

COMPUTER SYSTEMS ARCHITECTURE LECTURE 0x00

ADMINISTRATIVE INFORMATION

Cristian Rusu, Bogdan Macovei

CONTENTS

- **teaching staff**
- **organization**
- **assessment**
- **course structure**
- **course objectives**
- **general references**

TEACHING STAFF

- **Cristian Rusu**

- lecture and seminar
- contact: cristian.rusu@unibuc.ro
- course webpage: https://cs.unibuc.ro/~crusu/asc_en

- **Bogdan Macovei**

- lecture and seminar
- contact: bogdan.macovei.fmi@gmail.com

- **lab team**

- contact
 - Ruxandra Bălucea ruxandra.balucea@unibuc.ro
 - Mihail Dănuț Dogaru mihail-danut.dogaru@s.unibuc.ro
 - Ioan Ionescu ioan.ionescu@s.unibuc.ro
 - Maria Preda maria.preda@s.unibuc.ro
 - Teodora-Ioana Nae teodora-ioana.nae@s.unibuc.ro

ORGANIZATION AND ASSESSMENT

- **organization:**

- 2h lecture / week
- 2h lab / week

3. Timpul total estimat (ore pe semestru al activităților didactice)

3.1.	Număr de ore pe săptămână	4	din care:	Curs	2	Seminar	1	Laborator	1	Proiect	-
3.2.	Total ore pe semestru	56	din care:	Curs	28	Seminar	14	Laborator	14	Proiect	-
3.3 Distribuția fondului de timp											ore
3.3.1. Studiul după manual, suport de curs, bibliografie și notițe – nr. ore SI											14
3.3.2. Documentare suplimentară pe platformele electronice de specialitate											14
3.3.3. Pregătire seminare/ laboratoare/ proiecte											14
3.3.4.Examinări											2
3.3.5. Alte activități											0
3.4.	Total ore studiu individual (3.3.1 + ... + 3.3.5)	44									
3.5.	Total ore pe semestru (3.2 + 3.4)	100									
3.6.	Numărul de credite	4									

ORGANIZATION AND ASSESSMENT

- **organization:**
 - 2h lecture / week
 - 2h lab / week
- **assessment:**
 - 60% exam (in person, with materials on the desk – no laptop)
 - only the final exam is retaken in the resit session
 - 40% lab
 - 10% lab test (in November)
 - 10% lab activity (throughout the semester; this is not attendance)
 - 20% final lab assessment (last lab or lecture)
 - 10% extra
 - projects (in groups) discussed during the course
- **passing requirements:**
 - over 50% in the lab assessment and overall across the three lab criteria
 - note: if this score is not achieved, the lab must be repeated in the following academic year (the same applies to the exam)
 - over 50% on the final exam

ORGANIZATION AND ASSESSMENT

- **for the lecture/lab**
 - be prepared with paper/pen to write down fundamental ideas and exercises (you also have the materials electronically, but some problems will be solved together both in the lecture and in the seminar)
 - you need a laptop/computer for the lab
- **for the exam: everything presented in the lecture/lab is included (except for slides/concepts that are explicitly mentioned)**

ORGANIZATION AND ASSESSMENT

- **for the lab**
 - there will be exercises using the notions introduced in the lecture
 - a lot of programming
 - x86 assembly (AT&T syntax)
 - assembly RISC-V
 - you will also see Linux/Git/C language
 - there is a tutorial for the required installation
 - there is a C tutorial
 - there are two parts: 5 labs each
- **after about one month, you will be able to assess the difficulty of the course**

ORGANIZATION AND ASSESSMENT

- you are no longer in high school
- we no longer repeat the same type of problems/exercises
- the lecture
 - the focus is on concepts
 - the focus is on understanding what is happening
 - the idea is the most important thing
- the lab
 - exercises using the concepts from the lecture
 - we consolidate what we saw in the lecture
 - this involves a lot of individual work
 - *we encourage you to work together and collaborate in order to understand, but not to copy*

DO NOT COPY/PLAGIARIZE

- **the penalty for copying is not just failing the course**
- **you will be reported to the ethics committee and risk expulsion**
- **no copy/paste in seminar/lab/homework/test/exam**
- **no copying from colleagues (all of you are at risk)**

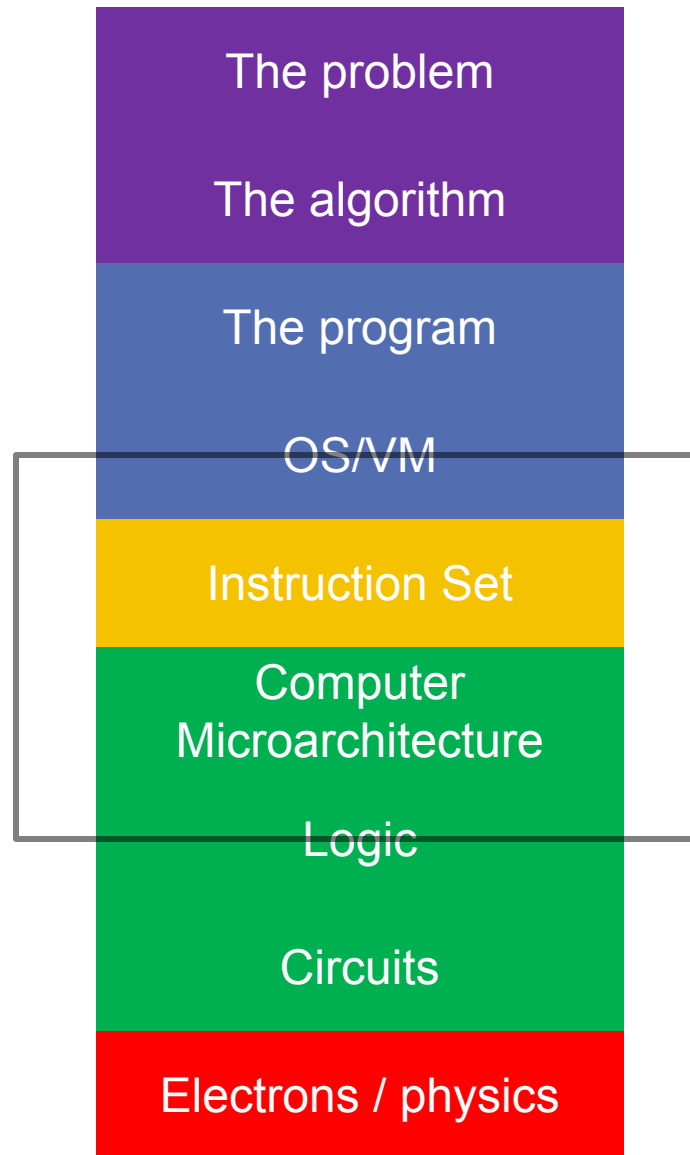
COURSE STRUCTURE

- **digital circuits**
 - information theory and digital abstraction
 - logic functions and circuits
- **computer architectures** ← [the material covered in the lab starts from here](#)
 - instruction sets
 - assembly language
 - compilers
 - pipelining
 - memory hierarchy
- **computer organization**
 - central processing unit
 - computer performance
 - peripheral devices and interrupts
 - parallel computing
- **potential modern lecture topics: WebAssembly, TockOS, hardware for machine learning (GPU, TPU, etc.), etc.**
- **lab: programming in x86 Assembly and RISC-V Assembly**

COURSE STRUCTURE

- **lab: x86 Assembly and RISC-V Assembly**
- **nobody (very few people) programs only/entirely in Assembly**
- **x86 assembly is useful for:**
 - computer security
 - Reverse Engineering (RE)
 - hacking
 - optimization
 - game development
 - Machine Learning (ML/AI)
 - debugging
 - low-level software development
 - embedded systems development
 - operating systems development

COURSE POSITIONING



COURSE OBJECTIVES

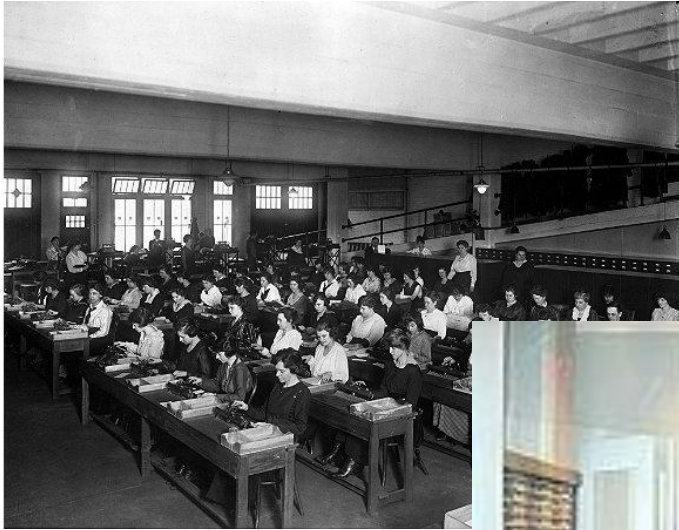
- **to understand the principles of computer systems architecture**
 - what we aim for
 - what limitations exist
 - balancing conflicting performance criteria
- **to understand what technology exists today**
 - what limitations exist today
 - what a computer that you program actually executes
 - how you can optimize program execution
- **to understand how to use what you have learned in the future**
 - design of new hardware
- **to make sure that things are going well, you will have the opportunity to provide feedback (anonymously, of course) halfway through the course**
- **during the semester, if there are problems or unclear points, please let me know (by e-mail, in class, either directly or anonymously, or through a representative, etc.)**

COURSE OBJECTIVES

- **to understand the principles of computer systems architecture**
 - what we aim for
 - what limitations exist
 - balancing conflicting performance criteria
- **to understand what technology exists today**
 - what limitations exist today
 - what a computer that you program actually executes
 - how you can optimize program execution
- **to understand how to use what you have learned in the future**
 - design of new hardware
- **to make sure that things are going well, you will have the opportunity to provide feedback (anonymously, of course) halfway through the course**
- **during the semester, if there are problems or unclear points, please let me know (by e-mail, in class, either directly or anonymously, or through a representative, etc.)**

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

TECHNOLOGICAL PROGRESS



- before World War II, 1939, USA
- computing power: 50 operations / second

- ENIAC
- 1947 – 1955, USA
- 1000 operations / second



- EL CAPITAN - HPE CRAY, 2025, USA
- cores: 11,039,616
- 2,746 peta operations / second



<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>

TODAY'S REQUIREMENTS (EXAMPLE)

- **hardware for machine learning**
- **DeepMind and OpenAI talk about two things:**
 - new algorithms for modeling and training
 - “compute” (the hardware infrastructure on which the algorithms run)



- **AlphaZero: ML-based software**
 - Stockfish (classic software based on search methods – mainly alpha/beta pruning)
 - 3500 ELO
 - AlphaZero vs Stockfish:
 - $+155 -6 = 839$
 - Magnus Carlsen (current world champion)
 - < 2900 ELO
 - AlphaZero (white)
vs Stockfish (black)



TODAY'S REQUIREMENTS (EXAMPLE)

- hardware for machine learning
- DeepMind and OpenAI talk about two things:
 - new algorithms for modeling and training
 - “compute” (the hardware infrastructure on which the algorithms run)



- **AlphaZero: ML-based software**

- Stockfish (classic software based on search methods – mainly alpha/beta pruning)
 - 3500 ELO
- DeepMind says it trained this algorithm for 4 hours (learned through self-play)
 - yes, 4 hours on their computing platform
 - a quick approximate calculation shows that the same training procedure would take 30 years on my laptop
 - the cost? approximately \$20 million
- how do we reduce the cost?
 - more efficient algorithms

specialized, dedicated hardware

TODAY'S REQUIREMENTS (EXAMPLE)

- CSA is also important for everyday software
- what does the following algorithm do?

```
# variante 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];
```

TODAY'S REQUIREMENTS (EXAMPLE)

- CSA is also important for everyday software
- what does the following algorithm do?

```
# variante 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];
```

- it multiplies two matrices
- what is the numerical complexity?

TODAY'S REQUIREMENTS (EXAMPLE)

- CSA is also important for everyday software
- what does the following algorithm do?

```
# variante 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];
```

```
# variante 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];
```

- it multiplies two matrices
- what is the numerical complexity?
 - $O(n^3)$
 - $2n^3$ operations
- what does version B do?

TODAY'S REQUIREMENTS (EXAMPLE)

- CSA is also important for everyday software
- what does the following algorithm do?

```
# variante 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];
```

```
# variante 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];
```

- it multiplies two matrices
- what is the numerical complexity?
 - $O(n^3)$
 - $2n^3$ operations
- what does version B do?
- the two algorithms are mathematically equivalent

TODAY'S REQUIREMENTS (EXAMPLE)

- CSA is also important for everyday software
- what does the following algorithm do?

```
# variante 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];
```

```
# variante 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];
```

- it multiplies two matrices
- what is the numerical complexity?
 - $O(n^3)$
 - $2n^3$ operations
- what does version B do?
- the two algorithms are mathematically equivalent

which algorithm runs faster?

TODAY'S REQUIREMENTS (EXAMPLE)

- CSA is also important for everyday software
- what does the following algorithm do?

```
# variante 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];
```

```
# variante 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];
```

- it multiplies two matrices
- what is the numerical complexity?
 - $O(n^3)$
 - $2n^3$ operations
- what does version B do?
- the two algorithms are mathematically equivalent

on my computer system, version B is 4 times slower than version A

how is this possible?

what is the difference between the two versions?

TODAY'S REQUIREMENTS (EXAMPLE)

- CSA is also important for everyday software
- what does the following algorithm do?

```
# variante 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];
```

```
# variante 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];
```

- it multiplies two matrices
- what is the numerical complexity?
 - $O(n^3)$
 - $2n^3$ operations
- what does version B do?
- the two algorithms are mathematically equivalent

on my computer system, version B is 4 times slower than version A
how is this possible?

what is the difference between the two versions?

the order in which instructions are executed is very important

TODAY'S REQUIREMENTS (EXAMPLE)

- CSA is also important for everyday software
- what does the following algorithm do?

```
# variante 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];
```

```
# variante 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];
```

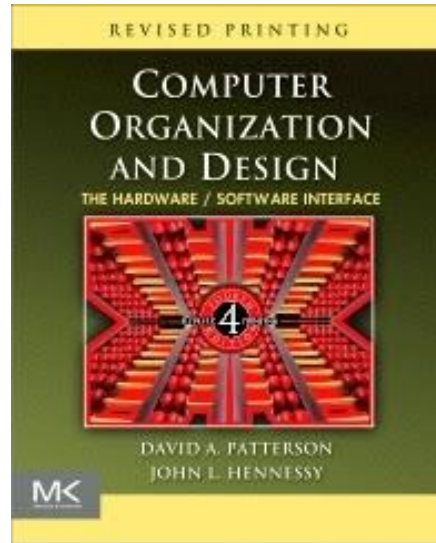
conclusions:

- numerical complexity is very important, but it is not everything
- what we compute mathematically and what we can execute are two different things
- sometimes progress in the real world is surprising and completely non-obvious

- it multiplies two matrices
- what is the numerical complexity?
 - $O(n^3)$
 - $2n^3$ operations
- there is an algorithm that computes the result C in $O(n^{2.8074})$
 - unfortunately, on modern computer architectures, this algorithm is slower than the classical method (a form of version A)

GENERAL REFERENCES

- 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/>
- these slides are also based on other CSA courses at UB (R. Olimid and D. Dragulici)

At the end of each lecture, you will have a specific list of references for that lecture's content

A DEMO FOR THE END

- what can we do if we understand computer systems architecture / assembly / operating systems / web technologies?
- Doom 3 running in the browser (WebAssembly demo):
<http://wasm.continuation-labs.com/d3demo/>

