01: Course formalities, introduction to microcontrollers, processor core

Microcontrollers

Stefan Huber
www.sthu.org

Dept. for Information Technologies and Digitalisation
FH Salzburg

Winter 2024

$ whoami

- Since 2019 professor at FH Salzburg, since 2022 head of Josef Ressel Center for Intelligent and Secure Industrial Automation (JRC ISIA), since 2023 head of research of this department.
- Research focus on algorithms, machine learning and industrial automation.
  $\approx$ 70 publications (1 book, 12 patents), 2 commercial software packages, $\approx$ 1000 citations.
- PostDoc at IST Austria, PostDoc and Senior Scientist at Univ. of Salzburg between 2012 and 2015. Team and project lead at B&R Industrial Automation in 2015 to 2019.
- B.Sc. in computer science, B.Sc. in mathematics, M.Sc. in computer science, M.Sc. in mathematics, PhD in computer science in 2006, 2007, 2008, 2009 and 2011, respectively.

$\rightarrow$ `https://www.sthu.org`

JRC ISIA research center

Goal

Make industrial machines intelligent, secure and autonomous.

- 3 M€ research center with B&R industrial automation, COPA-DATA, SIGMATEK
- Artificial intelligence, cybersecurity, industrial system

Course formalities

Course organization and grading

Grading of the lecture is based on an exam at the end.

This lecture is accompanied by a lab (Olaf Saßnick, Christian Feitler).

- Grading of lab will be explained in the lab.

This lecture serves two purposes:

- Laying foundations for lab assignments.
- Forming a larger theoretical body of topics around microcontrollers and, in a broader sense, embedded systems.

Course material and organization

Moodle course for material and non-realtime interaction.

- Lecture notes
- Accompanied code
- Lecture videos
- Supplementary material

Teams team for realtime interaction, like online lectures and chat.

Lecture notes:

- The slide decks are the lecture notes.
- They are considered to be self-contained and complete.[1]

[1] Hence, they are more verbose than "typical" presentations.

Content outline

This course is about microcontrollers and how to program them.

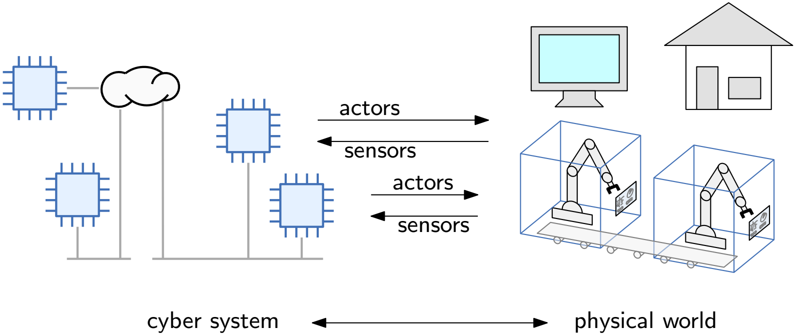
A rough outline of topics:

- Computer and processor architecture
- Design of microcontroller software
- Interfacing: Communication, analog/digital input/output and the like
- Selected topics of embedded and real-time systems

Our ambitious goal

We want to understand the inner workings, from a line of C code to the digital circuitry.

We want to understand and being able to build distributed, embedded, real-time, cyber-physical systems.



Prerequisites

This course builds on your prior knowledge in

- Programming in C — e.g, on bit-manipulation level
- Digital electronics, signal processing — e.g, for processor design, PWM, I/O
- Networking, operating systems — e.g., OSI layers 1 and 2, memory layout, concurrency

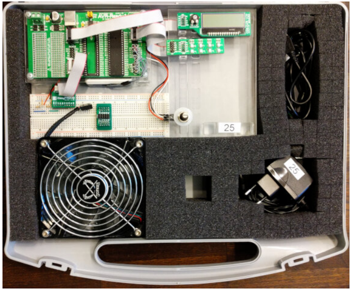
Literature:

- [TB14]: Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. 4th ed. Prentice Hall Press, 2014. ISBN: 013359162X
- [TW11]: Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. 5th ed. Pearson, 2011. ISBN: 9780132126953
- [Koc15]: Stephen G. Kochan. *Programming in C*. 4th ed. Pearson, 2015. ISBN: 9780132781190
- [Kut18]: Rade Kutil. {*C, C++;*}. lecture notes. 2018. URL: https://www.cosy.sbg.ac.at/~rkutil/ccpp18/ccpp.pdf
- [gnu-c]: *The GNU C Reference Manual*. URL: https://www.c-asm.com/gnu-c-manual.html
- [Fri18]: Klaus Fricke. *Digitaltechnik*. 8th ed. Springer Vieweg, 2018. ISBN: 978-3-658-21065-6

Hardware organization

This course is largely aligned to these two hardware platforms:

- An ATmega32 microcontroller evaluation board with peripherals in a box.
  Please treat the equipment with respect.[2]



- Your Raspberry Pi.
  We assume that you know how to establish an SSH connection to your RPi and you can deal with the shell on a basic level.[3]

---

2 For instance, do not leave it in a car at high or low temperatures.

3 See also `https://www.raspberrypi.org/documentation/remote-access/ip-address.md`.

Literature

These lecture slides are considered to be self-contained.

However, we often refer to the ATmega32 data sheet.

- [ATmega32]: *ATmega32: 8-bit AVR Microcontroller with 32KBytes In-System Programmable Flash*. Atmel Corporation. Feb. 2011
- Get a copy from Moodle and work with it. Really. Do.

A beginners book to programming, microcontrollers and the ATmega32:

- [Sch08]: Günter Schmitt. *Mikrocomputertechnik mit Controllern der Atmel-AVR-RISC-Familie*. 4th ed. Oldenbourg, 2008. ISBN: 978348657906

For further reading I recommend:

- [GW07]: Günther Gridling and Bettina Weiss. *Introduction to Microcontrollers*. lecture notes. 2007. URL: https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf
- [Fri18]: Klaus Fricke. *Digitaltechnik*. 8th ed. Springer Vieweg, 2018. ISBN: 978-3-658-21065-6
- [Kop11]: Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. 2nd. Springer Publishing Company, Incorporated, 2011. ISBN: 9781441982360
- [HP12]: John L. Hennessy and David A. Patterson. *Computer Architecture*. 5th ed. Morgan Kaufmann, 2012. ISBN: 978-0-12-383872-8

Goals for this course

Basics:

- You can read and understand data sheets
- You can compare and choose microcontrollers by their features

Advanced:

- You can be a responsible developer for embedded systems
- You gain a deeper understand for computers and system-level software
- You become a more effective software developer, also way beyond embedded systems

This is an advanced course that spans over various foundations formed within this curriculum.

Introduction to microcontrollers

The origin of microcontrollers

The Intel 4004 was the first commercially available microprocessor (1971).

A microprocessor is a computing unit. It applies operations on data and typically comprises

- registers to store operands, a pointer to the current instruction, a pointer to a stack, status information, and so on,
- a arithmetic logic unit[4] (ALU) that performs arithmetic operations, and
- a control unit[5] that essentially coordinates the temporal sequence of operations and flow of data.

For a working computing system, additional hardware is required, in particular

- RAM,
- permanent memory,
- peripherals

[4] Dt. Rechenwerk
[5] Dt. Steuerwerk

The origin of microcontrollers

In 1974, Texas Instruments introduced the TMS 1000, which is seen as the first microcontroller. It included RAM, ROM and I/O on-chip.

A microcontroller is a microprocessor plus (typically)

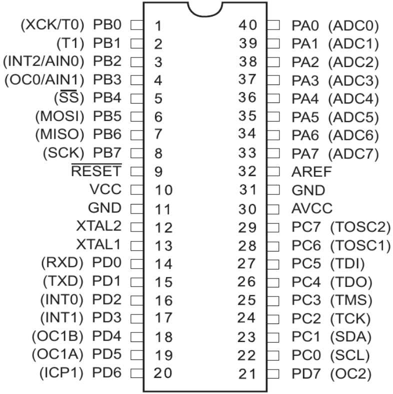
- memory for program and data, a programming interface,
- digital or analog I/O,
- various communication interfaces, e.g., UART, SPI, I^{2}C, CAN,
- timers, PWM generators.

Integrating all into one microcontroller chip improves

- price
- reliability
- energy efficiency
- spatial footprint

- Z80 microprocessor: Address and data pins for memory access
- ATmega32 microcontroller: ADC, voltage references, SPI, external oscillators, . . .

(a) Zilog Z80 microprocessor [02]



(b) Atmel ATmega32 microcontroller [ATmega32]

Application domains

Microcontrollers (MCUs) are “everywhere”:

- About 26 Billion MCUs sold in 2017 [Any18].
- Mechanical solutions were replaced by electric solutions, and electronic hardware is replaced by software on microcontrollers.
- Even plain electric devices (e.g., a light switch) became “smart”.

Application domains:

- Automotive[6] and aerospace industry
- Industrial automation (PLC, I/O modules, servo drives, . . .), IoT/IIoT[7]
- Household products (coffee machine, dish washer, microwave, kitchen scales, . . .)
- Consumer electronics (watch, mobile phone, television, remote control, smart home devices, . . .)
- . . .

6 More than a hundred MCUs in a car (fuel injection ABS, ESP, wiper control, window regulator, in-car entertainment, . . .)

7 (Industrial) Internet of Things

MCUs in a car

A modern car comprises more than 100 ECUs[8]

- For computer science, a car is an embedded, distributed, real-time system.
- ABS and ESP, wiper control, window regulator, in-car entertainment, cruise control, . . .

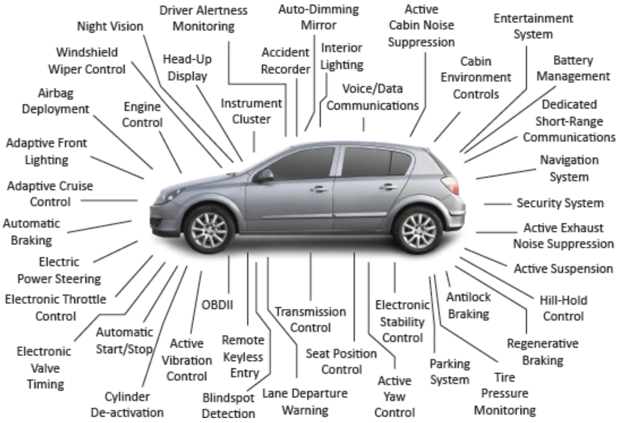


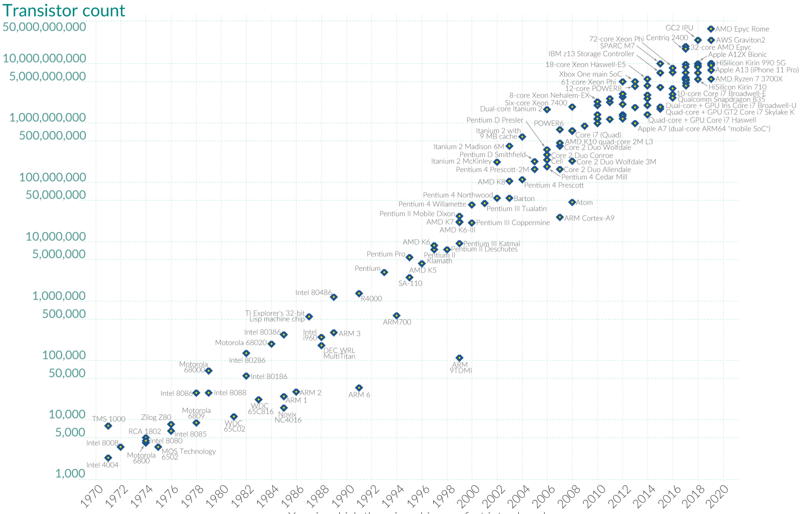
Figure: Features requiring MCUs in a car. Source: https://www.popsautoelectric.com/tag/automotive-trouble-lights/

8 Electronic Control Unit. An ECU contains an MCU.

Moore’s law

An empiric relationship phrased by Gordon Moore (1929–2023): every two years the number of transistors doubles.

Original figure from Our World in Data, hosted on Wikipedia. (Note the logarithmic scale!)



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world’s largest problems.

Diversity

There is a large diversity and spectrum of microcontrollers.[9]

- There is typically a whole family of similar controllers that differ slightly, e.g., in the number of ports, PWM generators, size of RAM and ROM, and the like.
- For instance, the ATmega48A/48PA/88A/88PA/168A/168PA/328/328P only differ in memory sizes (and interrupt vector sizes). [ATmega328fam]

A few examples:

- Historically, the 8-bit Intel 8051 controller family has been important.
- The ATmega32 follows the 8-bit AVR architecture. Same for the ATmega328P (Arduino).
- In industry, the STM32 family with powerful ARM cores gained some popularity.
- The ESP32 family integrates WiFi and Bluetooth. See NodeMCU IoT platform.
- Some have builtin DSP (digital signal processing) functionality – e.g., fixed-point arithmetics – like the 16/32-bit Blackfin family.

9 See [Wiki-ListOfUC] for a list of common microcontrollers.

Microcontroller features

The ATmega32 is simple microcontroller. See [ATmega32, p. 1] for an overview of its features:

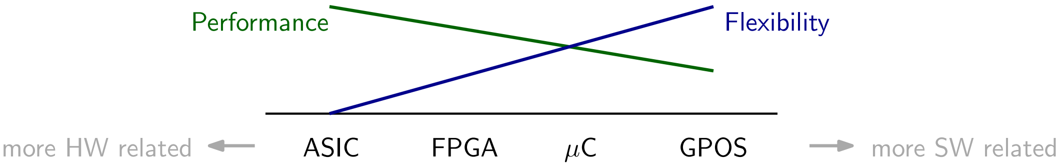
- RISC
- Fully static operations
- Self-programmable
- Peripheral features

Typical features of (lightweigth) microcontrollers:

- 8- or 16-bit register size, larger ones are 32-bit
- RISC
- No Memory Management Unit (MMU), sometimes Harvard architecture
- No Floating Point Unit (FPU)

Technology trade-offs

- **ASIC** An *Application-Specific Integrated Circuit* may contain entire processor cores. It cannot be changed, requires certain quantities to be cost efficient, but gives high performance.
- **FPGA** A *Field-Programmable Gate Array* is like a programmable breadboard with logic blocks using a Hardware Description Language (HDL).[10]
- **μC** A *microcontroller* is typically operated bare metal[11] or with a thin (often proprietary) special-purpose OS. Software is often proprietary or ported, but it can grow over time.
- **GPOS** A *general-purpose OS* (e.g., Linux) provides a rich software ecosystem, quick time to market, high flexibility, but lower performance and less real-time capabilities.



10 CPLDs (Complex Programmable Logic Devices) play a similar role.

11 Without an operating system.

Some more terms

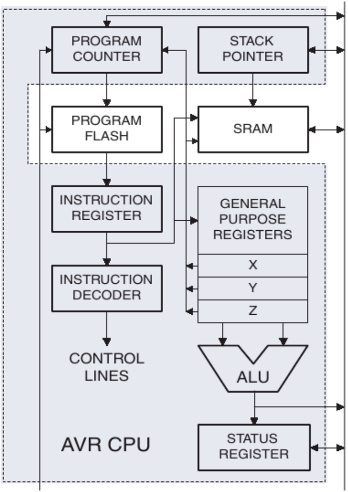
SoC A *System on a Chip* puts all computer components on a single chip. It may contain an ARM processor, an FPGA, a GPU, WiFi modules or sensors (accelerometer, GPS) on a single chip.
Example: SoCs of the Xilinx Zync-7000 family comprise a dual core Arm Cortex-A9 processor and an FPGA.
Example: i.MX 8M Plus comprises a Arm Cortex-A53 with Neural Processing Unit (NPU) for machine learning and a Cortex-M7 for real-time control.

Embedded system Is a computer system that is *embedded* in a larger technical (mechanical or electronic) system. Unlike a general-purpose desktop computer in an office, it has a dedicated function, e.g., as a controller for a smart home temperature control, but also an ordinary PC can be embedded into a system that controls a factory.

Cyber-physical system Is a computer system that interacts with a physical system through sensors and actuators. It emphasizes the interaction of computing, algorithms, software with physics (e.g., mechanics, electronics), like in robotics, smart grids, or autonomous driving.

Processor core

The AVR CPU of the ATmega32 follows the following architecture:



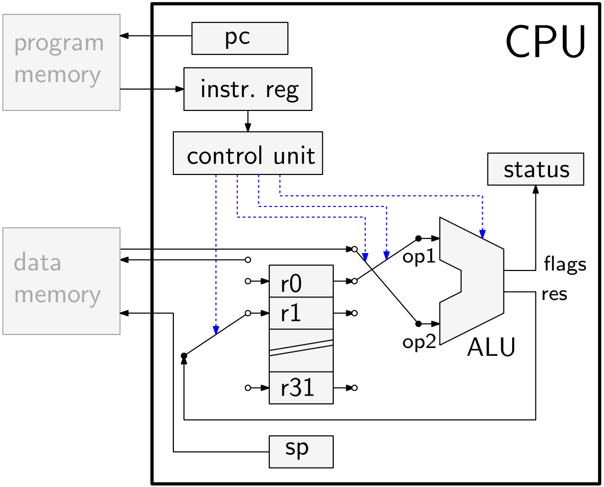
See [ATmega32, p. 3, fig. 2] for the whole figure.

Remarks on the figure:

- The microcontroller contains a microprocessor (AVR CPU)
- The memory (Flash, SRAM) is outside the processor
- The AVR CPU is an 8-bit CPU: the registers are 8 bit wide.

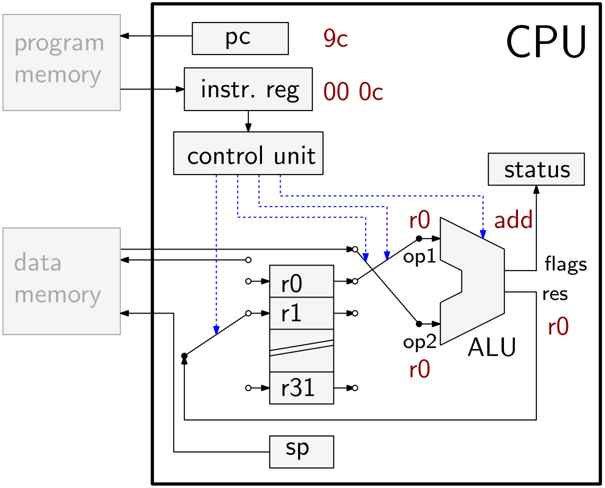
A general basic CPU architecture

- The program counter (pc) contains the address of the current instruction
- The instruction register tells the control unit how data is flowing and what operation to execute in the ALU. It sets the control lines (blue)
- The ALU takes two operands, either from registers or from the data memory
- The status register contains flags for overflow, carry, zero, negative, ...
- The stack pointer (sp) contains the top address of the stack



A snippet of the program memory:

```
8a:   cf 93         # push r28
8c:   df 93         # push r29
8e:   cd b7         # in   r28, 0x3d
90:   de b7         # in   r29, 0x3e
92:   62 e0         # ldi  r22, 0x02
94:   81 e0         # ldi  r24, 0x01
96:   0e 94 36 00   # call 0x6c
9a:   08 2e         # mov  r0, r24
9c:   00 0c         # add  r0, r0
9e:   99 0b         # sbc  r25, r25
a0:   df 91         # pop  r29
a2:   cf 91         # pop  r28
a4:   08 95         # ret
```



Instructions in 2-address format:

- `add x, y` means `x += y`

Status register SREG

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure: See [ATmega32, p. 10].

- I: Global Interrupt Enable
- Z: Zero flag for the result of an arithmetic or logical operation.
- N: Negative flag for the result of arithmetic or logical operation.
- C: Carry flag for the result of an arithmetic or logical operation.
- V: Two's complement overflow flag.

Von Neumann versus Harvard architecture

There are two competing architectures concerning the role of program and data memory.

- von Neumann architecture does not distinguish between instructions and data
- Harvard architecture separates them

von Neumann architecture

There is a common bus to program and data memory, i.e., a common address space.

- Allows to simply load programs from disk, as usual for desktop computers: data fetched from disk is executed as program
- Allows for self-modifying code, e.g., JIT compilation (Java, Python, .NET)
- But also allows for polymorphic viruses and code injection
- It cannot access program and data at the same time: This is called the von Neumann bottleneck.
- Example: x86 architecture

von Neumann versus Harvard architecture

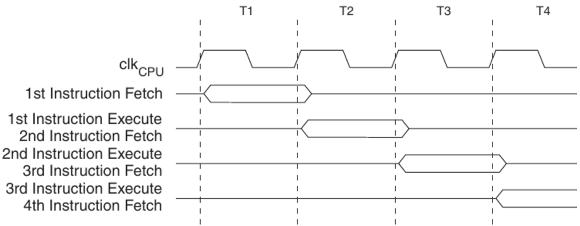
Harvard architecture

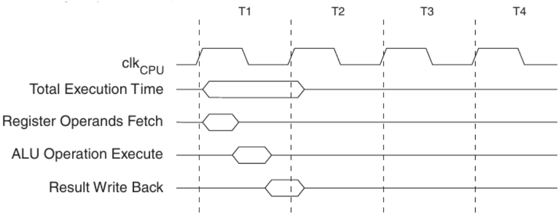
Separate buses to program memory and data memory.

- Allows for concurrent and therefore faster access to program and data
- Address `0x0` can either refer to the program memory or the data memory, i.e., program and data memory have different address spaces

The ATmega32 follows a Harvard architecture:

- See [ATmega32, p. 8, fig. 3]
- Program memory (Flash) and data memory (SRAM) are attached by individual buses





- While the *i*-th instruction is executed (and data memory is accessed), the $(i+1)$-th instruction can be fetched from the program memory in parallel.
- Also one ALU operation is executed in a single CPU cycle. This leads about[12] to 8 MIPS (Million Instructions Per Seconds) for a CPU clock of 8 MHz.
- For modern processors it is essentially impossible[13] to predict the timing of an operation. However, for real-time systems we like determinism.

12 A few instructions take more than one cycle.

13 Caching hierarchies, complex instruction scheduling, pipelining, reordering mechanisms, speculative execution and so on interfere with the precise timing.

Von Neumann versus Harvard architecture

Most modern computers are neither pure Harvard nor pure von Neumann architectures:

- **Split instruction and data cache:**
  Modern von Neumann architectures actually have split instruction and data caches within processors, like the common Intel x86 processors. This is a bit of Harvard in a von Neumann architecture. (See later chapter.)
- **Instruction memory as data:**
  Many Harvard machines provide instructions to access the program memory. For instance, in order to write program code into a microcontroller Flash memory, e.g., for a firmware upgrade.

References I

[02] *Z8400/Z84C00 NMOS/CMOS, Z80 CPU Product Specification*. Zilog, Inc. 2002. URL: http://www.zilog.com/docs/z80/ps0178.pdf.

[Any18] AnySilicon. *MCUs (Microcontrollers) Sales History and Forecast 2016–2022*. Sept. 2018. URL: https://anysilicon.com/mcus-microcontrollers-sales-history-forecast-2016-2022/.

[ATmega32] *ATmega32: 8-bit AVR Microcontroller with 32KBytes In-System Programmable Flash*. Atmel Corporation. Feb. 2011.

[ATmega328fam] *ATmega48A/PA/88A/PA/168A/PA/328/P: ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTESIN-SYSTEM PROGRAMMABLE FLASH*. Atmel Corporation. Nov. 2015. URL: https://dtsheet.com/doc/1815373/atmega48a-pa-88a-pa-168a-pa-328-p---summary.

[Dav01] Martin Davis. *Engines of Logic: Mathematicians and the origin of the Computer*. Collection of Jamie and Michael Kassler. New York, NY, USA: W. W. Norton & Co., Inc., 2001. ISBN: 9780393322293.

References II

[Dol13] Stephen Dolan. "mov is Turing-complete". In: (July 2013). URL: http://web.archive.org/web/20190331191157/https://www.cl.cam.ac.uk/~sd601/papers/mov.pdf.

[Dom] Chris Domas. *M/o/Vfuscator2*. URL: https://github.com/Battelle/movfuscator.

[Fri18] Klaus Fricke. *Digitaltechnik*. 8th ed. Springer Vieweg, 2018. ISBN: 978-3-658-21065-6.

[gnu-c] *The GNU C Reference Manual*. URL: https://www.c-asm.com/gnu-c-manual.html.

[GW07] Günther Gridling and Bettina Weiss. *Introduction to Microcontrollers*. lecture notes. 2007. URL: https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf.

[HP12] John L. Hennessy and David A. Patterson. *Computer Architecture*. 5th ed. Morgan Kaufmann, 2012. ISBN: 978-0-12-383872-8.

References III

[Koc15] Stephen G. Kochan. *Programming in C*. 4th ed. Pearson, 2015. ISBN: 9780132781190.

[Kop11] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. 2nd. Springer Publishing Company, Incorporated, 2011. ISBN: 9781441982360.

[Kut18] Rade Kutil. {*C, C++;*}. lecture notes. 2018. URL: https://www.cosy.sbg.ac.at/~rkutil/ccpp18/ccpp.pdf.

[Sch08] Günter Schmitt. *Mikrocomputertechnik mit Controllern der Atmel-AVR-RISC-Familie*. 4th ed. Oldenbourg, 2008. ISBN: 978348657906.

[TB14] Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. 4th ed. Prentice Hall Press, 2014. ISBN: 013359162X.

[TW11] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. 5th ed. Pearson, 2011. ISBN: 9780132126953.

[Wiki-ListOfUC] *List of common microcontrollers*. URL: https://en.wikipedia.org/wiki/List_of_common_microcontrollers.

The universal computer

Code as data and universal computers

- The breakthrough idea behind Babbage's *Analytical Engine* in 1833 was to construct a machine which we can tell what to do through a *program* rather than building specialized machines.
- Are there (computability) problems that can only be solved on modern processors? Is there a sorting method or a matrix inversion method that *requires*, say, a newer Intel Core i7 processor?

Code as data and universal computers

- The breakthrough idea behind Babbage's *Analytical Engine* in 1833 was to construct a machine which we can tell what to do through a *program* rather than building specialized machines.
- Are there (computability) problems that can only be solved on modern processors? Is there a sorting method or a matrix inversion method that *requires*, say, a newer Intel Core i7 processor?
- Alan Turing introduced in 1936 a simple mathematical computing model called Turing machine. (But more complex than a finite automaton.) According to the Church-Turing thesis for *any* calculable function there is a Turing machine that can compute it.
- There are universal Turing machines (UTM) that can simulate *any* other Turing machine. A processor (resp. programming language) is called Turing-complete if it can simulate a UTM. Hence, all calculable functions are therefore computable by such a processor (resp. by a program in this programming language).
- A UTM treats a Turing machine as input data. This is the origin of stored-program computers, code as data and the von Neumann architecture (1946). [Dav01]
- Already the simplest microprocessors from the 70s were Turing-complete: regarding computability they are just as powerful as any modern processor. They are universal computers.

The mov instruction of x86

The x86 instruction set is a CISC instruction set with complex addressing modes.

- In fact, the mov instruction alone is so powerful and versatile, it is Turing complete by itself [Dol13].
- There is even a compiler called *movfuscator* [Dom] that uses mov instructions only.