# Computer Systems, B1-2 2025-26

# Introduction

David Marchant

DIKU, September 1, 2025

# Hello

- David on the Discord, david.marchant@di.ku.dk, 772-01-0-s06
- Mit dankse er ikke godt
- I will make an appearance at exercise classes and assignment cafes
- I run Linux, not Mac, not Windows
- Questions and comments are always welcome

# Lecturers

 Michael Kirkedal Thomsen: Course root, Networks and Security

 Finn Schirmer Andersen: Computer Architecture

 David Gray Marchant: C programming, Operating Systems and Network programming

# TAs

TAs:

- Carl August Gjerris Hartmann
- Emil Viggo Dalsgaard
- Mads Presfeldt
- Malte Emil Wechter
- Philip Shun Buenaventura Jensen
- Tobias Andersen

TAs will gladly help with

- Group members
- The right way to the administration
- A fellow student that can answer questions (or help find the answers)

# Who to contact

- Your TA - Should be able to at least guide you in the right direction about anything
- Me/Finn - We are responsible for most of the course material and assignments
- Michael - Course admin and exam qualification

# Overall outline

| | |
|---|---|
| Week 36-37 | Data representation and machine model |
| Week 38 | C programming |
| Week 39-40 | Memory and operating systems |
| Week 41 | Concurrent and parallel programming |
| Week 42 | Fall break |
| Week 43-45 | Computer networks - application and transport layer |
| Week 46 | No activities (reexam week) |
| Week 47 | Computer networks - security and efficiency |
| Week 48-51 | Machine architecture |
| Week 52 | Christmas vacation |
| Week 1-2 | Computer networks and security - network and link layer |
| Week 4 | 4-hour written exam |

# Teaching Material

- **COD** Computer Organization and Design (RISC-V Edition), David A. Patterson and John L. Hennesy, second edition, ISBN: 978-0-12-820331-6
- **KR** Computer Networking: A Top-Down Approach, James F. Kurose and Keith W. Ross, Pearson, 8th and Global Edition, ISBN 13: 978-1-292-40546-0 (This book will not be used before December)–7th edition is also acceptable
- **JG** Modern C, Jens Gustedt, `https://hal.inria.fr/hal-02383654/document`
- **OSTEP** Operating Systems: Three Easy Pieces, Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, `https://pages.cs.wisc.edu/~remzi/OSTEP/`
- **??** Some notes and book chapters that will be made available through the detailed course schedule

COD is (and KR will be) available at Academic Books at Panum (`http://www.academicbooks.dk/`) and Polyteknisk Boghandel at Biocenteret (`http://www.polyteknisk.dk/`).

# How to Use This Course

- Videos
  - Released the week before
  - Introduces the topic of the week
- Monday: Exercise class
  - Will assume you've seen the videos
  - First shot at playing around with the topics in practice
- Monday: Review lecture
  - Review videos, answer questions, go over material as more traditional lecture
  - Won't add anything not covered in videos or exercise. Attendance optional
- Wednesday: Exercise class
  - More practical practice
- Wednesday: Advanced lecture
  - Additional material not covered by videos.

# Lectures

- Review Lecture:
  - Mondays 15:15-17:00
  - Reviews the material already covered in the videos
  - Intended for those with questions from the material or exercises
  - Or those who would just rather learn from a lecture
  - If you're already ahead, this can be skipped
- Advanced Lecture:
  - Wednesdays 15:15-17:00
  - Additional course material not covered in the videos
  - Questions are also welcome here as always, but not the focus
  - Attendance is expected at these

# Exercises and Assignment Cafés

## Exercises

- Mondays 13:15-15:00
- Wednesdays 13:15-15:00

Exercises are only for posted exercises. Work on the exercises as they will prepare you both for the exam and assignments.

## Cafés

- Wednesdays 10:15-12:00

Cafés are primarily for help with assignments.

Details: `https://github.com/diku-compSys/compSys-e2025`. Also on Discord. See Absalon/Modules.

# Assignments

- There are 6 assignment in total during the course with deadlines every two or three weeks (all Sundays). The assignments will be evaluated with points.
- Assignments will be awarded varying points (typically 4-6).
- You are required to achieve at least 50 % of the total number of points (equal to 12).
- Also we will require that you achieve points in each the of topics of the course to ensure that you have touched all parts of the curriculum.
- Assignments are made to be solved in groups of 2-3 students, but you can also do them alone.
- If you are resitting the course you do not need to resubmit assignments you've already completed, but practice is always best

# Assignment rules

## The Fundamental Principle of Group Assignments

Each group must make their own solution.

This means

- You can talk with other people about the assignments: Teachers, TAs, other students, etc.
- You cannot share written code with other groups.
- You are not allowed to use code that you did not write yourself without proper citation.
- You cannot share written text with other groups.
- You are not allowed to use text of material without proper citation
  - This also includes material provided on the course.
- You are only allowed to use ChatGPT and other AI as any other person.
- *MUST* include a KU Generative AI Declaration

# Assignments vs. exercises

- Note! Both are equally important
- Assignments:
  - Seek to test learning goals that relates to implementation and development of computer systems.
  - Do not fully prepare you for the written exam.
- Exercises:
  - Help you understand the theoretical parts of the material.
  - Prepare you for part of the exam.

# Groups

## Size

2-3 student advised. 1 can be accepted but not recommended. More than 3 is only allowed is in special circumstances

- Sign up for classes with your group-mates on Absalon
- If you need one or more members
  - Look on announcements for details (use Absalon and Discord)
  - TA can help
- You can change groups at any time, but be reasonable to your colleagues

# Tools

- RARS - RISC-V simulator
- C compiler – gcc (clang on macOS)
- C debugger – gdb (lldb on macOS)
- Profiler – Valgrind (perhaps leaks on macOS)
- You can also install all tools on you laptop
  - Linux: most available though apt
  - macOS: most available though Homebrew
  - Windows: Windows Subsystem for Linux

- Set up your tool chain
  - recommended using git to share code and reports in your group
  - Sign-up at GitHub today and apply for the *Student Developer Pack*
  - `https://education.github.com/`

- Tool-site is available on GitHub

# LLMs

- You are allowed to use them
- But we'd prefer you didn't for the programming tasks
- I won't be using them for this course, either to make or mark material
- Most of the programming tasks we're going to ask of you are not things you'd have to do in your regular work
- But the point is to do them to really learn what is important and why
- Learning these concepts are what makes you a Computer Scientist
- Remember the KU Generative AI Declaration

# CompSys Masterclass

- September 10th, 10.15-12.00
- DIKU 1-0-30, 1-0-34, and 1-0-37
- Introductions to many of the ancillary tools that will make life easier
  - Git
  - Unix
  - GDB
- Has been very helpful in previous years...

# Exam

- A 4-hour written exam; Jan 21st 2025.
- The exam will be a BYOD-exam.
- The course syllabus is the exercises, assignments and reading material/videos.
- Previous exams will be available.

# Registers

- I've been researching different methodologies for teaching and their effects on final grades
- But to present my findings I need actual data
- So I'll be collecting registers are lectures and exercise classes
- For research only! Will not affect your grades or whatnot, in fact probably won't be looked at till after the course is done
- Any research will only be in aggregate, e.g. everyone attending week 4 got a 12 on the exam
- Please be honest.

## Course structure

Course Content

## Representing information as bits

Everything is bits

Bit-level manipulation

Integers

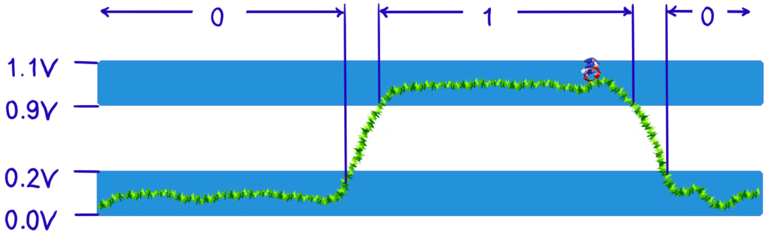
# Everything is bits

- Each bit is 0 or 1.
- By interpreting sets of bits in various ways...
  - ...computers determine what to do.
  - ...represent and manipulate numbers, sets, strings—*data*.

**Why bits? Why not decimals? Could it have been some some other way?**

# Everything is bits

- **Why bits? Electronic implementation.**
  - Easy to store with bistable elements.
  - Reliably transmitted on noisy and inaccurate wires (error correction).




- **... But there exist models that do not use bits.**
  - The Soviet Setun computer used ternary *trits*.
  - Quantum computers use *qubits* that are in a superposition of the two states.
    - ...error correction is the main challenge here.

# Binary numbers

- **Base 2 number representation.**
  - Represent $15213_{10}$ as $11101101101101_2$
  - Represent $1.20_{10}$ as $1.0011001100110011[0011]\ldots_2$
  - Represent $1.5213 \times 10^4$ as $1.1101101101101_2 \times 2^{13}$
- **Machine numbers are of some finite size.**
  - If we use $k$ bits to represent a number, only $2^k$ distinct values are possible.
  - How we interpret those bits can vary.
  - **Why do we use finite-sized numbers?**

# Binary numbers

- **Base 2 number representation.**
  - Represent $15213_{10}$ as $11101101101101_2$
  - Represent $1.20_{10}$ as $1.0011001100110011[0011]\ldots_2$
  - Represent $1.5213 \times 10^4$ as $1.1101101101101_2 \times 2^{13}$
- **Machine numbers are of some finite size.**
  - If we use $k$ bits to represent a number, only $2^k$ distinct values are possible.
  - How we interpret those bits can vary.
  - **Why do we use finite-sized numbers?**
  - A "$k$-bit machine" handles numbers of up to $k$ bits "natively" (meaning fast).

# Encoding byte values

**Byte = 8 bits**

- (Machine-specific, but is true for all mainstream machines.)
- 256 different values.
- Binary $00000000_2$ to $11111111_2$.
- Decimal $0_{10}$ to $255_{10}$.
- Hexadecimal $00_{16}$ to $FF_{16}$.
  - Base 16 number representation.
  - Uses characters 0—9 and A—F.
  - In C we write $FA1D37B_{16}$ as
    - `0xFA1D37B`
    - `0xfa1d37b` (case does not matter)

| Hex | Dec | Bin |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Example sizes of C types on various computers

| C data type | Typical 16-bit | Typical 32-bit | Typical 64-bit | x86-64 |
|---|---|---|---|---|
| `char` | 1 | 1 | 1 | 1 |
| `short` | 1 | 2 | 2 | 2 |
| `int` | 2 | 4 | 4 | 4 |
| `long` | 4 | 4 | 8 | 8 |
| `int32_t` | 4 | 4 | 4 | 4 |
| `int64_t` | 8 | 8 | 8 | 8 |
| `float` | 4 | 4 | 4 | 4 |
| `double` | 8 | 8 | 8 | 8 |
| `long double` | - | - | - | 10 |
| `pointer` | 2 | 4 | 8 | 8 |

# Boolean algebra

**Developed by George Boole in 19th century**

- Algebraic representation of logic (“truth values”).
- Encode *true* as 1 and *false* as 0.

**And**

| & | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

**Or**

| \| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

**Not**

| ~ | |
|---|---|
| 0 | 1 |
| 1 | 0 |

**Exclusive-or**

| ^ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

- These operations can be implemented with tiny electronic *gates*.

# General boolean algebras

- The truth tables generalise to operate on *bit vectors*, applied elementwise.

```
   01101001      01101001      01101001
 & 01010101    | 01010101    ^ 01010101    ~ 01101001
 ----------    ----------    ----------    ----------
   01000001      01111101      00111100      10010110
```

- This is the form they take in programming languages such as C.

# Bit-level operations in most C-like languages

## Operations &, |, ~, ^ available in C.

- Apply to any integral type.
  - E.g. `long`, `int`, `short`, `char`...
- Interpret operands as bit vectors.
- Applied bit-wise.

## Examples

- `~0x41 = 0xBE`
  - $\sim 01000001_2 = 10111110_2$
- `~0x00 = 0xFF`
  - $\sim 00000000_2 = 11111111_2$
- `0x69 & 0x55 = 0x41`
  - $01101001_2 \; \& \; 01010101_2 = 01000001_2$
- `0x69 & 0x55 = 0x7D`
  - $01101001_2 \; \& \; 01010101_2 = 01111101_2$

# Shift operations

- **Left shift `x << y`**
  - Shift bit-vector `x` left by `y` positions.
    - Throws away excess bits on the left.
    - Fills with zeroes on right.
- **Right shift `x >> y`**
  - Shift bit-vector `x` right by `y` positions.
    - Throws away excess bits on the left.
  - Logical shift: Fill with 0s on left.
  - Arithmetic shift: Replicate most significant bit on left.
- **Undefined behaviour**
  - Shifting a negative amount or by the vector size or more.

| | |
|---|---|
| `x` | `01100010` |
| `x << 3` | `00010000` |
| `x >> 2 (log)` | `00011000` |
| `x >> 2 (arith)` | `00011000` |
| `x` | `10100010` |
| `x << 3` | `00010000` |
| `x >> 2 (log)` | `00101000` |
| `x >> 2 (arith)` | `11101000` |

# Encoding integers

Suppose $x_i$ is the $i$th bit of a $w$-bit word (with $x_0$ being the least significant bit).

**Unsigned**

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

**Two's complement**

$$B2S(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

```
int16_t x =  15213;
int16_t y = -15213;
```

| | Decimal | Hex | Binary |
|---|---|---|---|
| x | 15213 | 3 B 5 D | 0011 1011 0110 1101 |
| y | -15213 | C 4 9 3 | 1100 0100 1001 0011 |

**Sign bit**

- For 2's complement, most significant bit ($x_{w-1}$) indicates sign.
  - 0 for non-negative.
  - 1 for negative.

# Two's complement encoding example

```
int16_t x =  15213; // 0011 1011 0110 1101
int16_t y = -15213; // 1100 0100 1001 0011
```

| Weight | 15213 | | -15213 | |
|---:|:--|--:|:--|--:|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 2 |
| 4 | 1 | 4 | 0 | 0 |
| 8 | 1 | 8 | 0 | 0 |
| 16 | 0 | 0 | 1 | 16 |
| 32 | 1 | 32 | 0 | 0 |
| 64 | 1 | 64 | 0 | 0 |
| 128 | 0 | 0 | 1 | 128 |
| 256 | 1 | 256 | 0 | 0 |
| 512 | 1 | 512 | 0 | 0 |
| 1024 | 0 | 0 | 1 | 1024 |
| 2048 | 1 | 2047 | 0 | 0 |
| 4096 | 1 | 4096 | 0 | 0 |
| 8192 | 1 | 8192 | 0 | 0 |
| 16384 | 0 | 0 | 1 | 16384 |
| -32768 | 0 | 0 | 1 | -32768 |
| **Sum** | | **15213** | | **-15213** |

# Let's play a game

http://topps.diku.dk/compsys/integers.html

# Numeric ranges, here for 16-bit signed and unsigned integers

**Unsigned**

$$\text{UMin} = 0 = 0\ldots0_2$$
$$\text{UMax} = 2^w - 1 = 1\ldots1_2$$

**Two's complement signed**

$$\text{SMin} = -2^{w-1} = 10\ldots0_2$$
$$\text{SMax} = 2^{w-1} - 1 = 01\ldots1_2$$
$$-1 = 1\ldots1_2$$

**Values for** $w = 16$:

| | Decimal | Hex | Binary |
|---|---|---|---|
| **UMax** | 65535 | `F F F F` | `1111 1111 1111 1111` |
| **SMax** | 32767 | `7 F F F` | `0111 1111 1111 1111` |
| **SMin** | -32768 | `8 0 0 0` | `1000 0000 0000 0000` |
| -1 | -1 | `F F F F` | `1111 1111 1111 1111` |
| 0 | 0 | `0 0 0 0` | `0000 0000 0000 0000` |

# Values for different word sizes

| | W | | | |
|---|---|---|---|---|
| | **8** | **16** | **32** | **64** |
| **UMax** | 255 | 65,535 | 4,294,967,295 | 18,446,744,073,709,551,615 |
| **SMax** | 127 | 32,767 | 2,147,483,647 | 9,223,372,036,854,775,807 |
| **SMin** | -128 | -32,768 | -2,147,483,648 | -9,223,372,036,854,775,808 |

## Observations

$$|\mathrm{SMin}| = \mathrm{SMax} + 1$$
$$|\mathrm{UMax}| = 2 \cdot \mathrm{SMax} + 1$$

**Note the assymetric range.**

## C Programming

- `#include <limits.h>`
- Declares constants, e.g:
  - `ULONG_MAX`
  - `LONG_MAX`
  - `LONG_MIN`
- Values are platform-specific.

# Unsigned and signed numeric values (here $w = 4$)

| $x$ | $B2U(x)$ | $B2S(x)$ |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0010 | 2 | 2 |
| 0011 | 3 | 3 |
| 0100 | 4 | 4 |
| 0101 | 5 | 5 |
| 0110 | 6 | 6 |
| 0111 | 7 | 7 |
| 1000 | 8 | -8 |
| 1001 | 9 | -7 |
| 1010 | 10 | -6 |
| 1011 | 11 | -5 |
| 1100 | 12 | -4 |
| 1101 | 13 | -3 |
| 1110 | 14 | -2 |
| 1111 | 15 | -1 |

- **Equivalence**
  - Same encoding for non-negative values.
- **Uniqueness**
  - Every bit pattern represents distinct integer value.
  - Each representable integer has unique bit encoding.
  - The representation is **bijective**.
- **Can invert mappings**
  - $U2B(x) = B2U^{-1}(x)$
    - Bit pattern for unsigned integer.
  - $S2B(x) = B2S^{-1}(x)$
    - Bit pattern for two's complement integer.

# Main takeaways

- Distinguish between **representation** and **interpretation**.
- Low-level values **do not describe their own structure**.
- Everything is built in layers.
- A good computer scientist adds new, clean, layers of abstraction.
  - A bad one adds layers that hide without simplifying.
  - A terrible one adds layers that complicate and obfuscate.
- **The point of this course is to show that there is no magic, only the work of careful people who put in a lot of effort.**