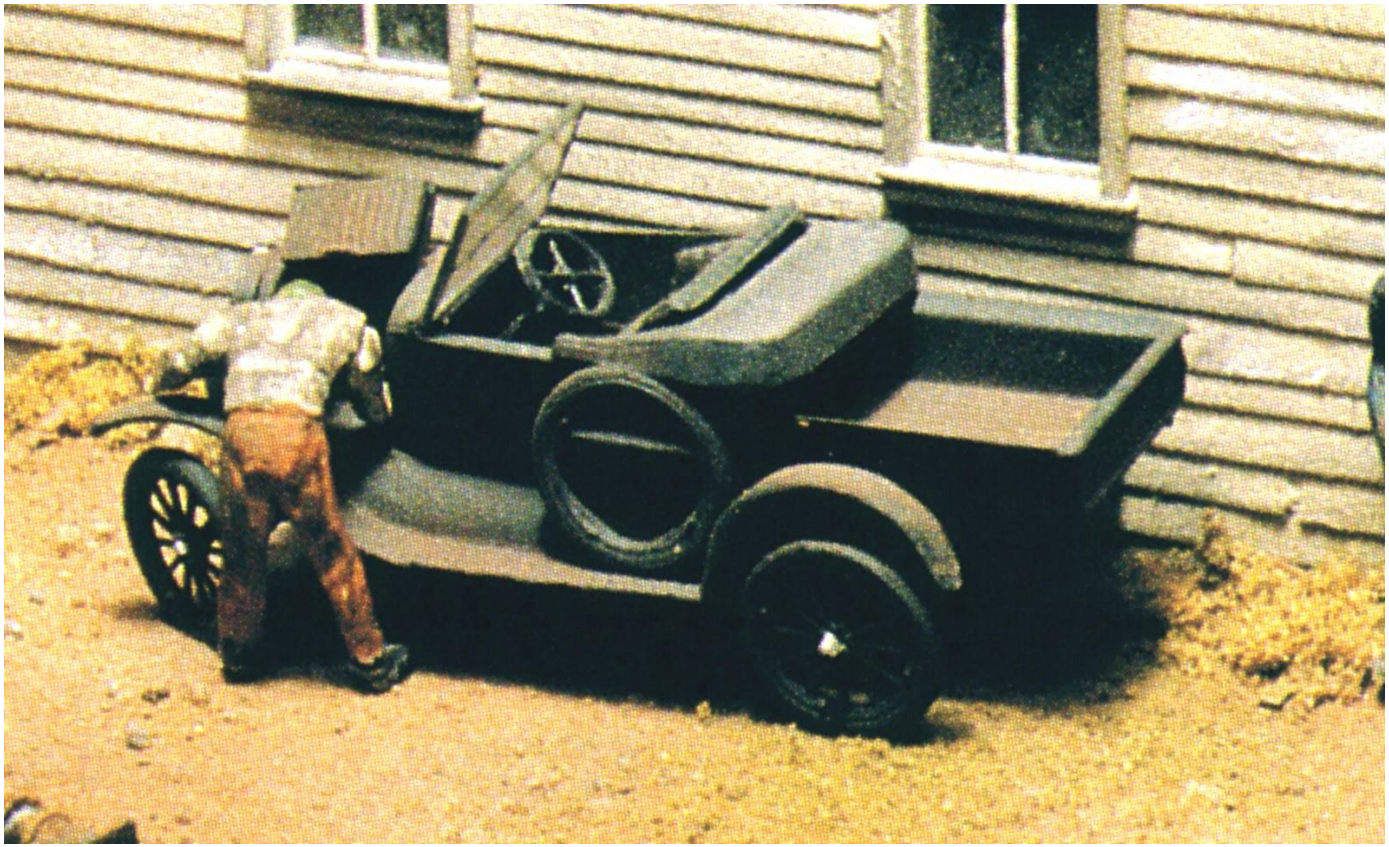
Computer Engineering

Computer Engineering 1

Motivation

School of Engineering
InES Institute of Embedded Systems

- **See what's inside**



Motivation

“I think there is a world market for maybe five computers.“
Thomas Watson, IBM, 1943

“Computers in the future may weigh no more than 1.5 tons.”
Popular Mechanics, 1949

“The number of transistors per IC doubles every year.”
Gordon Moore, Fairchild, 1965

”There is no reason for any individual to have a computer in his home.“
Ken Olson, DEC, 1977

“640K ought to be enough for anybody.”
Bill Gates, Microsoft, 1981

Todays Agenda

School of Engineering
InES Institute of Embedded Systems

- What is Computer Engineering?
- Course Content and Organization
- Computer History
- Properties of a Computer System
- von Neumann Architecture
- Hardware Components
  - CPU, Memory, Input/Output, System Bus
- Software Aspects
  - from C to executable
- Interaction of Hardware and Software

What is Computer Engineering?

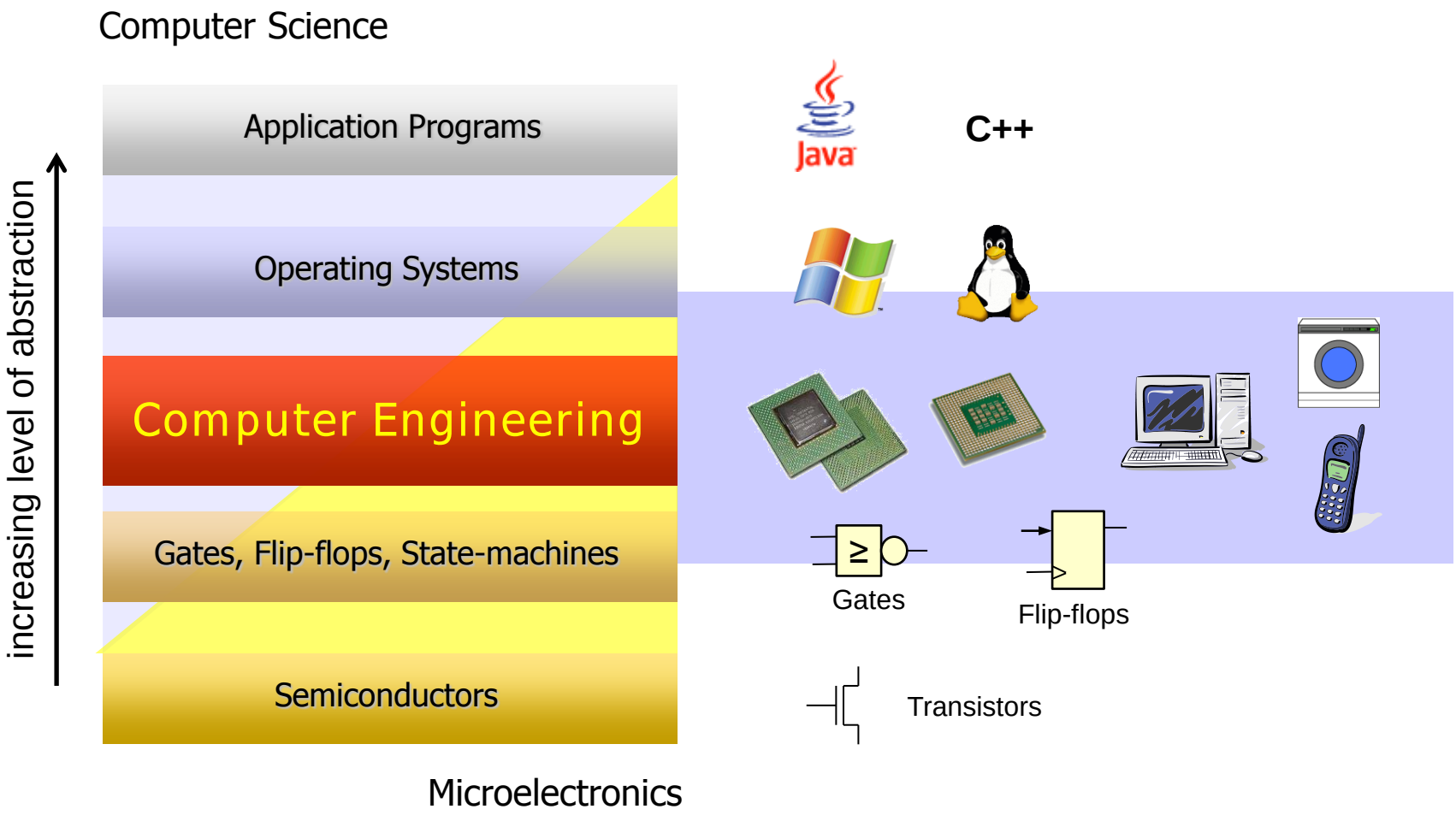
School of Engineering
InES Institute of Embedded Systems

- **Computer Engineering** (Technische Informatik)
  - architecture and organization of computer systems
  - combines hardware and software to implement a computer
- Where **Microelectronics** and **Software** meet
  - 70 years of computer hardware
    - 1940s relay / vacuum tubes
    - 1950s transistors
    - 1970s integrated circuits (CMOS[1)])
  - 40 years of software → Computer Science
    - Assembly Language ("Assembler")
    - High Level Language (e.g. C, ...)
    - Object Oriented Programming (C++, Java, ...)
    - Visual Programming (Model Driven Design)

[1)] Complementary metal–oxide–semiconductor

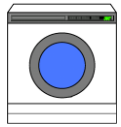
What is Computer Engineering?

School of Engineering
InES Institute of Embedded Systems



Applications

- **Embedded Systems**
  - often part of a larger system
  - control of devices, facilities, processes
  - wireless sensor networks (WSN)
- **Information Technology**
  - communication networks
  - processing of data
  - multimedia
- **Tools**
  - support of technical and scientific activities
  - simulation and modeling
  - logging and analysis of measurement data



Objectives CT 1

School of Engineering
InES Institute of Embedded Systems

After the course you will be able to

- describe the architecture and the operation of a basic computer system and a processor
- to explain how instructions are executed
- to describe the main architectures and performance features of processors as well as the concept of pipelining
- to comprehend how structures in C are compiled into executable object code and to use this knowledge to eliminate programming errors and to optimize program performance
- to develop, debug and verify basic hardware-oriented programs in C and in assembly language
- to explain the concept of interrupts and exceptions and to implement basic interrupt applications
- to find their way in other microprocessor systems

Course Content CT 1

School of Engineering
InES Institute of Embedded Systems

- **Organization of computer systems**
  - Representation of information
  - Program translation
  - Architecture: CPU, Memory, I/O, Bus
- **CPU: Principle of Operation**
  - Instruction set
  - Program execution
  - Memory map, little endian vs. big endian
- **Data transfer**
  - Addressing modes
  - Integer data types, arrays, pointers
- **Arithmetic and logic operations**
  - Computing with the ALU
  - Integer casting
- **Control flow**
  - Compare and jump instructions
  - Structured programming
- **Machine code**
  - Encoding of instructions and operands
- **Subroutines/functions**
  - Parameter passing
- **Exceptional Control Flow**
  - Hardware interrupts, interrupt service routine, vector table
  - Exceptions (Traps)
- **Linking**
  - Address Resolution and Relocation
  - Linker Map and Symbol Table
  - Static Linking vs. Dynamic Linking
- **Processor architectures**
  - von Neumann vs. Harvard, Pipelining
- **Hardware-oriented programming labs**
  - Working with cross-compiler, assembler, linker, loader and debugger

Objectives for Today's Lesson

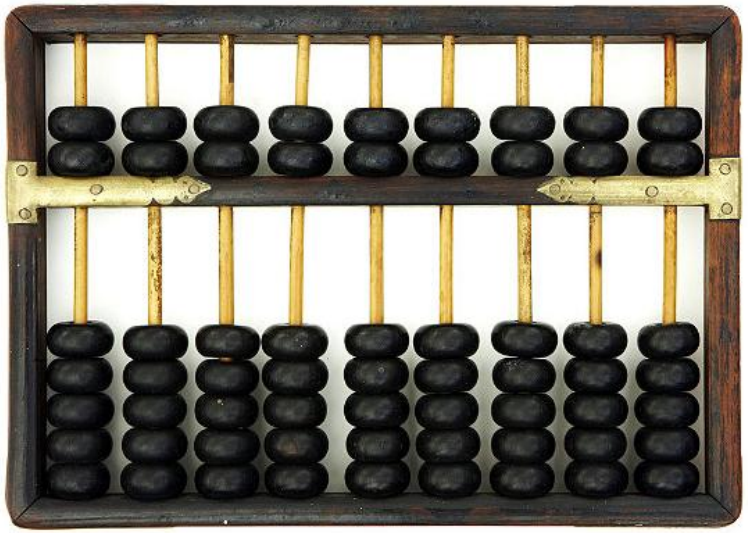
School of Engineering
InES Institute of Embedded Systems

You will be able to

- outline and explain the function of a simple computer system
- name the four main hardware components of a computer system and to describe their functions
- describe different forms of memory and storage
- recall and explain the four translation steps from source code in C to an executable program
- comprehend the use of target and host during development
- explain why knowledge of assembly language is important

Computer History

- **Support for calculations**
  - Babylonian / Chinese — between 1000 und 500 BC: Abacus
  - John Napier — beginning AD 1600: tables for multiplications and logarithms



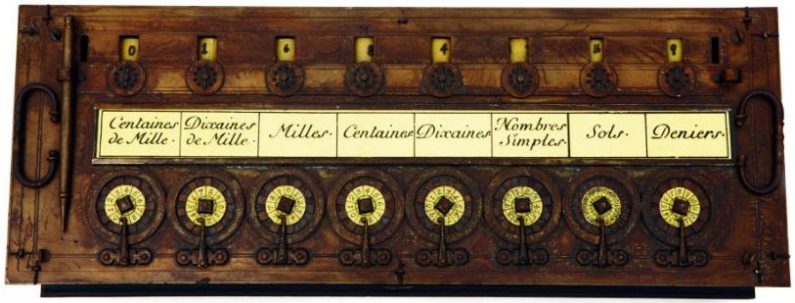
Abacus from *www.computerhistory.org*

Napier's Bones from *www.computerhistory.org*

Computer History

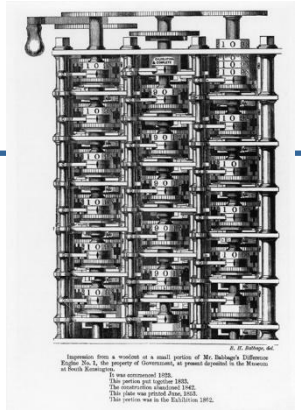
- **First mechanical computers: + - (* /)**
  - Leonardo da Vinci (1452 - 1519)
    - around 1500 → rebuilt successfully in 1967
  - Wilhelm Schickard (1592 - 1635)
    - around1625 → no preserved originals, rebuilt
  - **Blaise Pascal (1623 - 1662)**
    - around 1640 → arithmetic machine (Pascaline)
  - Gottfried von Leibnitz (1646 -1716)

    → enhancement of arithmetic machine



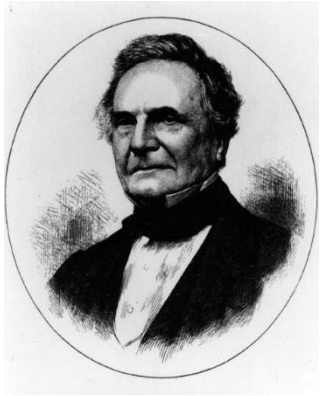
replica of a Pascaline from *www.computerhistory.org*

Computer History



School of Engineering
InES Institute of Embedded Systems

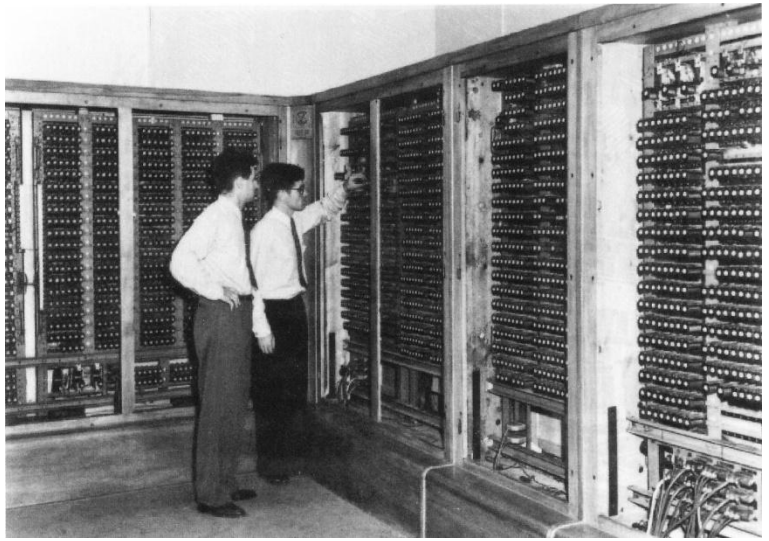
- **First mechanical computer in today's sense**
  - Charles Babbage
    - around 1822 "Difference Engine", not completed
    - replaced by "Analytical Machine"
  - Ada Lovelace
    - Mathematician
    - wrote programs for the Analytical Machine
    - Daughter of Lord Byron



Computer History

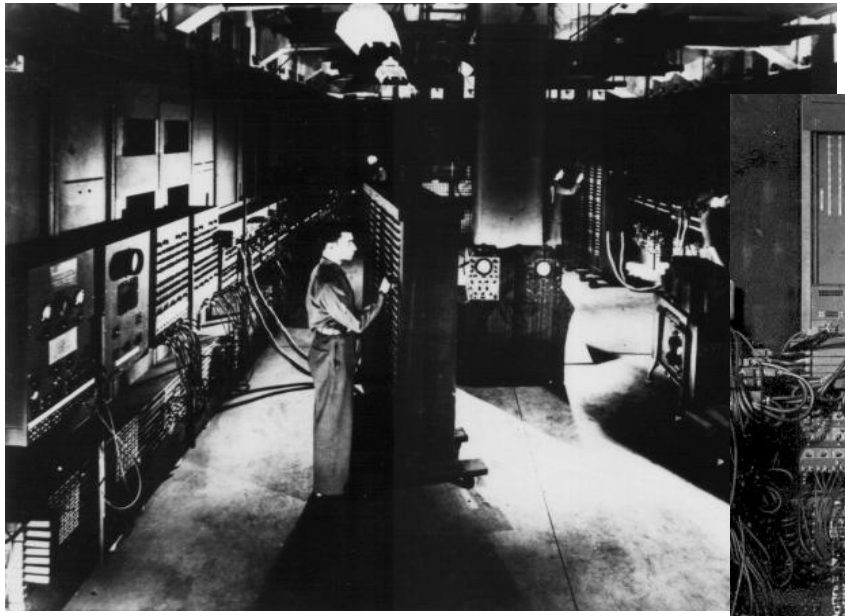
School of Engineering
InES Institute of Embedded Systems

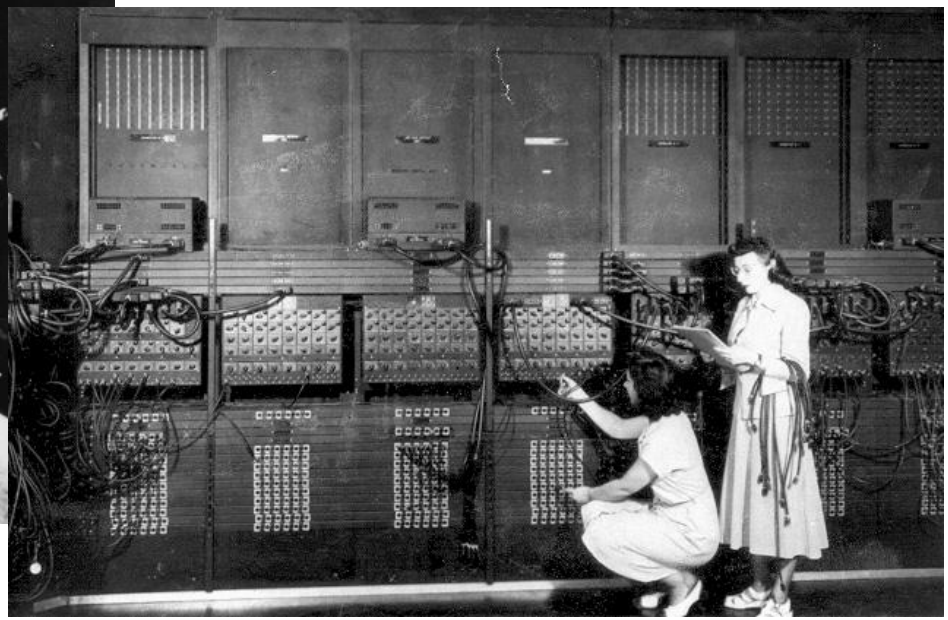
- **First electromechanical computers**
  - Howard H. Aiken
    - Harvard Mark 1, between 1939 and 1944
    - consisting of switches, relays
    - around 750'000 components:
    - 15m x 2.4m x 0.6m, 4500 kg
  - Konrad Zuse, Germany
    - Z3, built in 1941 in Berlin
    - 1944 destroyed by bombing
    - work on Z4 started around 1943
    - used at the ETH from 1950 on



Computer History

- **First electronic "general purpose" computer**
  - J. Presper Eckert and John Mauchly, Univ. of Pensylvenia
    - ENIAC, 1944 → Electronic Numerical Integrator And Calculator
    - around 18'000 tubes, 30 tons, 140 kW, 5'000 additions / s, 1400 m^2

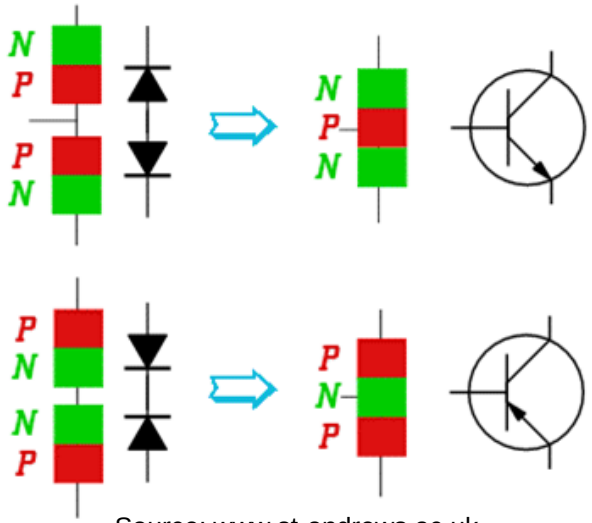




Computer History

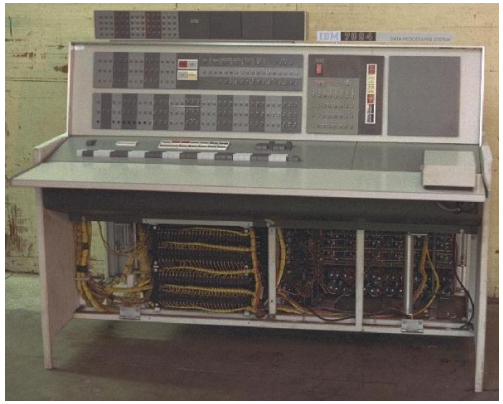
School of Engineering
InES Institute of Embedded Systems

- **First transistors**
  - 1926, patent by Julius Edgar Lilienfeld
  - 1947, Germanium transistor
    - W. Shockley, W. Brattain, J. Bardeen
  - 1950, Bipolar Transistor
    - William Shockley



Source: www.st-andrews.ac.uk

- **Early transistor-based computers**
  - around 1957
    - DEC PDP-1
    - IBM 7000
    - NCR & RCA



Computer History

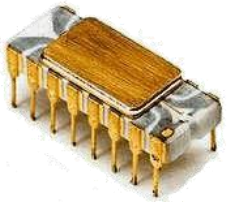
School of Engineering
InES Institute of Embedded Systems

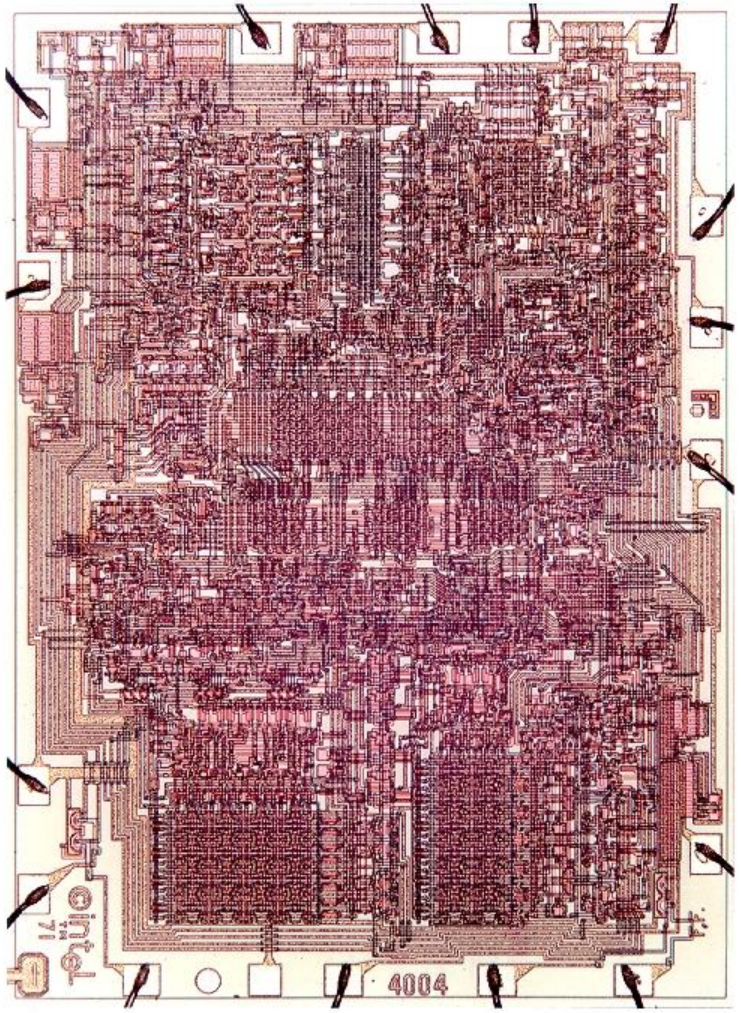
- **Early integrated circuits (IC)**
  - 1958, Jack Kilby at Texas Instruments (TI)
    - based on an idea from 1952
    - several components on the same substrate
  - 1963, Fairchild, "the 907 device"
    - 2 logic gates
  - 1967, Fairchild, "Micromosaic"
    - several 100 transistors
  - 1970, Fairchild
    - first 256-bit static RAM

Computer History

School of Engineering
InES Institute of Embedded Systems

- **Early microprocessors**
  - 1971, Intel 4004
    - all CPU components on a single chip
    - 4 Bit, 2300 transistors
    - 12mm^2 (3x4 mm)
  - 1972, Intel 8008
    - 8-bit version of the 4004

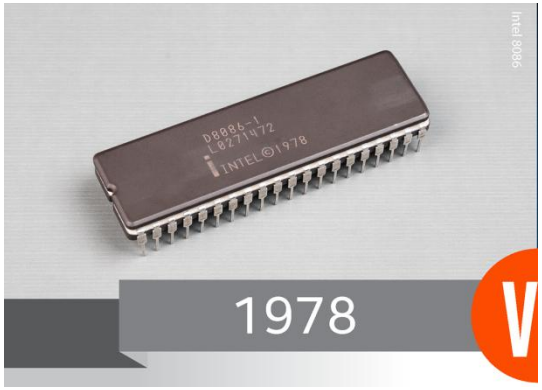




2018: 40 Years 8086

School of Engineering
InES Institute of Embedded Systems

- **1978: Intel 8086**
  - 29000 transistors
  - ~33 mm²
  - 5 MHz
  - 0,33 MIPS



- **2018 Intel Core i7 8086K multi core**
  - 6 Cores
  - ~149 mm²
  - 4 GHz
  - 14 nm gate length
  - 2000 – 5000 mio. transistors (estimated)
  - > 100’000 MIPS

Source: Intel

https://www.heise.de/newsticker/meldung/40-Jahre-8086-der-Prozessor-der-die-PC-Welt-veraenderte-4074260.html

Where are we today?

School of Engineering
InES Institute of Embedded Systems

- **AMD Ryzen 2 Architecture**
  - Up to 64 cores
  - 7 nm (cores) / 12 nm (interconnect) gate length
  - Example: AMD Ryzen 9 3950X, 16 cores (Q3/2019)
    - ~9’890’000’000 transistors
    - ~273 mm^2
    - Up to 4.7 GHz
- **Area**
  - $A_{i7} = \sim 23 \cdot A_{4004}$
- **Transistors**
  - $T_{i7} = \sim 4’300’000 \cdot T_{4004}$

Source: AMD

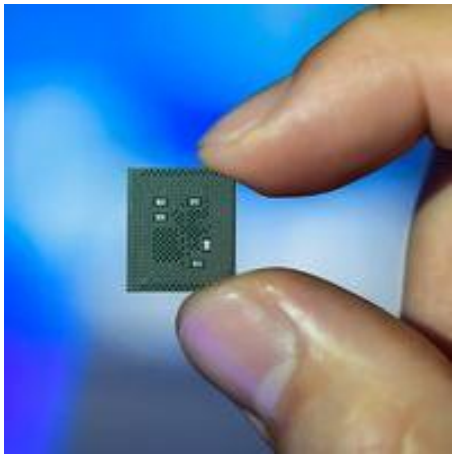
Where are we today?

- **ARM Architecture**
  - Formerly Advanced / Acorn RISC Machines

1985

ARM1
25k Transistors
5 MHz

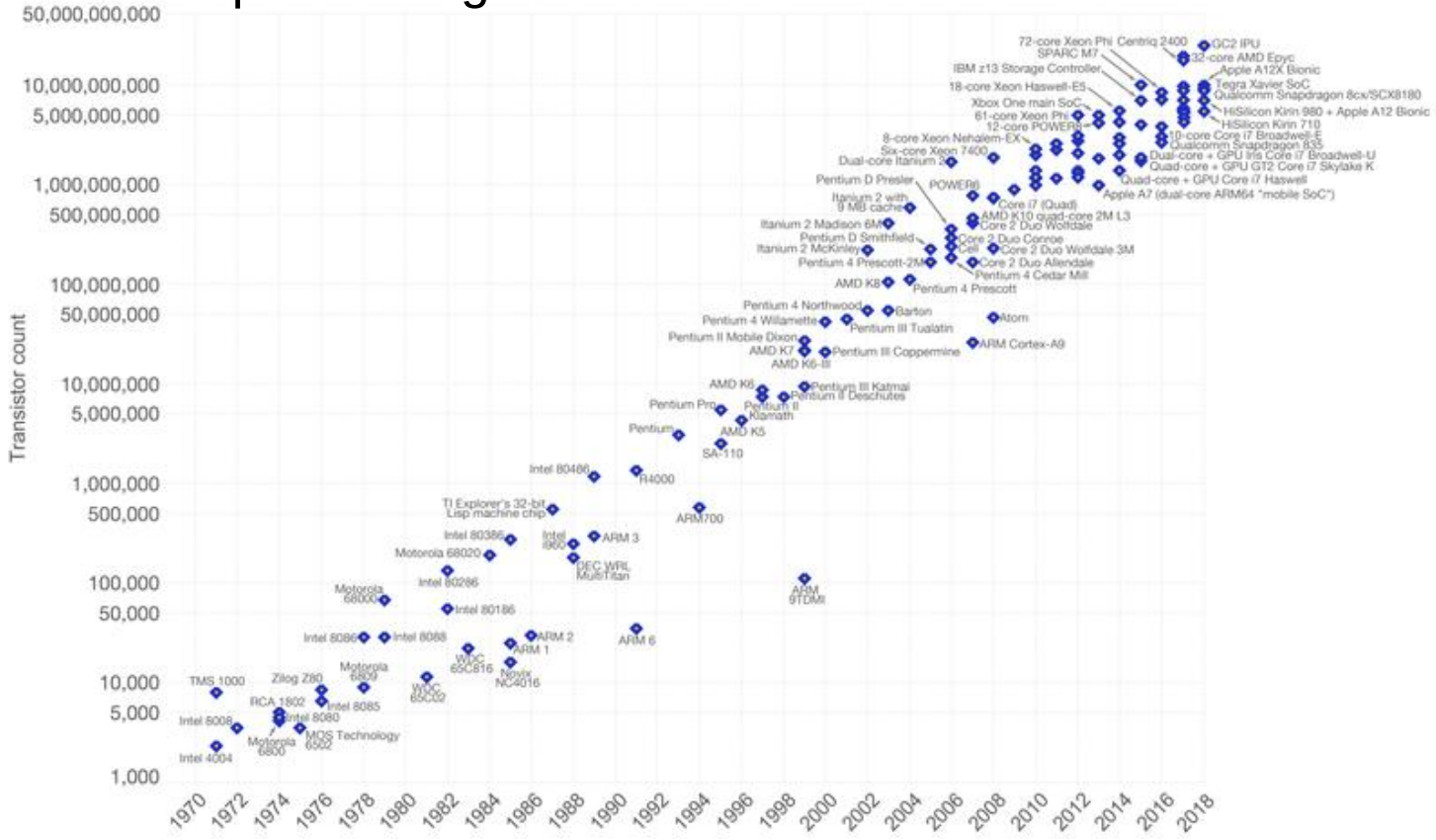
2020 (March)



Eample: Snapdragon 865
ARMv8.2 - Cortex-A77
8 Cores
2,83 GHz
SoC: BLE, WIFI, …

Moore's Law

- **Gordon Moore, Intel**
  - 1965: "The number of transistors per IC doubles every year"
  - somewhat slower since 1975 → doubles every 18 months, i.e. exponential growth



Source: Our World in Data

Computer Engineering Today

School of Engineering
InES Institute of Embedded Systems

Moore's Law lebt

Halbleiter-Trends und neue High-End-Chips

Der Forschungschef des Chip-Auftragfertigers TSMC skizziert auf dem Hot Chips Symposium die Zukunft der Halbleiterindustrie. Und ein Riesen-Chip für künstliche Intelligenz nutzt fast einen ganzen Wafer.

Geht es nach Philip Wong, Forschungschef des weltgrößten Chip-Auftragsfertigers TSMC, dann gilt das Moore'sche Gesetz auch noch im Jahr 2050. Die Metrik, die Wong für seine Aussage nutzt, ist die Packdichte der Transistoren. In der Vergangenheit war die Strukturbreite die gängige Stellschraube für immer dichter gepackte Transistoren, und an ihr werde man in absehbarer Zeit auch weiterhin drehen. 5-nm-Chips seien bereits als Prototypen verfügbar und danach werde es feinere Fertigungsprozesse geben. Wenn die Strukturgrößen in den Bereich weniger Atomlagen schrumpfen, verändert sich allerdings das Verhalten bisheriger Transistortypen zum Schlechteren. Bislang noch nicht verwendete Materialien wie MoS_2, WSe_2 oder WS_2 zeigen hingegen auch bei Dicken von einem Nanometer und darunter noch gute Resultate. Auch an den schon länger bekannten Kohlenstoffnanoröhrchen wird weiterhin viel geforscht. Zusätzlich kämen weitere Verbesserungen hinzu. Als aktuelles Beispiel nannte Wong FinFETs, also dreidimensionale (Gate-)Strukturen auf dem Wafer. Den aktuellen Trend, statt großer monolithischer Dies mehrere kleinere Chiplets zu einem großen Ganzen zusammenzuschalten - wie AMD es beispielsweise bei den jüngst erschienenen Zen-2-CPUs tut -, sieht Wong als einen weiteren Kniff. Die Idee sei aber alles

Flexiprozessor

Das Potenzial von RISC-V-Prozessoren

Die offene CPU-Architektur RISC-V steckt zwar erst in wenigen Chips, begeistert aber viele Entwickler. Denn RISC-V eignet sich für Einsatzbereiche vom Mikrocontroller bis zum Supercomputer und verspricht höhere Sicherheit.

Bild: Albert Hulm

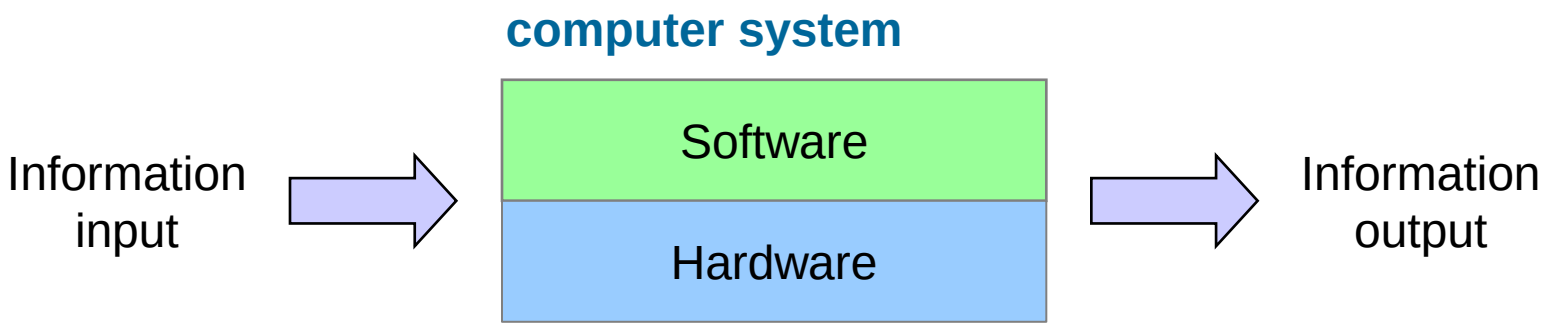
Das Interesse an RISC-V vereint so unterschiedliche Firmen und Institutionen wie Google, russische Waffenhersteller, die NSA, den chinesischen Geheimdienst, Blockchain-Chiphersteller, unterfinanzierte Unis und europäische Supercomputer-Forscher. Manche wollen Lizenzgebühren sparen, andere neuartige Prozessortechnik entwickeln, wieder andere suchen sichere Chips ohne Hintertüren. RISC-V verspricht, solche Anforderungen zu erfüllen, und wird von CPU-Experten heiß diskutiert - obwohl es bisher erst sehr wenige RISC-V-Chips gibt. Der folgende Überblick zeigt, was die

Source: *c't Heft 19/2019*
S. 46-47 / News - Prozessor-Entwicklung
S. 134-139 / Hintergrund - Prozessortechnik

Properties of a Computer System

School of Engineering
InES Institute of Embedded Systems

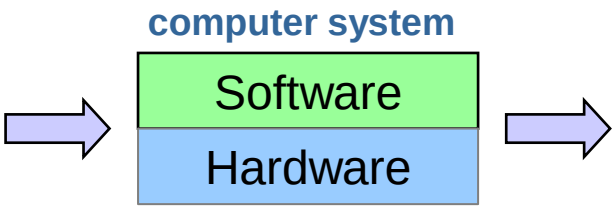
- **A computer system is a device that**
  - processes input
  - takes decisions based on the outcome
  - and outputs the processed information
- **Hardware and software work together → application**
  - often a common hardware is used for many different applications
  - application is defined by the software
    - e.g. controls for washing machines, vending machines, ….

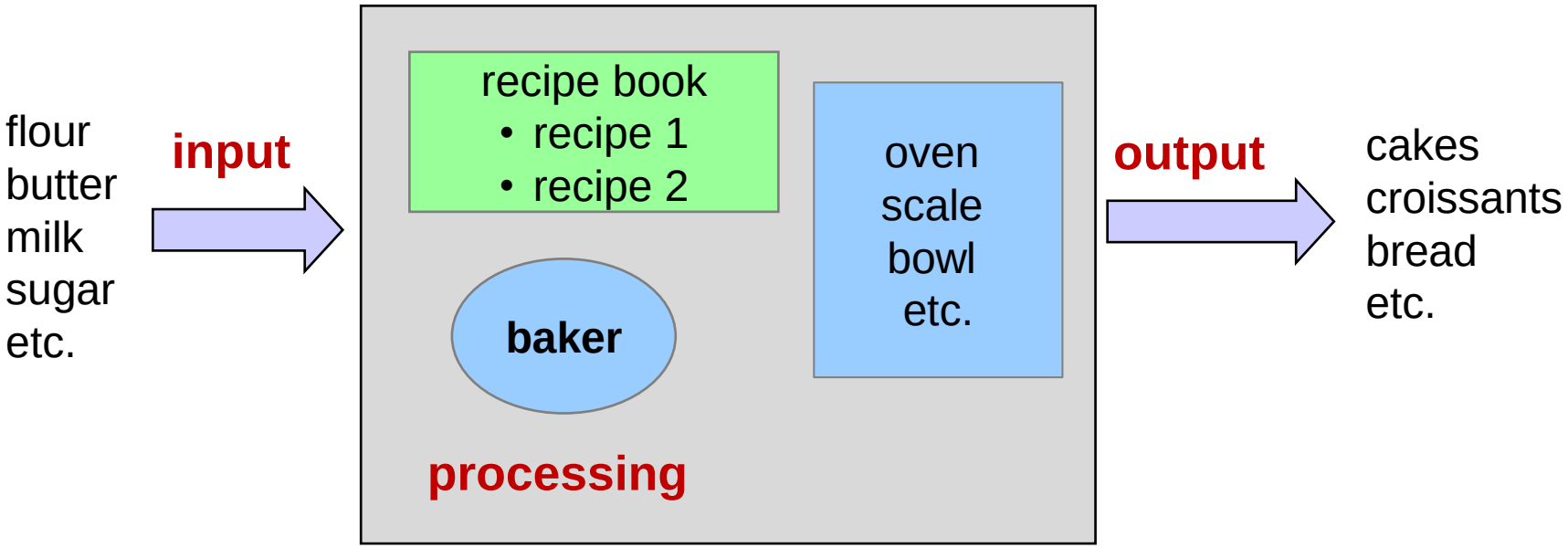


Properties of a Computer System

School of Engineering
InES Institute of Embedded Systems

- **Analogy → bakery**
  - baker ↔ processor
  - recipe book ↔ software
  - tools ↔ HW-resources

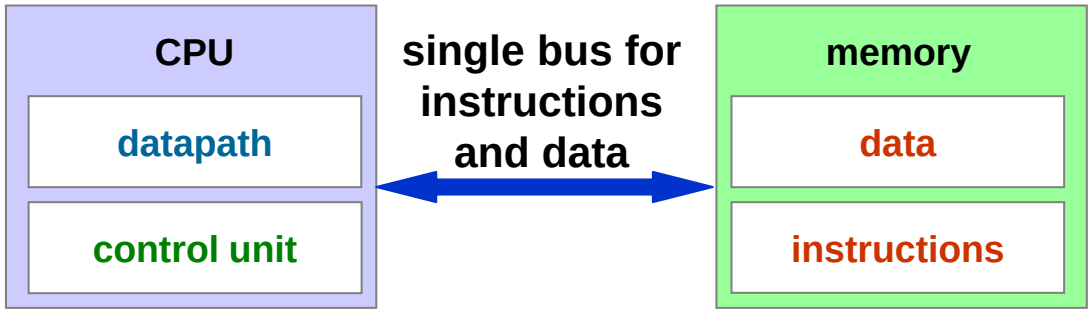




von Neumann Architecture

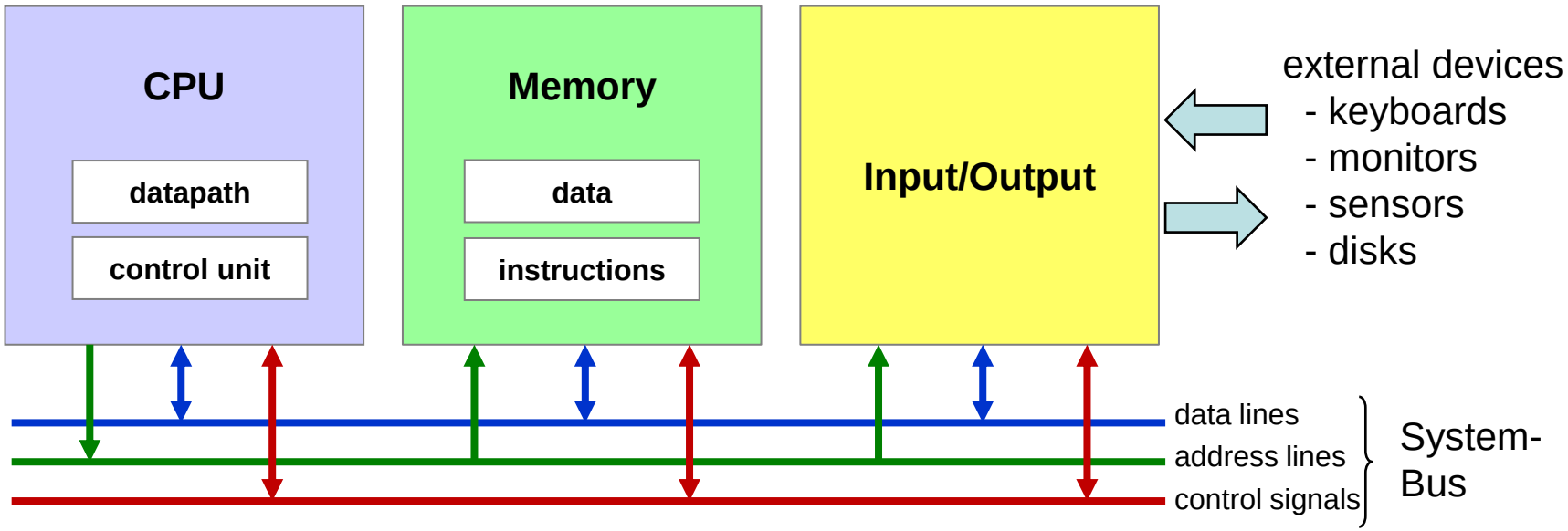
- **Many of today's computers are based on ideas of John von Neumann in the year 1945**
- **Properties**
  - **instructions** and **data** are stored in the same memory
  - **datapath** executes arithmetic and logic operations and holds intermediate results
  - **control unit** reads and interprets instructions and controls their execution

Von Neumann in the 1940s
en.wikipedia.org

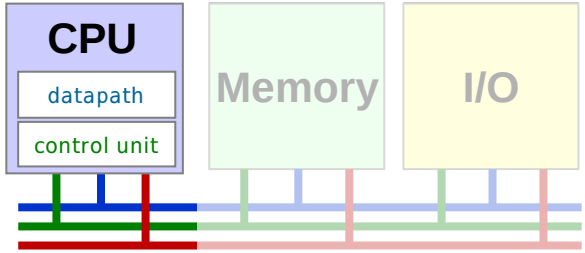


Hardware Components

- **CPU** — **Central Processing Unit or processor**
- **Memory** — **stores instructions and data**
- **Input / Output** — **interface to external devices**
- **System-Bus** — **electrical connection of blocks**



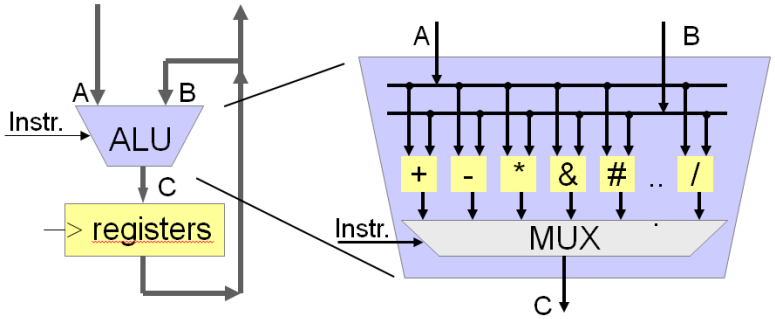
HW Components: CPU



School of Engineering
InES Institute of Embedded Systems

Datapath

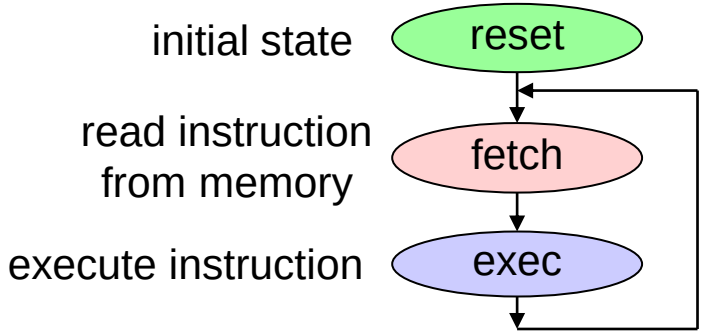
- ALU: Arithmetic and Logic Unit
  - performs arithmetic/logic operations



- registers
  - fast but limited storage inside CPU
  - hold intermediate results
- 4 / 8 / 16 / 32 / 64 bits wide

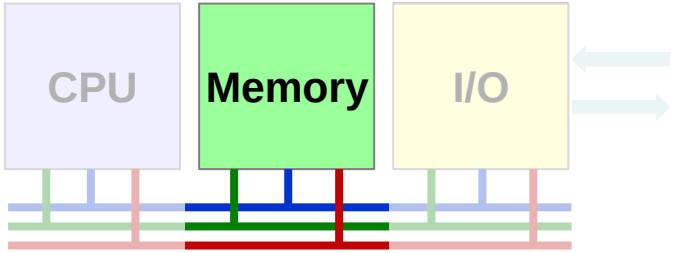
Control Unit

- Finite State Machine (FSM)
  - reads and executes instructions



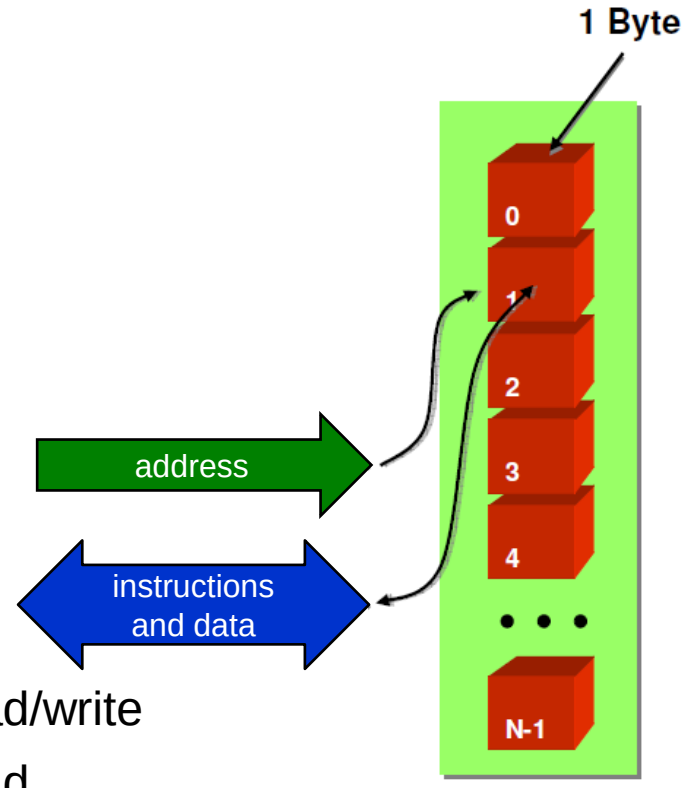
- types of operations
  - data transfer: registers ←→ memory
  - arithmetic and logic operations
  - jumps

HW Components

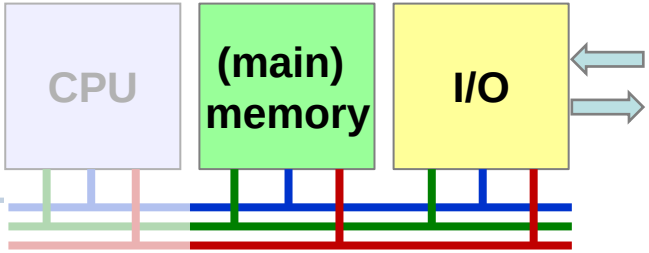


School of Engineering
InES Institute of Embedded Systems

- **Memory**
  - a set of storage cells
    - 8 bit → 1 byte
  - smallest addressable unit
    - one byte
    - one address per byte
  - 2^N addresses
    - from 0 to 2^N-1
    - can be read and sometimes written
    - RAM Random Access Memory read/write
    - ROM Read Only Memory read



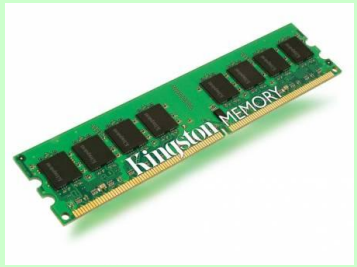
HW Components



School of Engineering
InES Institute of Embedded Systems

Main memory - Arbeitsspeicher

- central memory
- connected through System-Bus
- access to individual bytes
- volatile (flüchtig)
  - SRAM – Static RAM
  - DRAM – Dynamic RAM
- non-volatile (nicht-flüchtig)
  - ROM factory programmed
  - flash in system programmable



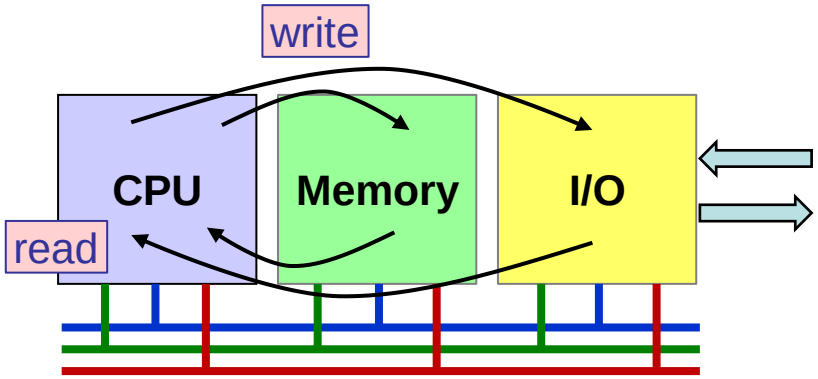
Secondary storage

- long term or peripheral storage
- connected through I/O-Ports
- access to blocks of data
- non-volatile
- slower but lower cost
  - magnetic hard disk, tape, floppy
  - semiconductor solid state disk
  - optical CD, DVD
  - mechanical punched tape/card

HW Components

School of Engineering
InES Institute of Embedded Systems

- **System-Bus**
  - CPU writes or reads data from/to memory or I/O

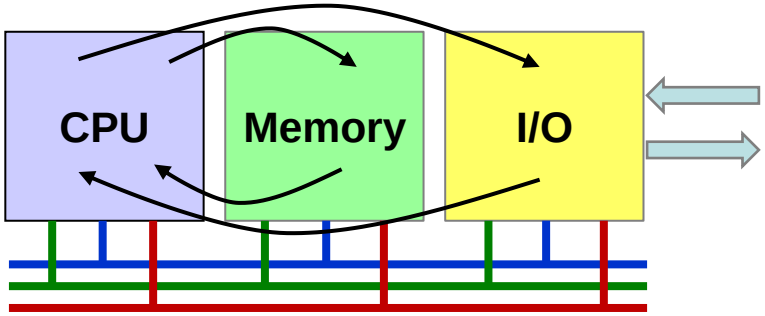


  - **address lines**
    - CPU drives the desired address onto the address lines
      - ► to which address does the CPU write?
      - ► from which address does the CPU read?
      - ► analogy → address on an envelope of a letter
    - number of addresses = 2^n → n = number of address lines
      - ► n = 16 → 2^{16} = 65'536 addresses → 64 KBytes
      - ► n = 20 → 2^{20} = 1'048'576 addresses → 1 MBytes

HW Components

School of Engineering
InES Institute of Embedded Systems

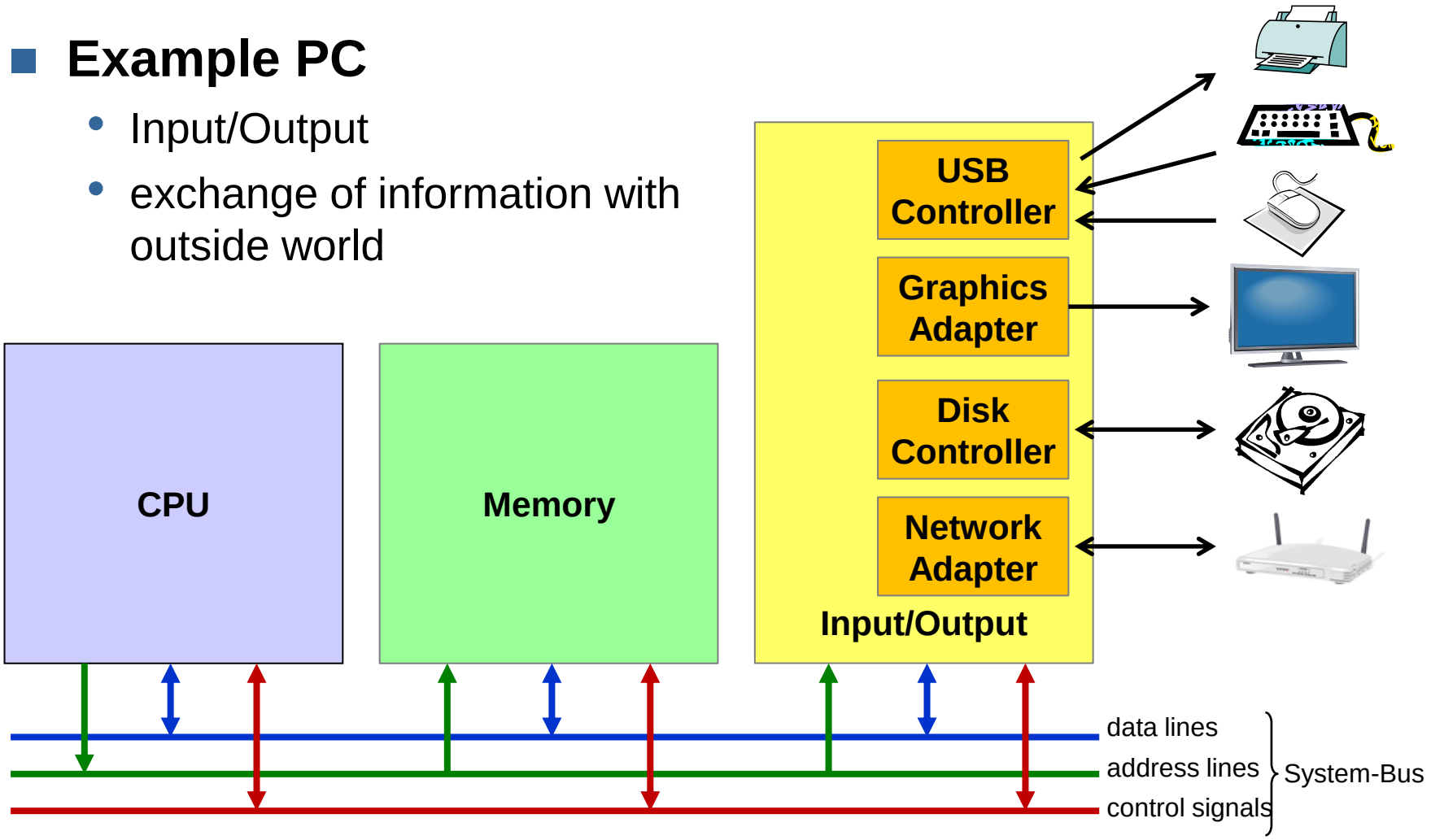
… System-Bus



- **control signals**
  - CPU tells whether the access is read or write
  - CPU tells when address and data lines are valid → bus timing
- **data lines**
  - transfer of data
    - ► analogy the letter that's inside the envelope
    - ► write CPU provides data → memory receives data
    - ► read CPU receives data ← memory provides data
  - 4/8/16/32/64 data lines

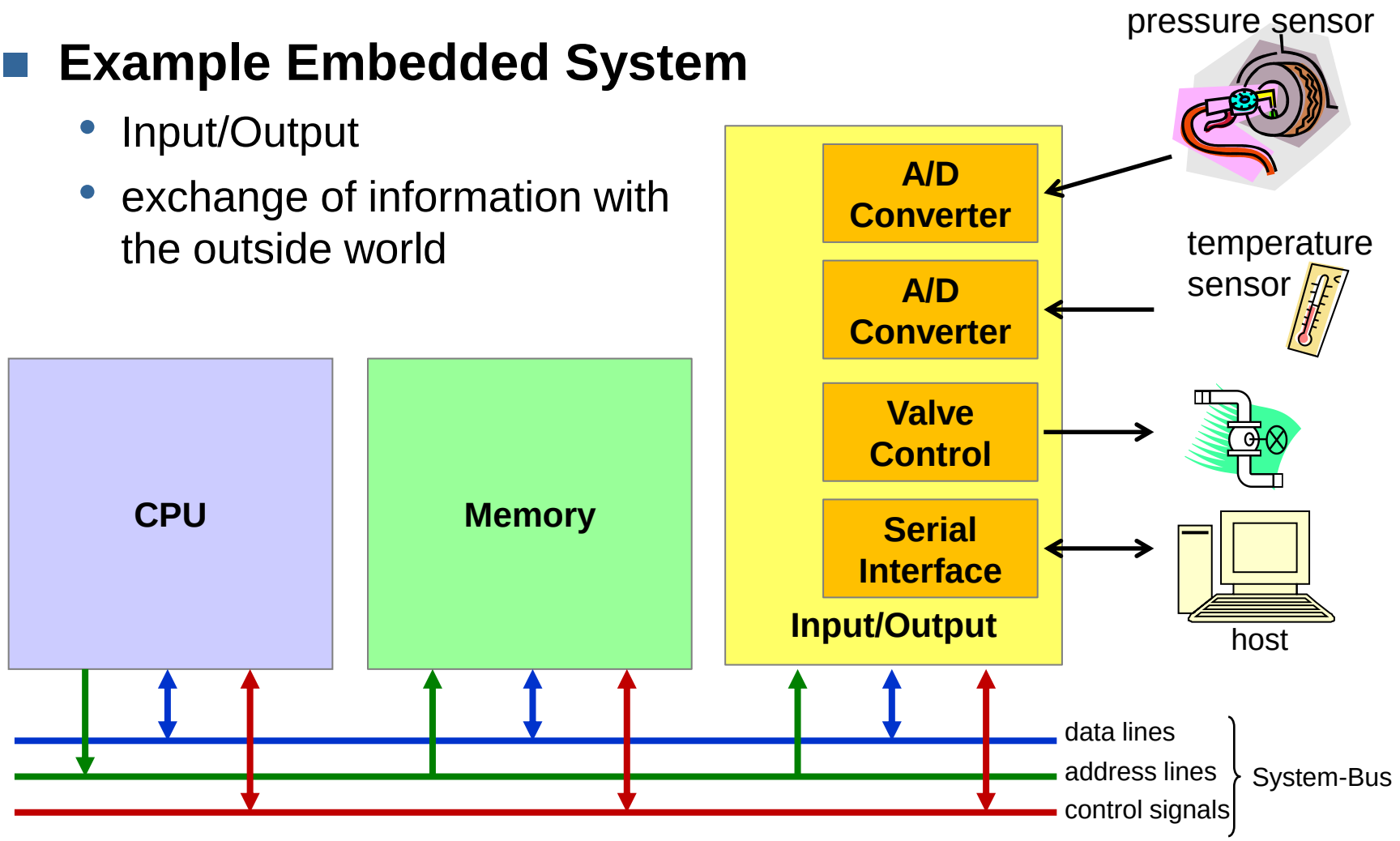
HW Components

- **Example PC**
  - Input/Output
  - exchange of information with outside world



HW Components

- **Example Embedded System**
  - Input/Output
  - exchange of information with the outside world



Software Aspects

So far

- CPU reads instructions from memory and executes them

But

- How to process a program in a high level language like C so that a CPU can interpret the instructions?
- What is needed for a program in C to allow execution on a CPU?
- What does the path from the C source code to the executable object file look like?

Software Aspects

School of Engineering
InES Institute of Embedded Systems

- **Programmer writes `main.c` in text editor**

`main.c`

```
#include "utils_ctboard.h"

#define LED_ADDR    0x60000100
#define LED_VALUE   0x12

int main(void)
{
    while (1) {
        write_byte(LED_ADDR, LED_VALUE);
    }
}
```

calls function **`write_byte()`** from module *utils_ctboard*

Software Aspects

School of Engineering
InES Institute of Embedded Systems

- **`main.c` is stored in ASCII / Unicode format on disk**

```
#include "utils_ctboard.h"

#define LED_ADDR    0x60000100
#define LED_VALUE   0x12

int main(void)
{
    while (1) {
        write_byte(LED_ADDR, LED_VALUE);
    }
}
```

23696E636C75646520227574696C735F
6374626F6172642E68220D0A0D0A2364
6566696E65204C45445F414444522020
.....

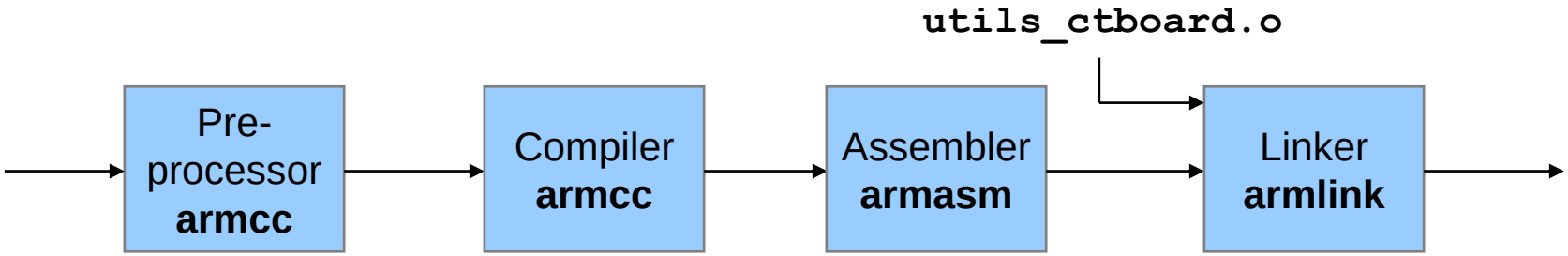
→ 0x23
i → 0x69
n → 0x6E
c → 0x63
....

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	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	Neg. 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	End 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	Substitution	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	□

Software Aspects

School of Engineering
InES Institute of Embedded Systems

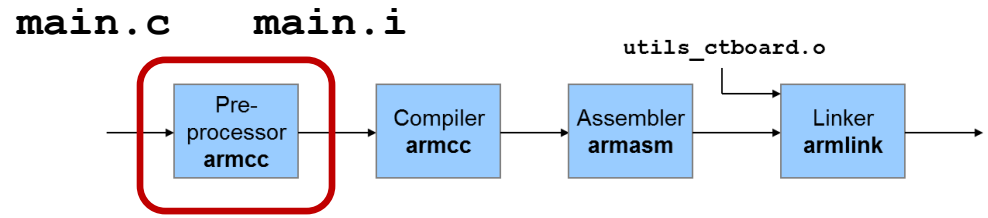
- **From C to executable**
  - Translation of **`main.c`** into machine language



`main.c`	`main.i`	`main.s`	`main.o`	`main.axf`
Source program (text)	*Modified source program (text)*	*Assembly program (text) human-readable, CPU specific*	*Relocatable object program (binary)*	*Executable object program (binary)*

Software Aspects

- **Preprocessor**
  - Text processing
  - Pasting of #include files
  - Replacing macros (#define)



pasting included content of included files

main.c

```
#include "utils_ctboard.h"

#define LED_ADDR    0x60000100
#define LED_VALUE   0x12

int main(void)
{
    while (1) {
        write_byte(LED_ADDR, LED_VALUE);
    }
}
```

main.i

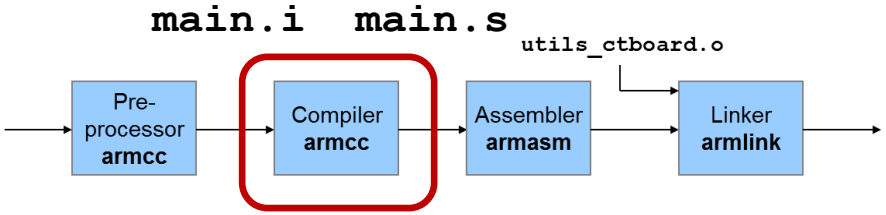
```
void write_byte(uint32_t address,
                uint8_t data);

int main(void)
{
    while (1) {
        write_byte(0x60000100, 0x12);
    }
}
```

replacing macros

Software Aspects

School of Engineering
InES Institute of Embedded Systems

- **Compiler**
  - Translate CPU-independent C-code into CPU-specific assembly code



main.c

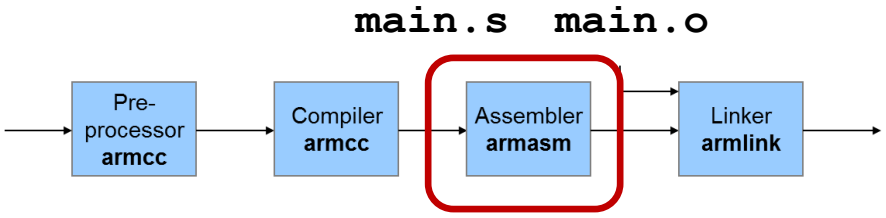
```
void write_byte(uint32_t address,
                uint8_t data);

int main(void)
{
    while (1) {
        write_byte(0x60000100, 0x12);
    }
}
```

main.s

```
 AREA .text, CODE, READONLY

main      PROC
          B         endloop

mloop     MOVS      r1,#0x12
          LDR       r0,L112
          BL        write_byte

endloop   B         mloop
          ENDP

L112      DCD       0x60000100
```

human readable

Software Aspects

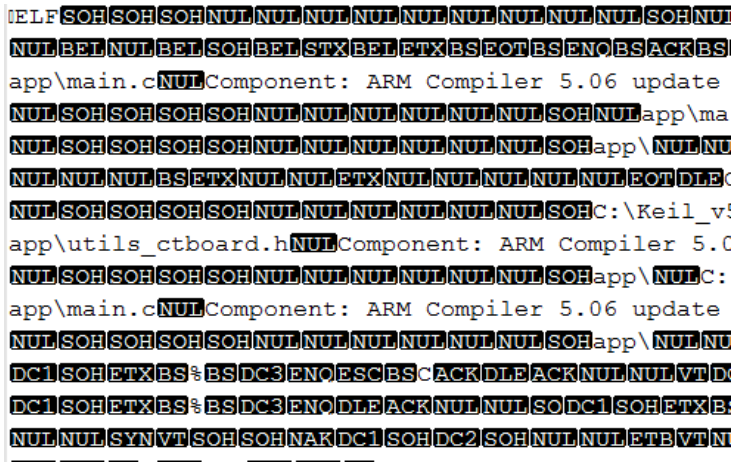


- **Assembler**
  - Translate to machine instructions
  - Result: Relocatable object file
  - Binary file → not readable with text editor, use Hex Dump

main.s

```
 AREA .text, CODE, READONLY

main     PROC
         B          endloop

mloop    MOVS       r1,#0x12
         LDR        r0,L112
         BL         write_byte

endloop  B          mloop
         ENDP

L112     DCD        0x60000100
```

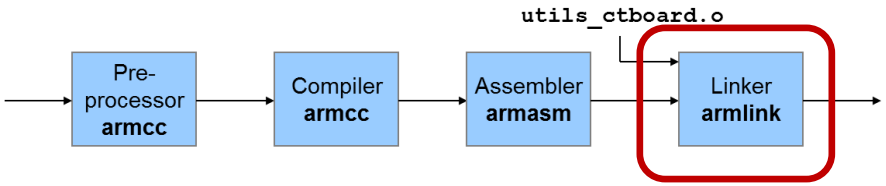
main.o



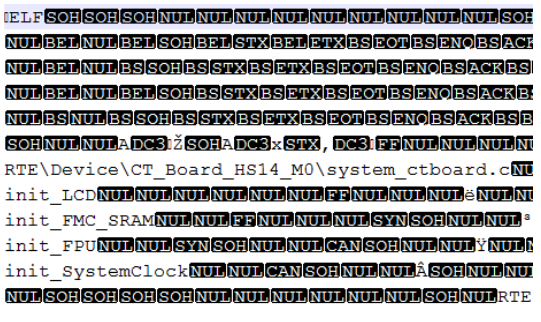
Software Aspects

School of Engineering
InES Institute of Embedded Systems

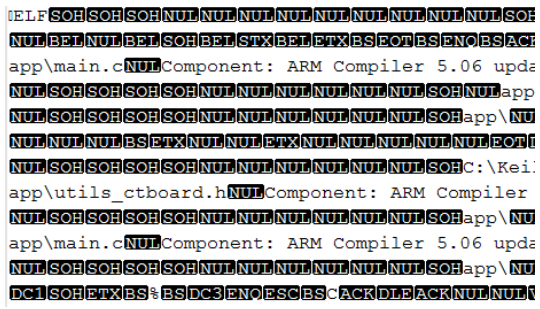
- **Linker**
  - Merge object files
  - Resolve dependencies and cross-references
  - Create executable



utils_ctboard.o



main.o

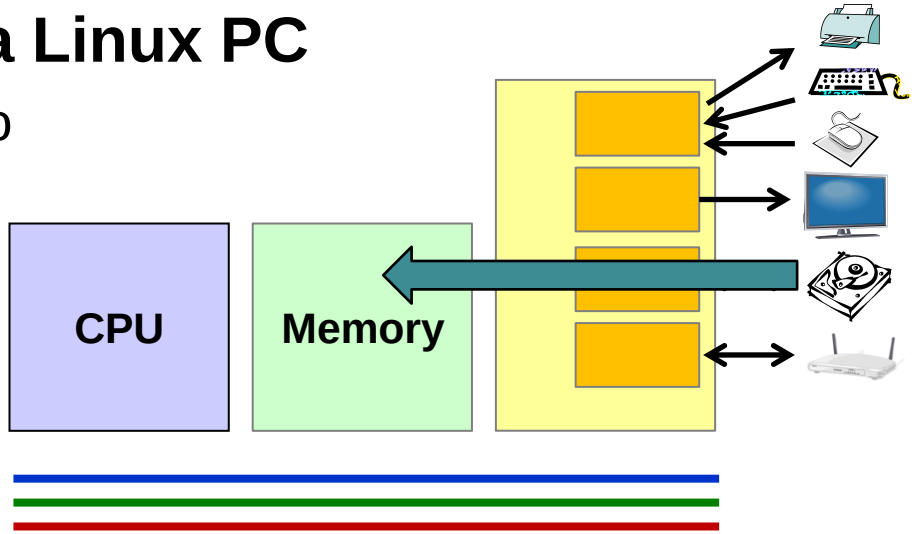


main.axf

remember: **`main()`** calls function **`write_byte()`** from module *utils_ctboard*

Software Aspects

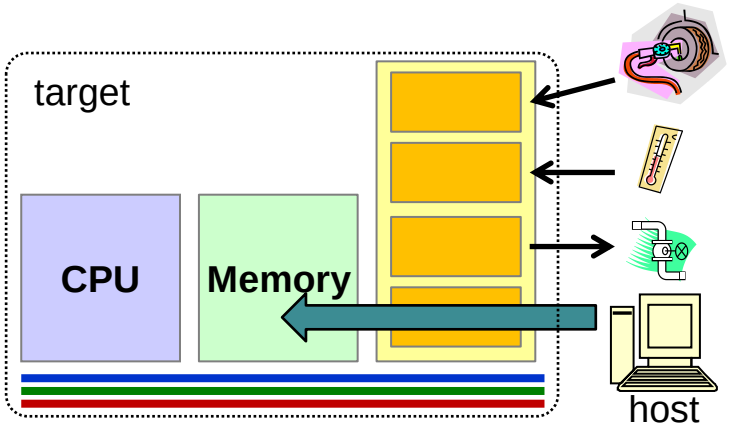
- **Program execution on a Linux PC**
  - load executable **`hello`** into memory and execute it

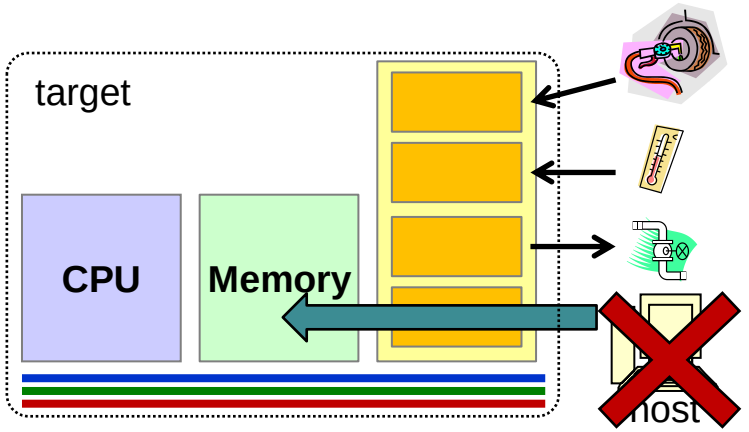


  - typing **`./hello`** in a shell
    - transfers the executable from disk to memory (RAM)
  - operating system
    - creates a new process
    - jumps to start of **`main()`** function and begins execution

Software Aspects

- **Program execution on Small Scale Embedded System**
  - Host vs. Target





Software development on host

- Compiler/Assembler/Linker on host
- Loader on target loads executable (e.g. *.axf) from host to RAM
- Loader copies executable from RAM into non-volatile memory (FLASH)
  → Firmware Update

System operation without host

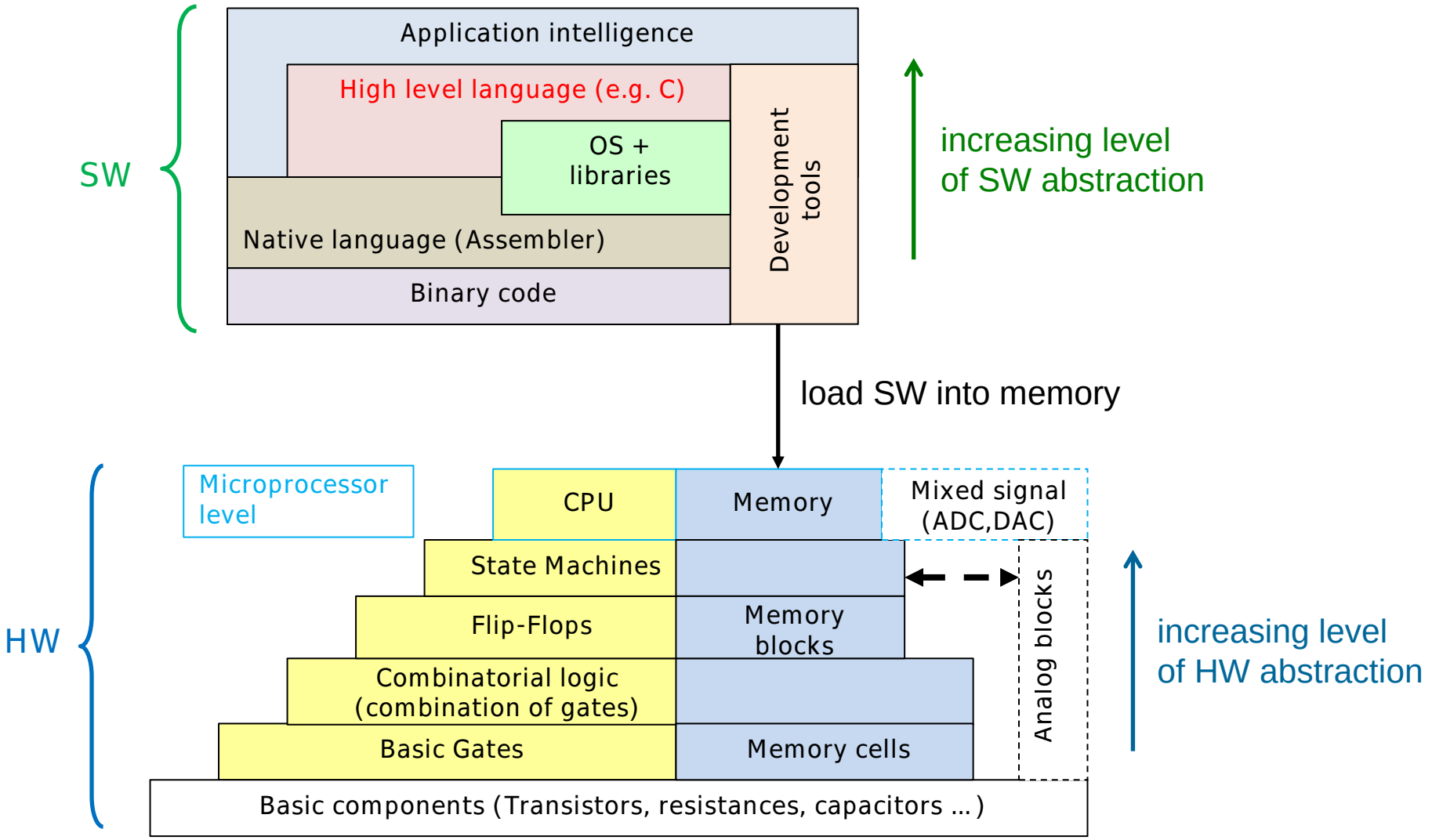
- Loader jumps to `main()` and starts execution
- Instruction fetch often takes place directly from FLASH

Software Aspects

School of Engineering
InES Institute of Embedded Systems

- **Why learn assembly language?**
  - few engineers write assembly code
    - use of High Level Languages (HLL) and compilers more efficient
- **But**
  - assembly language yields understanding on machine level
    - understanding helps to avoid programming errors in HLL
  - increase performance
    - understand compiler optimizations
    - find causes for inefficient code
  - implement system software
    - boot Loader, operating systems, interrupt service routines
  - localize and avoid security flaws
    - e.g. buffer overflow

Interaction of HW and SW



Conclusion

School of Engineering
InES Institute of Embedded Systems

- **Computer system → hardware and software**
- **Hardware**
  - CPU — Central Processing Unit or microprocessor (µP)
  - memory — stores instructions and data
  - I/O — input and output devices
  - system bus — electrical connection of blocks
- **Software**
  - source code in high level language (C)
  - assembly code → machine-oriented, human readable
  - object code → machine instructions in binary without libraries
  - executable → executable object file including libraries
- **Target vs. Host**