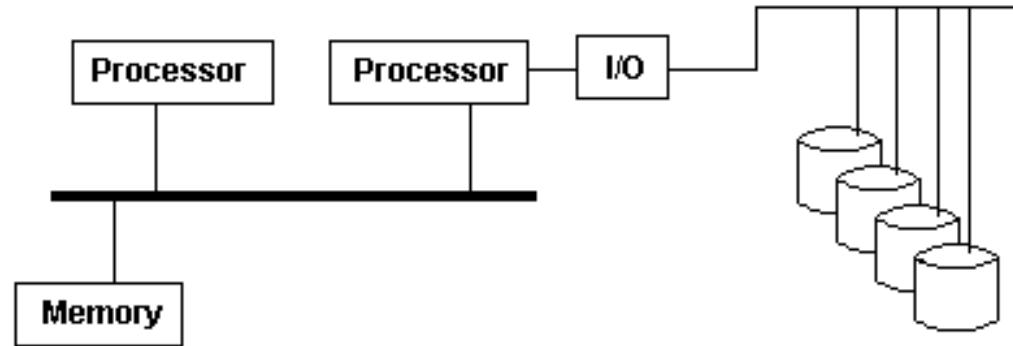
- când programați, nu vedeți această ierarhizare
- cine e responsabil de ce anume?
  - programatorul: transfer între HD/SSD și RAM (citire de pe disc)
  - logică hardware: din/în RAM în/din memoriile cache
  - compilatorul: generează cod care exploatează cache-ul
- cât timp performanța este acceptabilă totul e OK, apoi Assembly

# SYMMETRIC MULTIPROCESS SYSTEMS



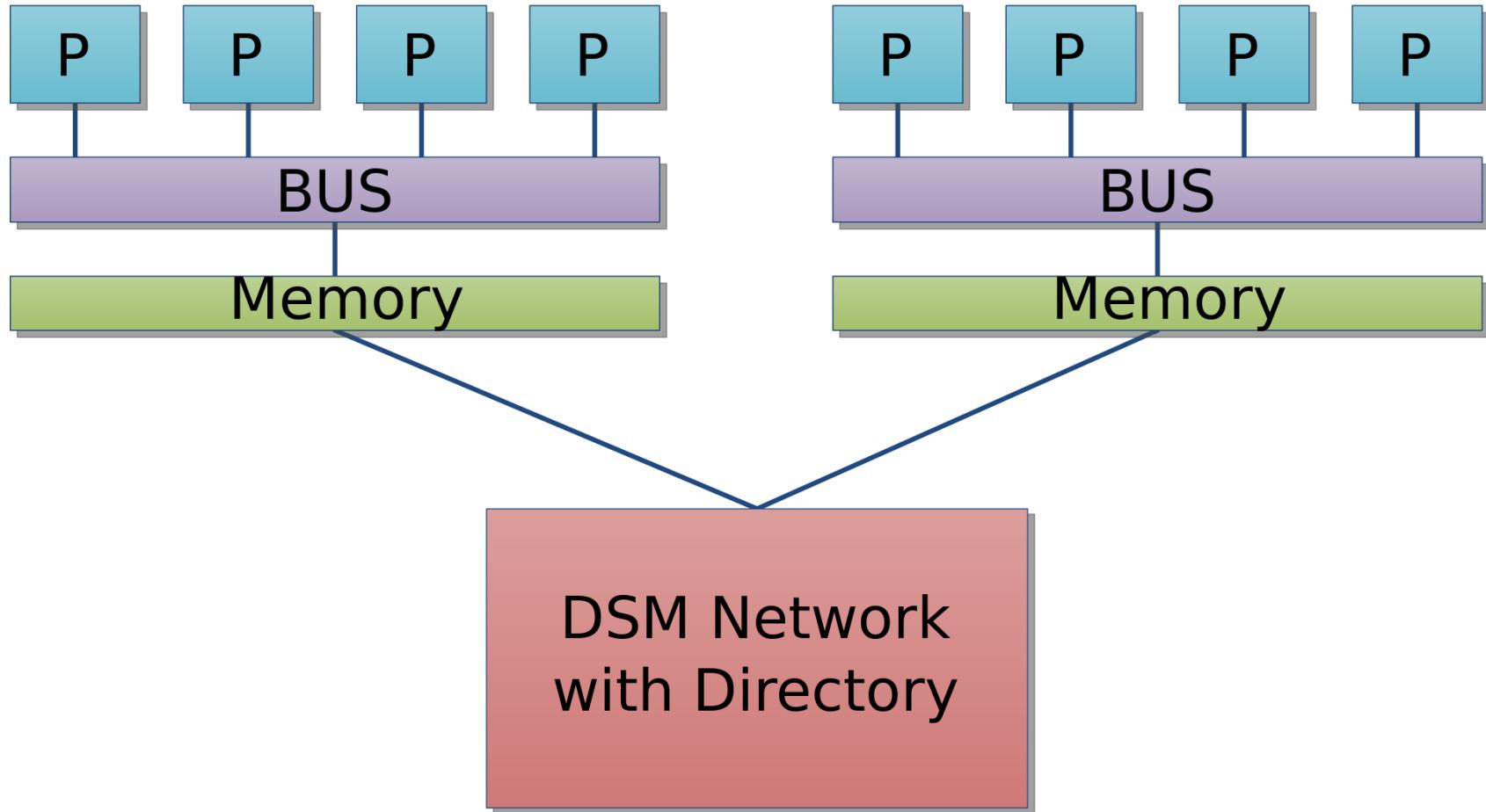
- **UMA (Uniform Memory Access)**
- **procese diferite pe procesoare diferite, dar e nevoie de modificarea programelor ca acestea să ruleze paralel**
- **dezavantaj: cache coherence**

# ASYMMETRIC MULTIPROCESS SYSTEMS



- fiecare procesor are se ocupă de ceva diferit
  - unul execute programe
  - altul se ocupă de I/O

# NON-UNIFORM MEMORY ACCESS



- rezolvă problema accesului la memorie de către procesoare multiple (fiecare procesor are memoria/cache-ul său)
- procesoarele așteaptă mai puțin să ajunge datele la ele

# **CE AM FĂCUT ASTĂZI**

- structura multi-core a calculatoarelor
- ierarhizarea memoriei
- beneficiile cache-ului

# DATA VIITOARE ...

- **ne apropiem de final**
  - Cursul 0x0B va fi în prima săptămână de școală din ianuarie
  - Cursul 0x0C va fi în a doua săptămână de școală din ianuarie
    - nu facem curs, avem verificarea
    - am să vă posteze detalii cu privire la testul de verificare
    - online pe moodle, tot anul
  - în ianuarie, nu se mai fac laboratoare/seminare noi
    - la laborator fiecare grupă are verificarea de laborator
    - la seminar facem recuperări, vă răspund la întrebări etc.
- **Cursul 0x0B**
  - performanța calculatoarelor
  - un demo interesant (sper)

# LECTURĂ SUPLIMENTARĂ

- PH book
  - 5 Large and Fast: Exploiting Memory Hierarchy
  - 6.6 Introduction to Graphics Processing Units
- Erik Demaine, Cache-Oblivious Algorithms: Medians & Matrices,  
<https://www.youtube.com/watch?v=CSqbjfCCLrU>

