

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
 - pagină web curs: <https://cs.unibuc.ro/~crusu/asc/index.html>
- **Bogdan Macovei, Ruxandra Bălucea și Silviu Stăncioiu**
 - laborator
 - contact
 - bogdan.macovei.fmi@gmail.com
 - ruxandra.balucea@unibuc.ro
 - silviu.stancioiu@unibuc.ro

[illegible]

ORGANIZARE ȘI EVALUARE

- **organizare:**

- 2h curs / săptămână
- 2h seminar / 2 săptămâni
- 2h laborator / 2 săptămâni

- **evaluare:**

- 60% examen (fizic, cu materiale pe masă – fără laptop)
 - doar examenul final se repetă la restanță
- 40% laborator
 - 4% test la începutul laboratorului 3
 - 16% temă (predare în decembrie/ianuarie)
 - 20% evaluare finală la laborator (ultimul laborator)
- 10% extra
 - proiecte (în grup) discutate la curs

- **condiții de promovare:**

- peste 50% la laborator pentru temă și evaluare, separat
 - **atenție**, dacă acest punctaj nu este îndeplinit laboratorul trebuie refăcut în următorul an universitar
- peste 50% la examenul final

ORGANIZARE ȘI EVALUARE

- **pentru curs/seminar/laborator**
 - fiți pregatiti 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 (sintaxa AT&T)
 - 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 important 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)

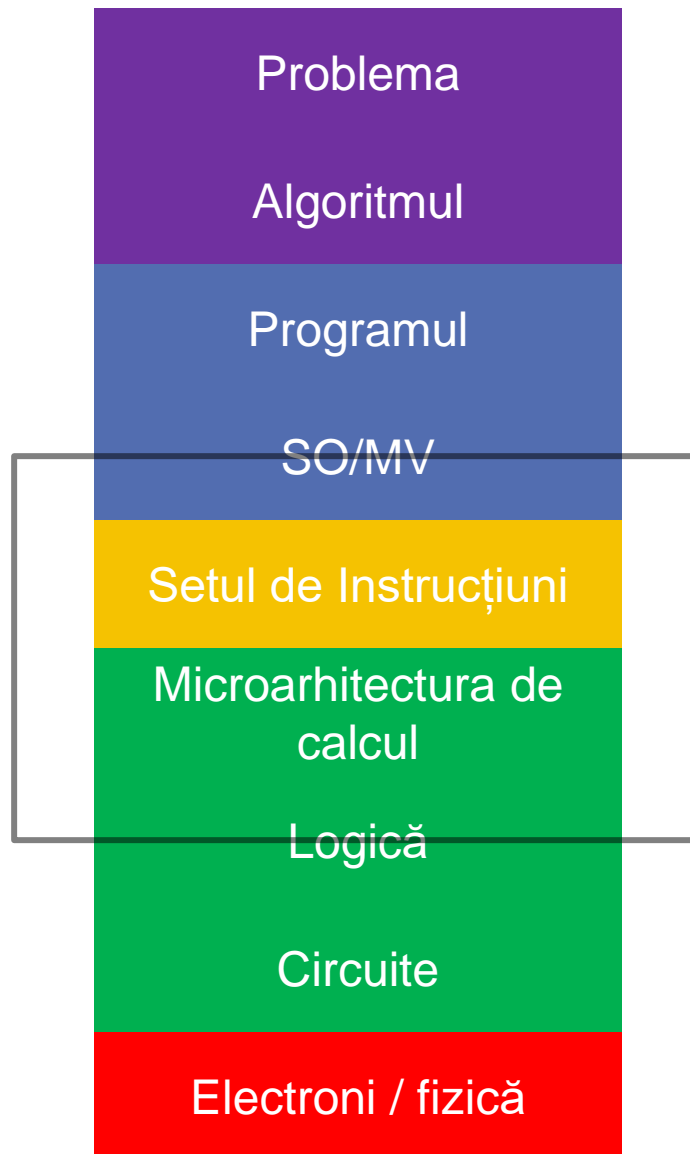
STRUCTURA CURSULUI

- **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**

STRUCTURA CURSULUI

- **laborator: programare în limbajul Assembly x86**
- **nimeni (foarte puțină lume) programează doar/complet în Assembly**
- **Assembly x86 este folosit pentru:**
 - Securitate informatică
 - Reverse Engineering
 - Hacking
 - Optimizare
 - Dezvoltare jocuri
 - Machine Learning (ML/AI)
 - Debugging
 - Dezvoltare software *low-level*
 - Dezvoltare pentru sisteme embedded
 - Dezvoltare pentru sisteme de operare

POZIȚIONAREA CURSULUI



OBIECTIVELE CURSULUI

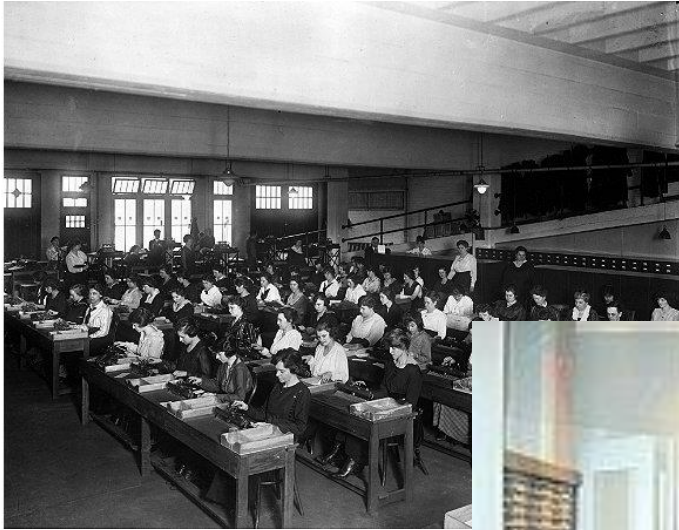
- **să înțelegeți principile arhitecturii sistemelor de calcul**
 - ce urmărim
 - ce limitări există
 - punerea în balanță a unor criterii de performanță contradictorii
- **să înțelegeți ce există ca tehnologie acum**
 - ce limitări există astăzi
 - ce execută un computer pe care îl programați voi
 - 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 execută un computer pe care îl programați 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.)

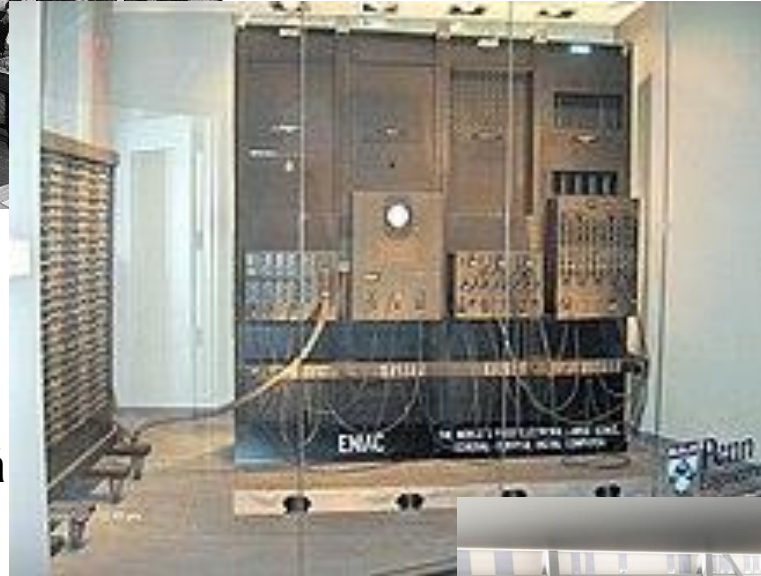
“The purpose of computing is insight, not numbers.” (Richard Hamming)

PROGRESUL TEHNOLOGIC FĂCUT



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

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



- HPE CRAY, 2021, SUA
- core-uri: 8.699.904
- 1679 peta 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 algoritmi)



- **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 algoritmi)



- **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 așa 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 așa 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];
```

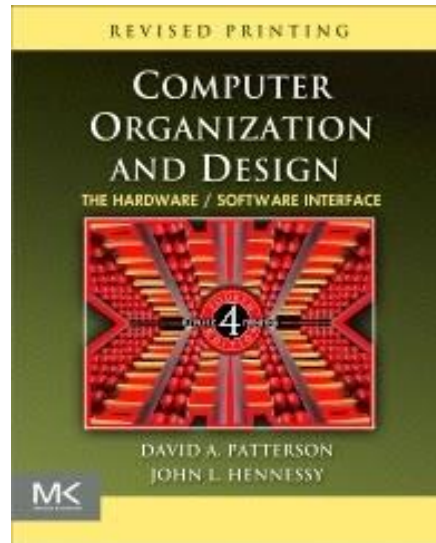
- înmulțește două matrice
- care este complexitatea numerică?
 - $O(n^3)$
 - $2n^3$ operații
- există un algoritm care calculează rezultatul C în $O(n^{2.8074})$
 - din păcate, pe arhitecturile de calcul moderne, acest algoritm este mai lent decât metoda clasică (o formă de varianta A)

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**)
- R. Blum, Professional Assembly Language (**ASM 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/>

