



# WIE WERDEN ZAHLEN ZU CODES?

# Andreas Kugel Technische Informatik OK Lab Karlsruhe



# WIE WERDEN ZAHLEN ZU CODES, WIE WERDEN CODES ZU ZAHLEN?

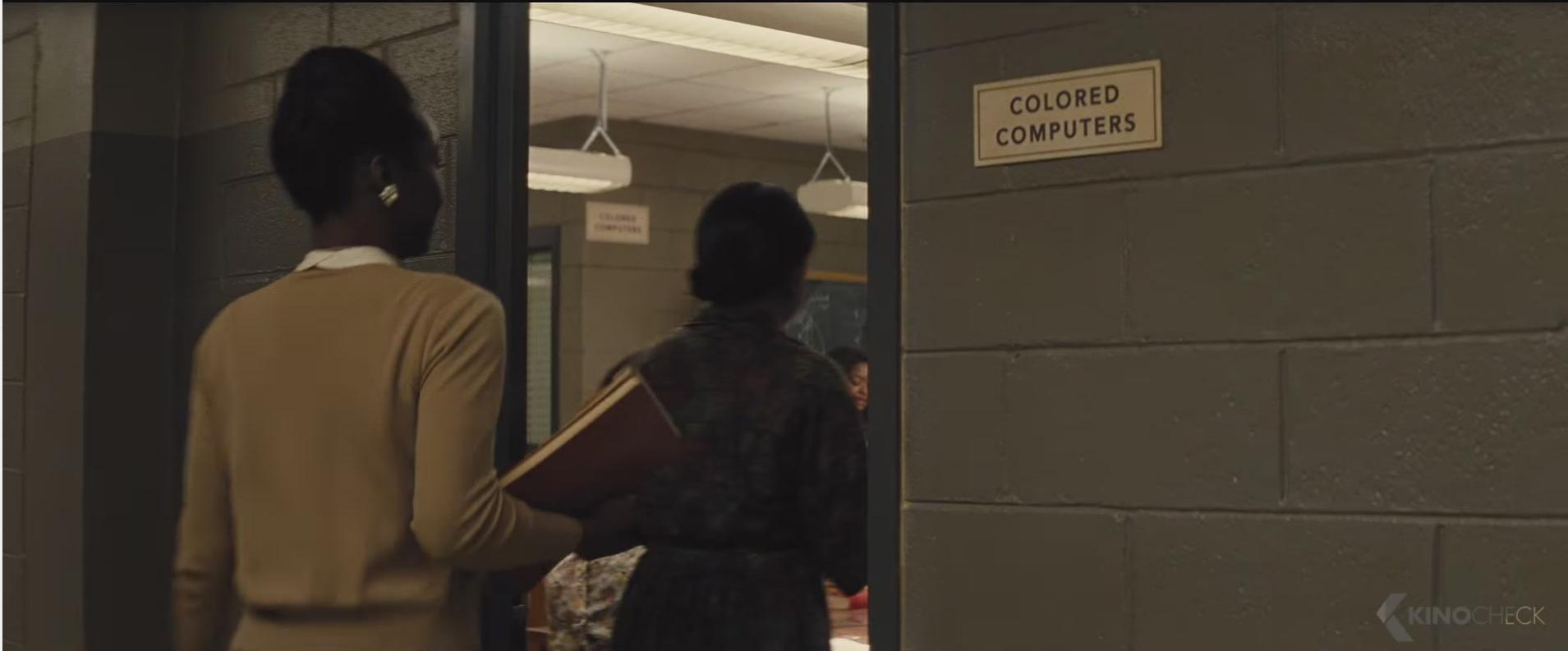
In unserer digitalen Welt sind überall Computer zu finden - große in den Rechenzentren, mittel große auf den Schreibtischen, kleine in den Smartphones oder ganz kleine in Sensoren und Spielzeugen. Ihr »Futter« sind Zahlen: 0-en und 1-en, dafür tun sie alles.

In diesem Workshop wird das Geheimnis dieser Zahlen gelüftet: was sie im Computer für Aufgaben haben, wie diese genau funktionieren und wie wir die richtigen Zahlen herstellen können. Dazu machen wir aus ein paar Zahlen Codes für einen super-einfachen »simulierten« Rechner und wir erstellen ein paar einfache Codes für richtige Computer. Wir nehmen dabei auch Bezug zu verschiedenen Objekten der Ausstellung »open codes. Leben in digitalen Welten«.

Begriffe wie Binärkode, Rechenwerk, Unterprogramm oder Variable werden danach keine Fremdworte mehr sein.



# DIE ERSTEN COMPUTER WAREN MENSCHEN



KINOCHECK



# Was verbinden wir mit „Computer“?

Technische  
Faszination

Kunst

Mathe

Werkzeug

Kreativität

Unterhaltung

Kommunikation

Maschine





# MEIN HINTERGRUND

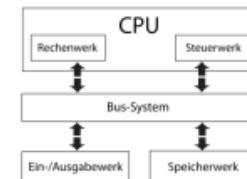
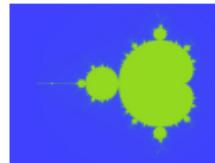
- Technische Informatik (Uni Heidelberg)
- CERN, XFEL
- OKLAB, Edulabs



# Themen

```
while (zx2 + zy2 < 4.0){  
    tmp = zx2 - zy2 + cx;  
    zy = 2.0*zx*zy + cy;  
    zx = tmp; }
```

Software/  
Codes



Geschichte



Micro  
Prozessor

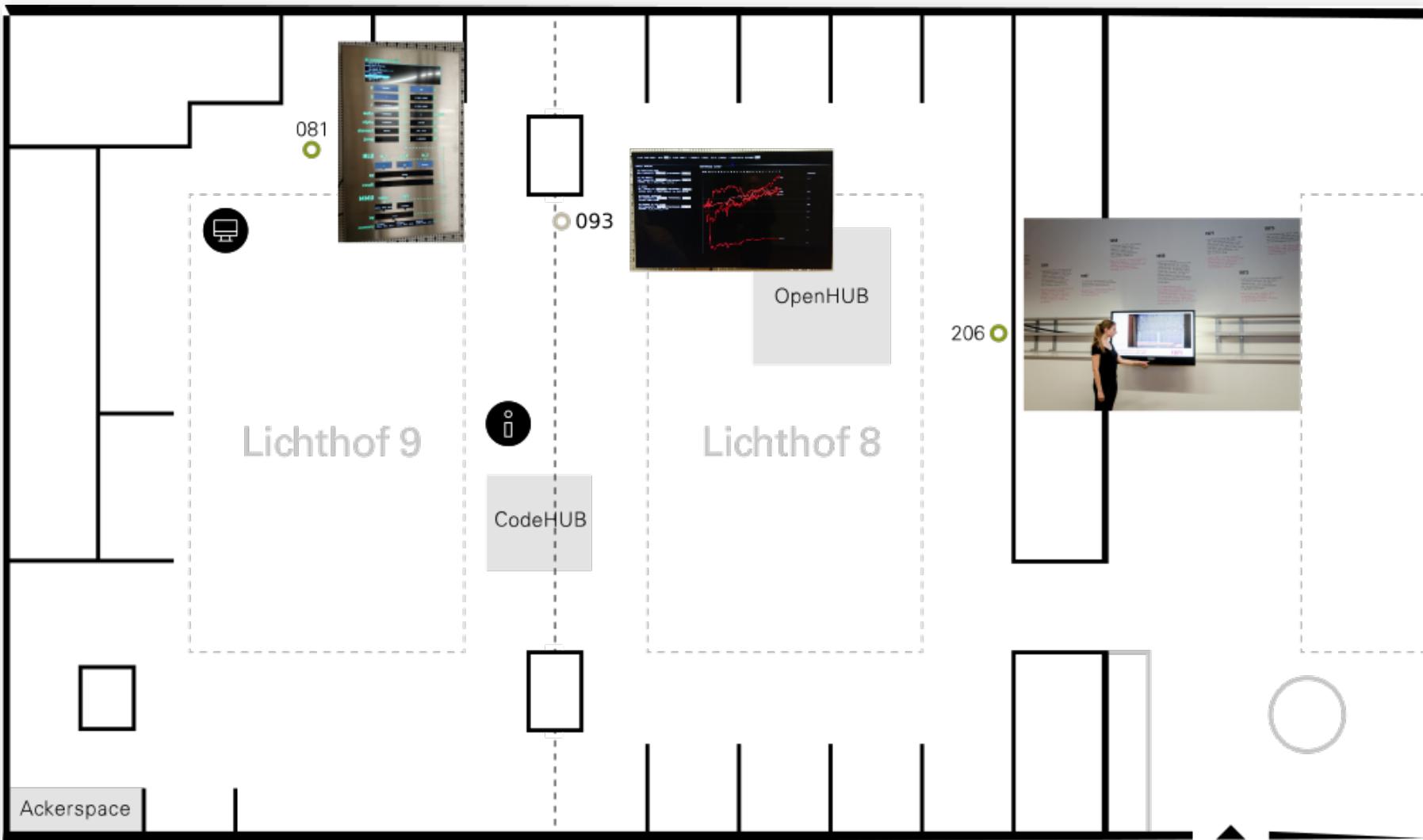
Hardware



Netzwerk



# EXPONATE





# SYMBOLE

## Bilder

Symbole für Objekte



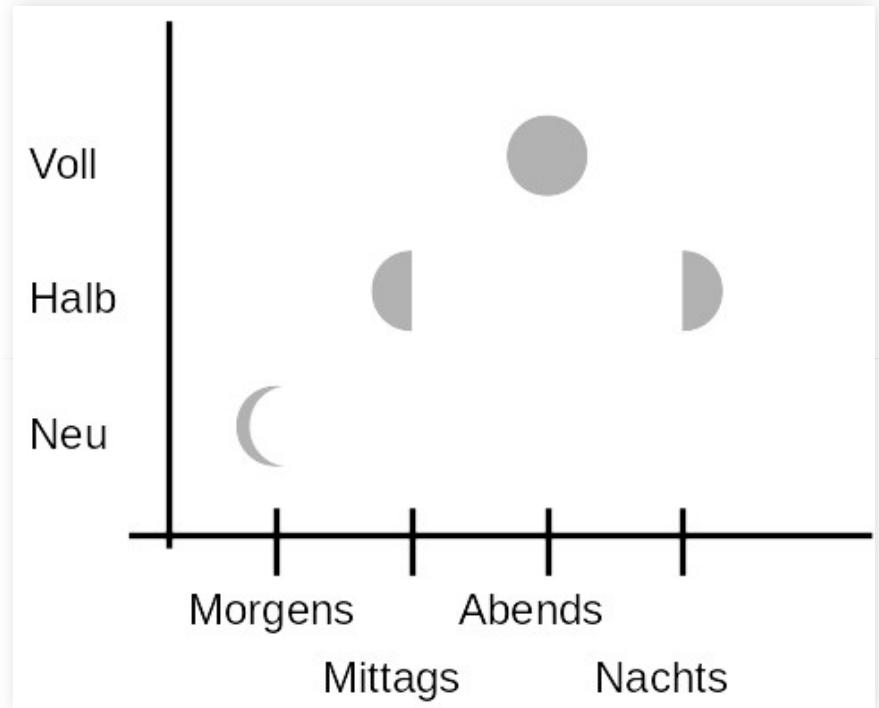
## Zahlen

Symbole für Regeln





# ANALOG -> DISKRET

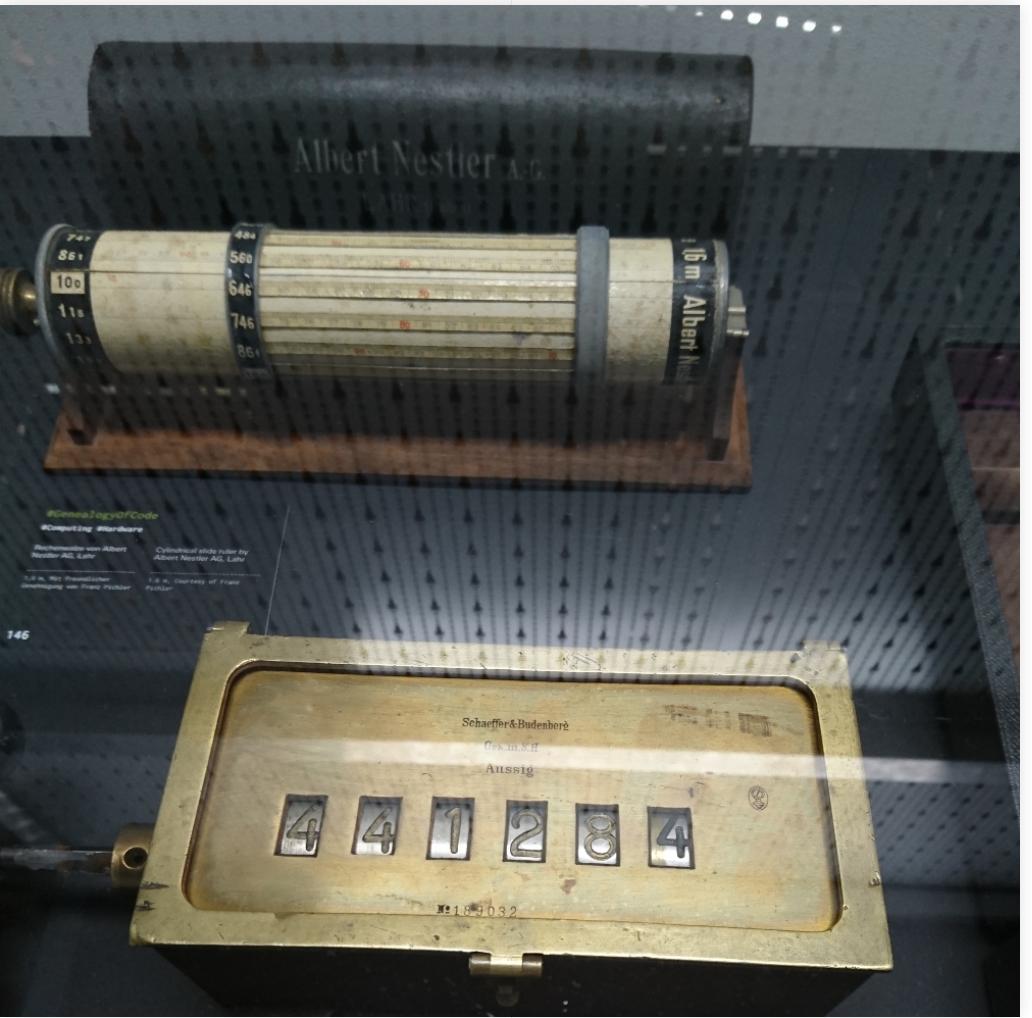




Diskrete Werte  
Nachvollziehbar  
Wiederholbar  
Nicht exakt (=> Fehler)



# DIGITAL: MIT ZAHLEN





# ZAHLSYSTEME

- 10-er System
  - 1-er, 10-er, 100-er
- Andere Systeme
  - 5-er, 12-er, 60-er
- Dual System
  - Leibniz, 1703
  - 0 und 1 reicht

$1+1=1$   
alles ist eins



$10^x$	Tabulag	ita	stabit
1	1	2	$2^0$
10	4	4	$2^1$
100	8	8	$2^2$
1000	16	16	$2^3$
10000	32	32	$2^4$
100000	64	64	$2^5$
1000000	128	128	$2^6$
10000000	256	256	$2^7$
100000000	512	512	$2^8$
1000000000	1024	1024	$2^9$



# BINÄR (DUAL) SYSTEM

Zweierpotenzen statt Zehnerpotenzen

1, 10, 100, 1000, ... => 1, 2, 4, 8, ...

Potenz	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Wert	32	16	8	4	2	1

Siebzehn plus Vier

	( $10^1$ $10^0$ )	( $2^5$ ... $2^0$ )
+ XVII	17	01 0001
+ IV	+ 4	+ 00 0100
-----	-----	-----
XXI	21	01 0101
	( $10^1$ $10^0$ )	( $2^5$ ... $2^0$ )
+ XVII	17	01 0001
+ IV	+ 4	+ 00 0100
-----	-----	-----
XXI	21	01 0101
+ 9	+ 9	+ 00 1001
-----	-----	-----
30	30	01 1110



# MAL AUSPROBIEREN

Wieviel Binär-Stellen für 100, 1000, 1000.000?

Ihr Geburtsjahr binär?

63 + 37 binär?

63 = 0011 1111

37 = 0010 0101

8 Stellen: 8Bit

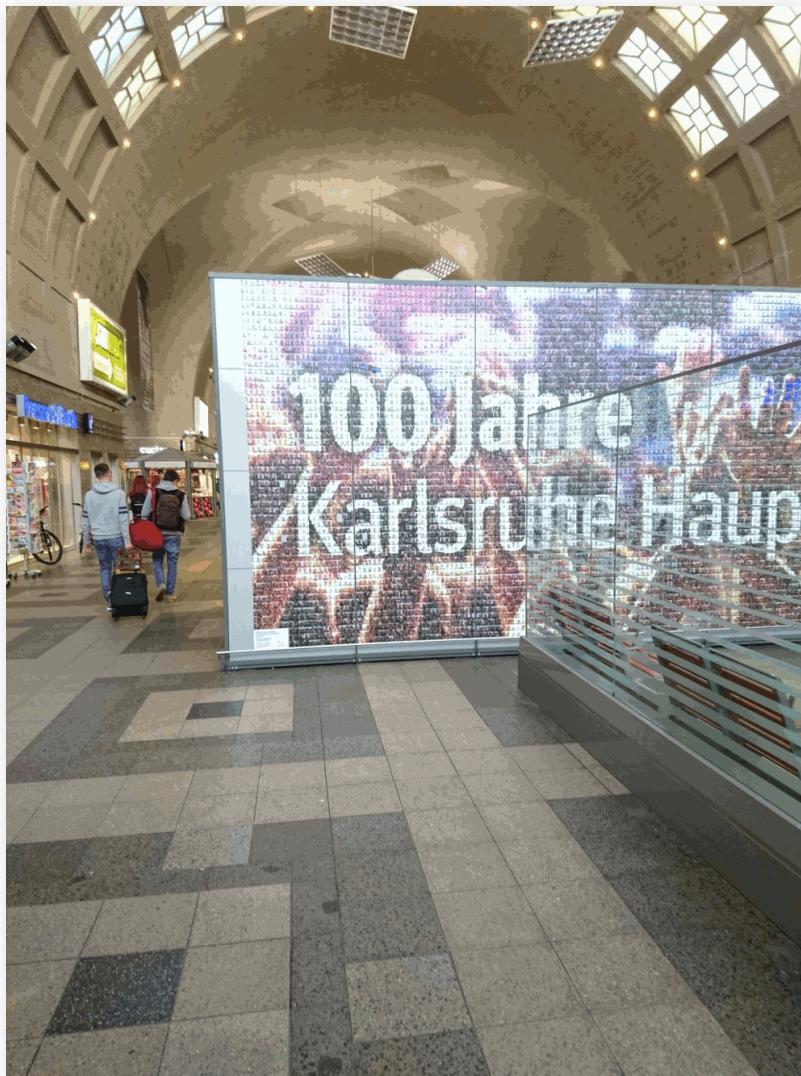
Wie weit kann man mit 8 Bit zählen?

# AUFLÖSUNG: REALITÄT





# AUFLÖSUNG: DETAILS





# AUSSAGENLOGIK

Aussagen sind entweder wahr oder falsch

- A: 9 ist durch 3 teilbar (wahr)
- B: 8 ist eine Primzahl (falsch)

Regeln, Bedingungen, Folgerungen ....

Bool'sche Algebra, 1847



# BINÄRE LOGIK

- Zahlen: Symbole für Wahrheitswerte
  - Wahr = 1
  - Falsch = 0
- Logische Operationen  
=> Rechenoperationen
  - A **UND** B => A \* B
  - A **ODER** B => A + B
- Negation
  - $A * \sim A = 0$
  - $A + \sim A = 1$

## Wahrheitstabellen

B	A	X
0	0	0
0	1	0
1	0	1
1	1	0

$$X = B * \sim A$$



# LOGIK IM ALLTAG

## Bewässerung

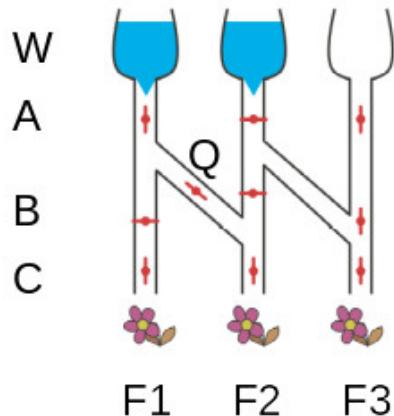
Wenn das Ventil zu ist,  
fließt kein Wasser.



Wenn das Ventil offen ist,  
fließt Wasser durch.



Welche der drei durstigen Blumen bekommen Wasser bei dieser Stellung?



$$F_1 = W_1 * A_1 * B_1 * C_1$$

$$F_2 = W_2 * A_2 * B_2 * C_2 + W_1 * A_1 * Q * C_2$$

$$F_3 = W_3 * A_3 * B_3 * C_3 + W_2 * A_2 * C_3$$

F1: 0 (kein Wasser)

F2: 1 (Wasser)

F3: 0 (kein Wasser)

Warum?

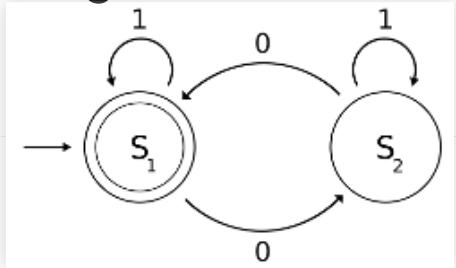


# LOGISCHE PLANUNG

## ENDLICHE AUTOMATEN

Zustand => Ausgabe

Eingabe => Zustandswechsel



I	S	S <sub>neu</sub>
0	S1	S2
0	S2	S1
1	S1	S1
1	S2	S2

## FLUSSDIAGRAMME

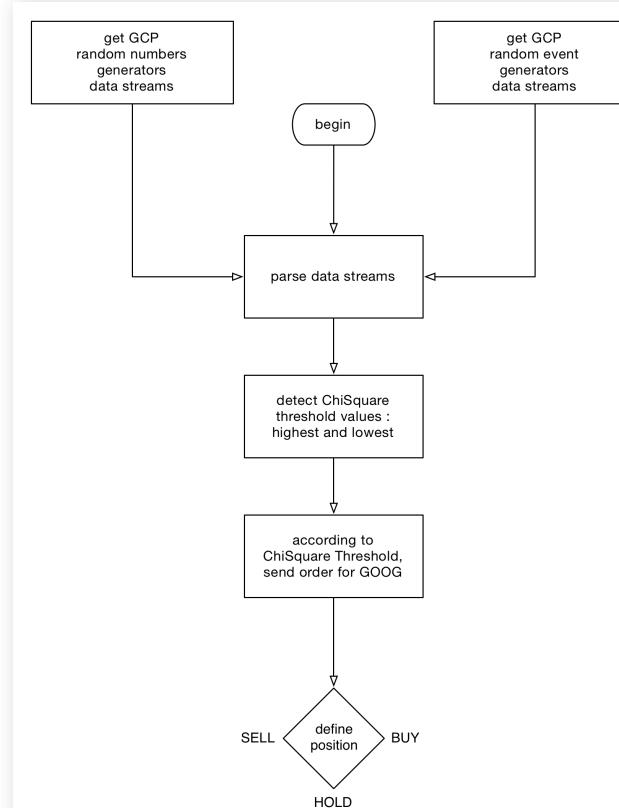


Fig. X1.7 : PSYCHIC INTERFACE, 2016



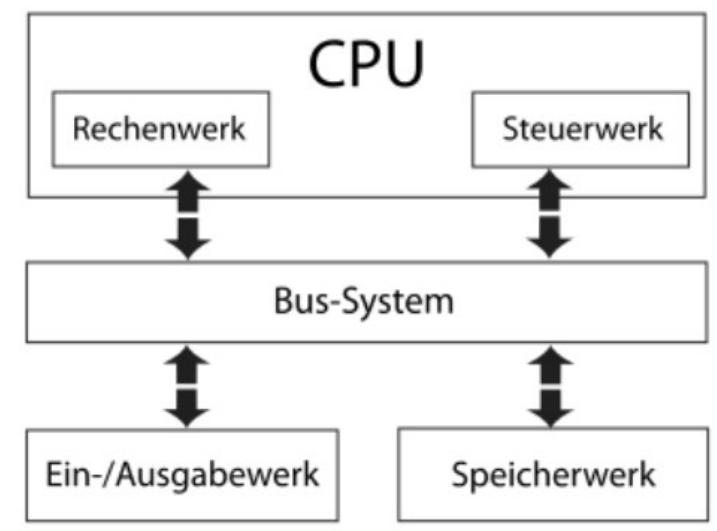
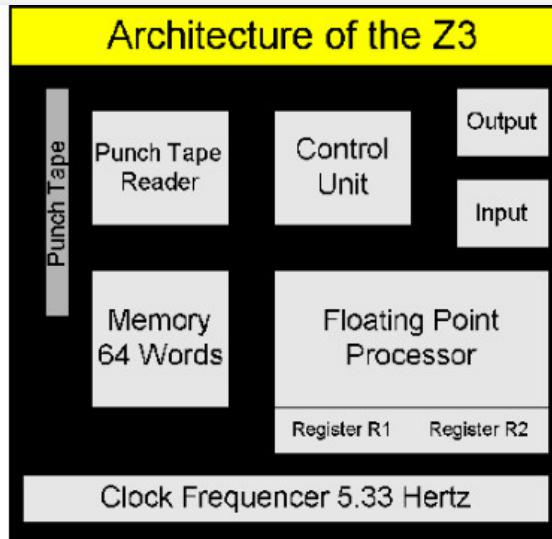
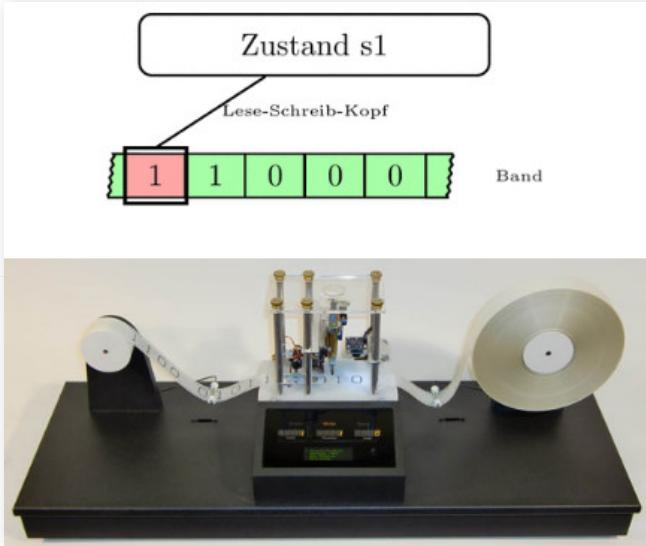
# MAL AUSPROBIEREN

- Flussdiagramm
  - Rechteck: Aktion
    - Eingabe
    - Ausgabe
    - Rechnen, ...
  - Rauten: Kontrolle
    - Bedingung abfragen
    - Verzweigen
- Ein Beispiel aus dem Alltag ...



# ARCHITEKTUR DIGITALER RECHNER

## TURING, ZUSE, VON NEUMANN



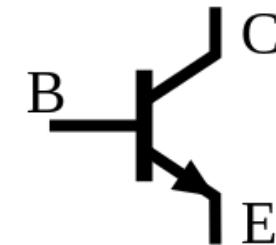
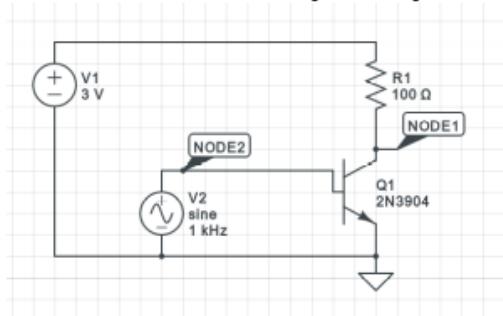
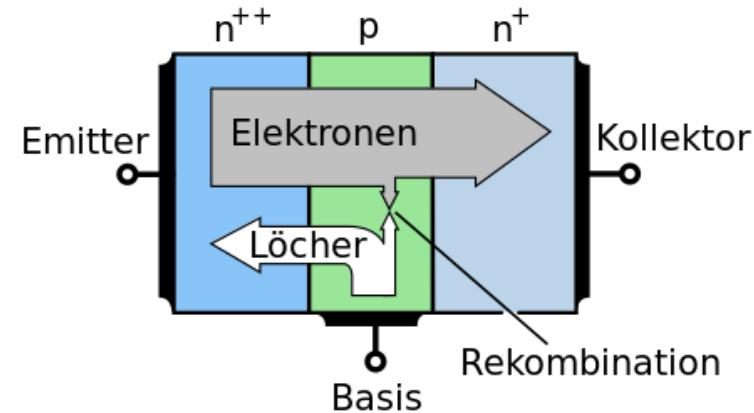


# ENIAC, 1945



# Elektronik: Der Transistor, 1947

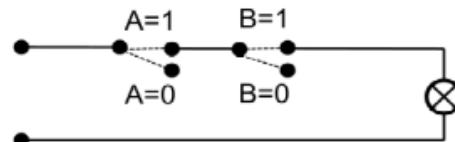
- Halbleiter: Trick (Dotierung)
- Verstärker: analog
- Schalter: digital
  - An und aus => 0 und 1
- Logische Schaltungen
  - NAND (nicht-UND)
  - NOR (nicht-Oder)
  - Inverter (Not)



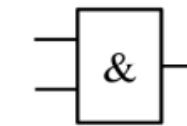


# Schaltalgebra/Gatterlogik

UND

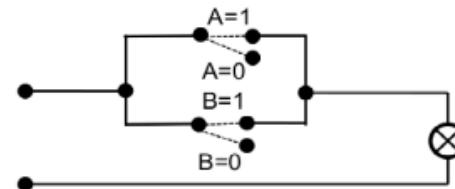


UND

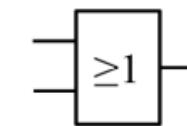


Gatter/  
Gates

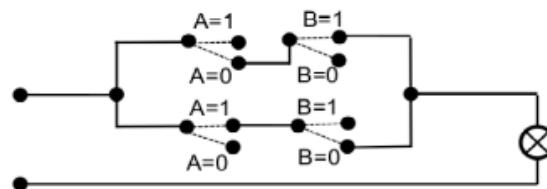
ODER



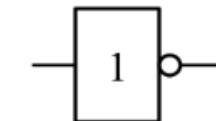
ODER



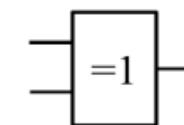
XOR-Funktion (exklusives ODER)



NEGATION



XOR

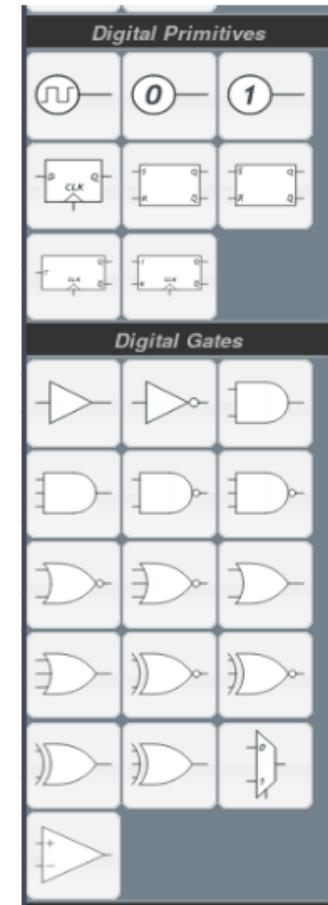


NAND oder NOR alleine reichen aus, um ALLE möglichen Schaltungen zu realisieren!



# Komplexere Grundschaltungen

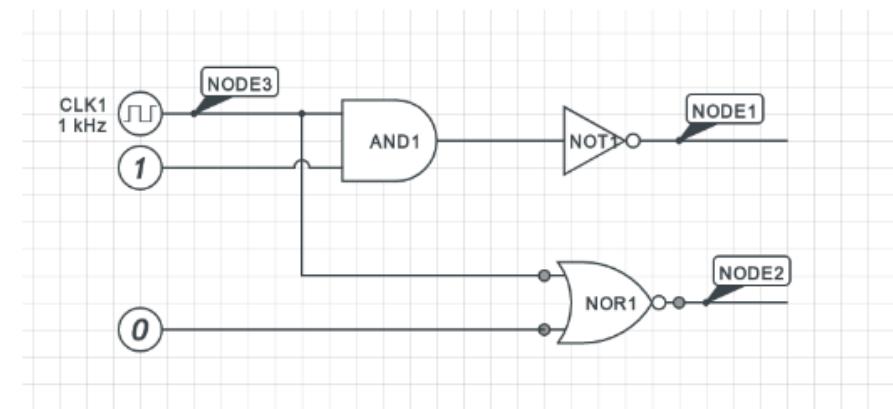
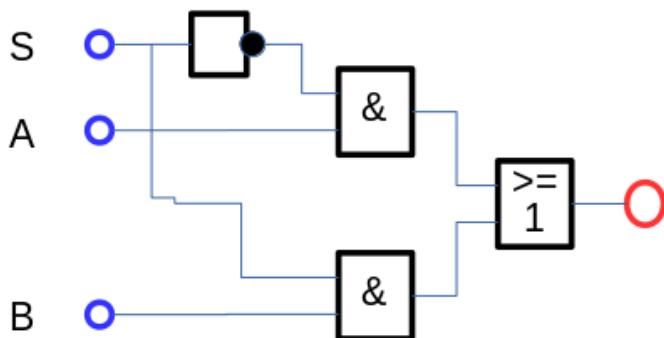
- Datenselektoren/Multiplexer
  - Verbindet genau eins von mehreren Eingangssignalen mit dem (einzigen) Ausgang
  - Zusätzliche Auswahlsignale
- Dekoder
  - Aktiviert genau einen von mehreren Ausgängen
  - Zusätzliche Auswahlsignale
- Zähler
- Addierer
- Speicherelement (Flip-Flop, Register)





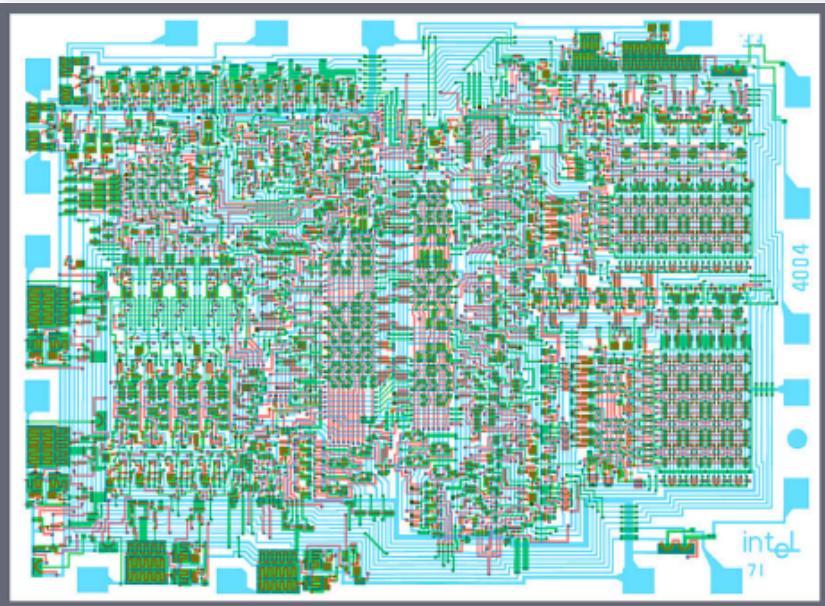
# ... mal ausprobieren ...

- 1 aus 2 Selektor
  - 2 Dateneingänge A, B
  - 1 Selektionseingang S
  - 1 Datenausgang Q
  - S=0: A  $\Rightarrow$  Q
  - S=1: B  $\Rightarrow$  Q
- Wahrheitstabelle?
- Mögliche Schaltung mit Gattern?
- <https://simulator.io>
- Oder
- <https://www.circuitlab.com/editor>
- Oder Papier + Stift ...



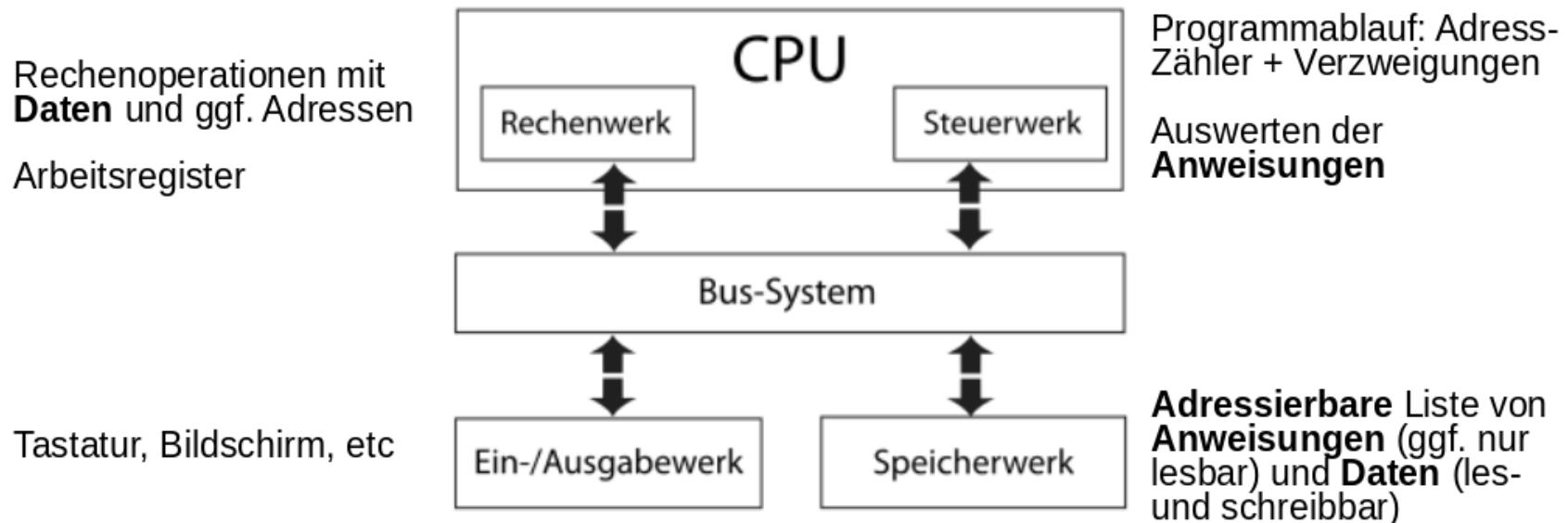


# MICROPROZESSOR ( $\mu$ P): INTEL 4004, 1971





# Nochmal µP-Architektur, Intel et al.





# Zahlen als ...

- Programmadresse
  - Adressarithmetik
- Daten
  - Beliebige Operationen
  - Kommt auf die Daten an
- Anweisung
  - Symbole
  - Berechnungen nicht sinnvoll
  - 2 Teile
    - Kontroll-Teil, Daten-Teil
- Anweisungen
  - Programm-Adresskontrolle
    - Nächste Adresse
    - Verzweigungsadresse
  - Daten-Adresskontrolle
  - Datenquelle für Arbeitsregister (AKKU)
  - Datum für Arbeitsregister
  - Datensenke für Arbeitsregister
  - Kontrolle für Rechenwerk (ALU)



# Buchstaben

# ... DATEN ...

# Zahlen

# Bild/Ton

**OPEN CODES DIE WELT VERSTEHEN, IN DER WIR LEBEN. DIE WELT VERSTEHEN, VON DER WIR LEBEN. DIE WELT VERSTEHEN , DIE WIR BEWOHNEN.** Wir leben heute in einer globalisierten Welt, die von digitalen Codes kontrolliert und erzeugt wird. Von der Kommunikation bis zum Transport (von Menschen, Gütern und Nachrichten) wird alles durch digitale Codes bestimmt. Die Mathematik und Elektronik haben eine neue, auf Computerprogrammen basierende Sprache entwickelt, die PhysikerInnen und PhysikerInnen und InformatikerInnen gestaltet werden will. Von Leibniz' Binärcode bis zum Morsecode, vom kosmischen Code bis zum genetischen Code – wir leben in einer Welt aus Codes. Die Ausstellung »Open Codes. Leben in digitalen Welten«, die künstlerische und wissenschaftliche Arbeiten versammelt, eröffnet uns die Möglichkeit, diese Welt zu verstehen. In einem neuen Ausstellungsformat wird die Ausstellung zeitgenössische Kunst und Wissenschaft präsentieren und einen neuen Wechselraum für BesucherInnen ermöglichen. Sie kann ein Meeting auf Labor und Lounge, aus Lernumgebung und »Club Méditerranée« sein – und das bei freiem Eintritt! In Museum = einem Ort der bezahlten Bürgerbildung – soll die Aneignung von Wissen belohnt werden. Denn die eigentliche Botschaft des digitalen Wandels läuft doch: Die Welt von morgen wird sich von einer Arbeits- zu einer Wissensgesellschaft verwandeln. Mit Ihnen, liebe BesucherInnen, wollen wir neue, vielleicht auch ungewöhnliche Formate von Bildung und Lernen erproben und in einer gelösten und das Lernen anregendem Atmosphäre neuen Zugang zur Wissen suchen. Für unsere Ausstellung »Open Codes. Leben in digitalen Welten« sollen für alle zugänglich gestaltet werden. Jede soll es ermöglicht werden, das »Dahinter« unserer heutigen digitalen Welt zu verstehen. Die Ausstellung präsentiert Kunstwerke und wissenschaftliche Arbeiten, die sowohl auf analogen wie auch auf digitalen Codes basieren. Sie versuchen die komplexen Dynamiken von Codes zu erklären und wie diese zunehmend die Art und Weise, wie wir leben und wie wir die Welt sehen, ge-

stalteten. Gemeinsam mit unseren BesucherInnen wollen wir fragen: Wollen wir die Welt verstehen, in der wir leben? Die Ausstellung wird von einem umfangreichen Rahmenprogramm, wie Vorträgen und Diskussionen, Workshops und Filmvorführungen begleitet, zu Beginn der Ausstellungslaufzeit organisiert das ZKM gemeinsam mit dem Europarat am Freitag, den 20. Oktober 2017 die Veranstaltung „4th Council of Europe Platform Exchange on Culture and Digitisation“ zum Thema „Empowering Democracy through Culture – Digital Tools for Culturally Competent Citizens“. Im Rahmen der Konferenz „Digitale Souveränität“ am Donnerstag, 2. November 2017 werden hochrangige VertreterInnen aus Wissenschaft, T, Wirtschaft und Politik über die Sensibilität digitaler Spuren in der heutigen Gesellschaft diskutieren.

# Direkte Kodierung =>

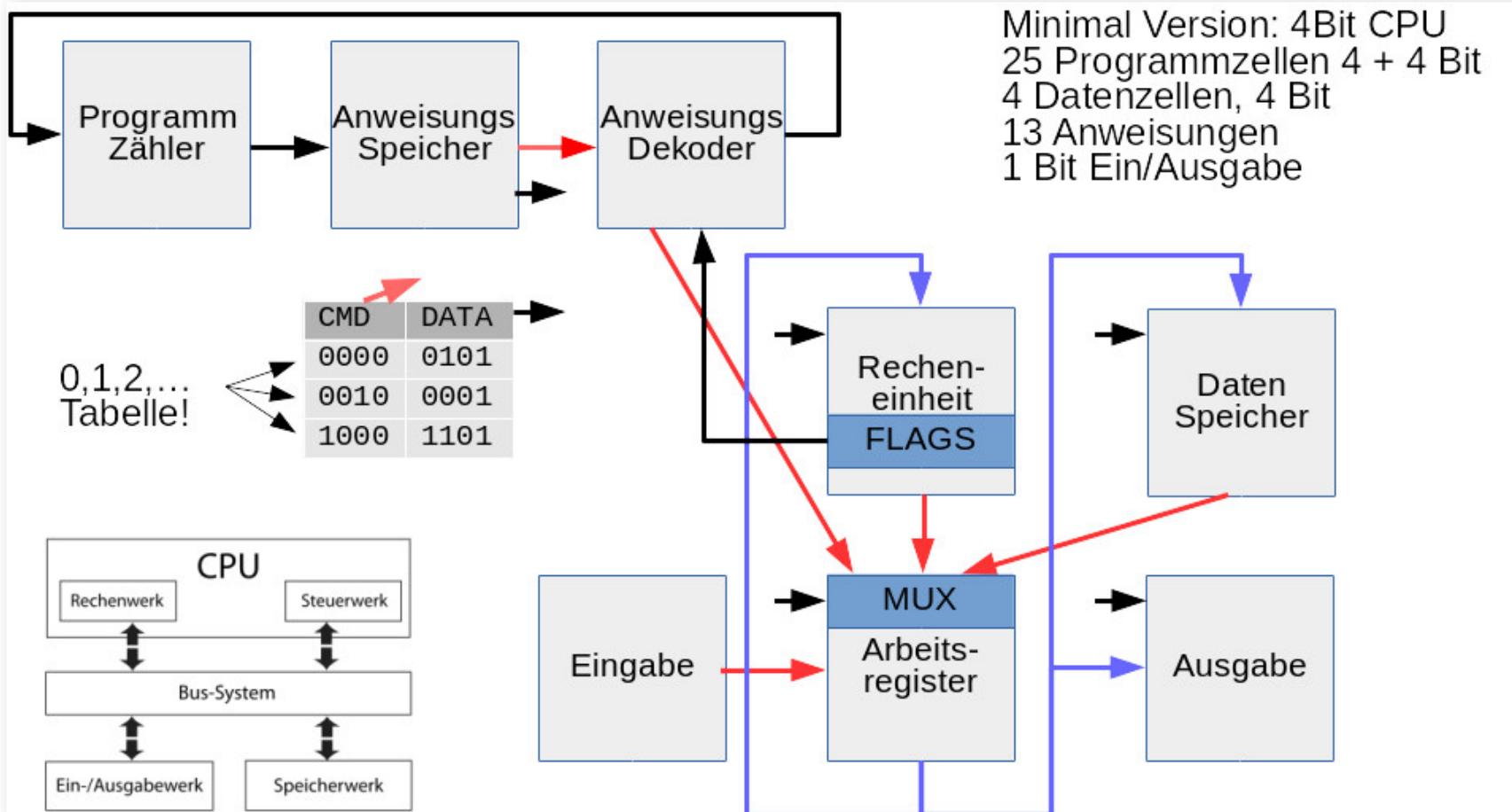
## Sprachkodierung =>



# ... ANWEISUNGEN

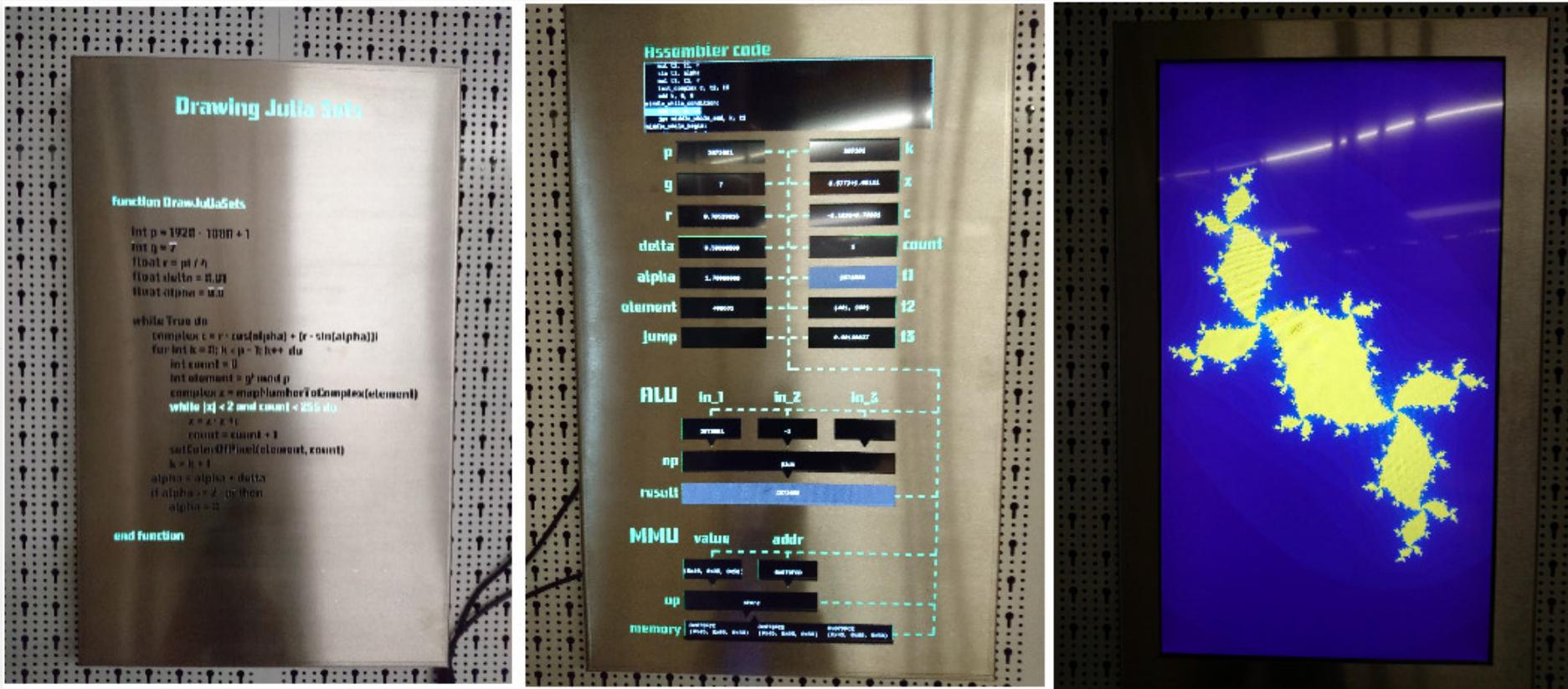


# UND NOCHMAL µP-ARCHITEKTUR (4 BIT)





# μP KUNST





# SUPER-EINFACHER BEFEHLSSATZ

Befehl	Programmzähler	Code	Abkürzung	Datum
Sprung	Auf Datum	0	JMP	Sprungadresse
Sprung, wenn Z=1	Auf Datum; sonst + 1	1	JZ	Sprungadresse
Sprung, wenn Z=0	Auf Datum; sonst + 1	2	JNZ	Sprungadresse
Addiere zu W	`+1	3	ADD	Operand
Subtrahiere von W	`+1	4	SUB	Operand
Lade in W	`+1	5	LD	Operand
Lade W von Port	`+1	6	IN	NA (implizit 0)
Schreibe W nach Port	`+1	7	OUT	NA (implizit 0)
Lade W von RAM#	`+1	8	RD	RAM Adresse
Schreibe W nach RAM#	`+1	9	WR	RAM Adresse

Das Z-Flag zeigt an, ob der Inhalt des Arbeitsregisters W gleich 0 ist ( $W=0 \Rightarrow Z=1$ )



# BEFEHLS-BITS

Funktion	Bits	Optionen
Adresse	1	+1 oder Sprung
W-Mux	3	W,ALU,Operand,RAM,Port
ALU	1	<u>ADD, SUB</u>
RAM	1	Schreiben
Port	1	Schreiben
<b>Summe</b>	<b>7</b>	

Nur 10 Befehle: Kodierung mit 4 Bit möglich



# SUPER-EINFACHER MIKROPROZESSOR

<http://digital-codes.de/zkm/controller.htm>

4-Bit Microprocessor Emulator

Instruction list:

JMP: 0, JZ: 1, JNZ: 2, ADD: 3, SUB: 4, LD: 5, IN: 6, OUT: 7, RD: 8, WR: 9

---

JMP: Jump, JZ: Jump if Zero, JNZ: Jump if not Zero, ADD: Add data to W, SUB: Sub data from w, LD: Load W, IN: Port input to W, OUT: Port output from W, RD: Read RAM, WR: Write RAM

---

Program Control:

Input Port Control:

Address	Program Memory			Instr	Variables	Input Port
	ADDR	INSTR	DATA			
0	0	6	0	JMP	0	OFF
1	7	0	0	Data	0	Output Port
2	0	0	0	0000	0	0
3				ALU	0	
4				NOP	0	
5				Z-Flag	Variable View	0
				1		



# BEDIENUNG

RUN/STOP: Programmlauf

RESET: Programmzähler auf 0 setzen

SET/CLR: Eingabebit auf 1/0 (ON/OFF)

Program Memory		
ADDR	INSTR	DATA
0	9	0
1	1	0
2	8	1
3	10	0
4	8	0
5	10	0
6	9	0
7	2	6
8	0	0
9		

# MAL AUSPROBIEREN

## PROGRAMMSCHLEIFEN

Eine Variable mit allen Werten beschreiben

Eingabe abfragen, Ausgang auf 1 wenn ON, sonst auf 0

Eingabe abfragen, 3 mal Ausgabe „blitzen“, wenn ON, warten bis wieder auf OFF



# Maschinensprache

- Im Speicher: Zahlen als Kodierung für Befehle, Daten, Adressen
  - Für Menschen: sehr unhandlich
    - Mnemonics: Text-Symbole für Befehle
      - LD, JP, ADD, ...
    - Symbolische Adressen
      - Namen für Sprungziele und Variablen
    - Macros: Abkürzung für Anweisungsfolge, mit Parameter, z.B. BLINK(5)
      - Anweisungen werden INLINE kodiert
  - Übersetzung der Mnemonics in Maschinencodes: Assembler
- cntVar. 0 ; counter variable  
outVar. 1 ; output variable  
initVal. 0 ; initialization value  
chkVal. 1 ; value to check for on input port  
00 - "0101000000" : **START**: LDI 0 ; clear timer  
01 - "1001000001" : out tmlo  
02 - "1001000010" : out tmhi  
03 - "0101000000" : LDI initVal ; init value  
04 - "0111000001" : **X0**: ST outVar ; store variable  
05 - "1001000000" : out 0 ; send to port.  
06 - "1000000000" : **X2**: IN 0; ; read port  
07 - "1011000001" : ANDI chkVal ; check bit 0  
08 - "0010000110" : JZ x2 ; loop if 0  
09 - "0101001111" : LDI tdelay  
0d - "0111000000" : **X1**: ST cntVar ; store variable  
0e - "1000000011" : **X4**: IN tmst  
0f - "1011000001" : ANDI tout



# Wesentliche Konstrukte

- Variablen
  - Veränderbare Speicherzellen
- Konstanten
  - Vordefinierte, feste, Werte
- Diverse Bedingungen für Verzweigungen
  - Zero, Carry, ...
- Wiederholungen
- Unterprogramme
  - Wiederverwendbare Programmteile
  - Rückkehr an die Aufrufadresse + 1
  - Übergabe von Werten
- *Verändern von Programmzellen*

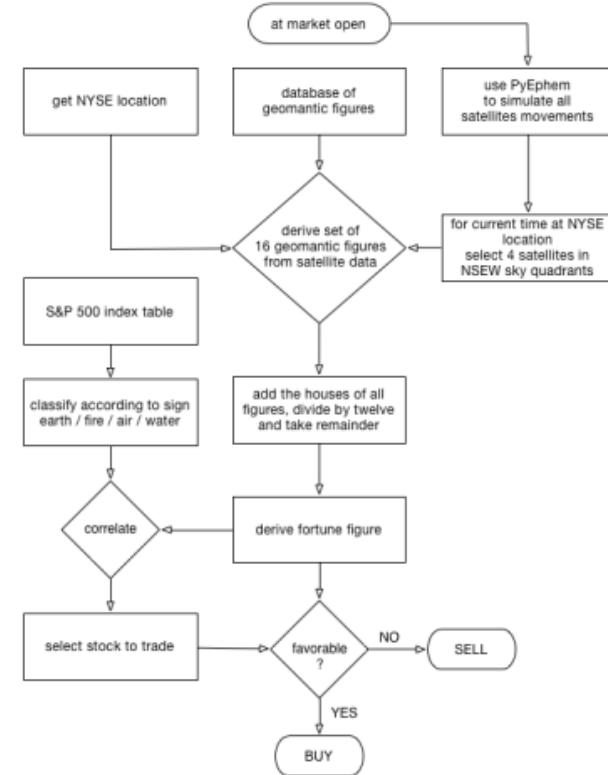


Fig. X1.3 : GPS-MANTIC, 2015



# PROGRAMMIEREN



# Selbst Programmieren ...

- Strukturieren der Aufgabe
- Festlegen von Zuständen und Abläufen
- Festlegen der Variablen (z.B. für Zähler)
- Umsetzen mit Programmiersprache
- Beispiel: Getränkeautomat
  - Geldeinwurf (3 Münzen)
  - Ausgabe
  - Abbruch mit Geldrückgabe

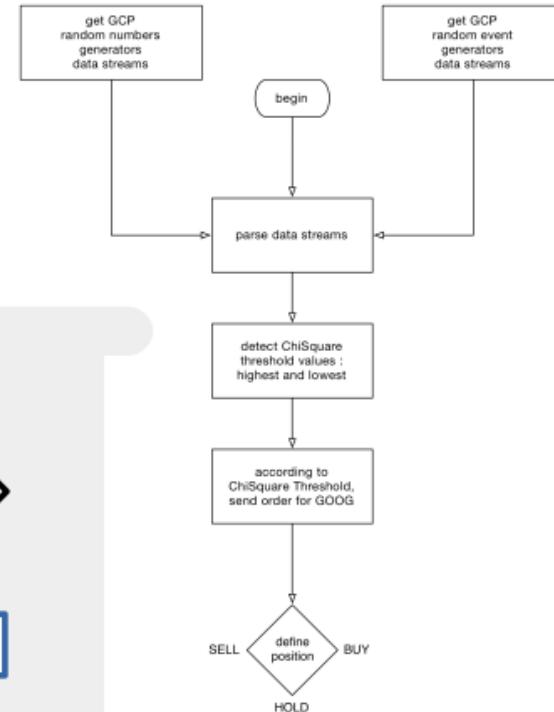
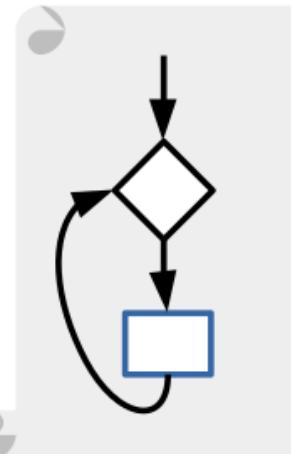
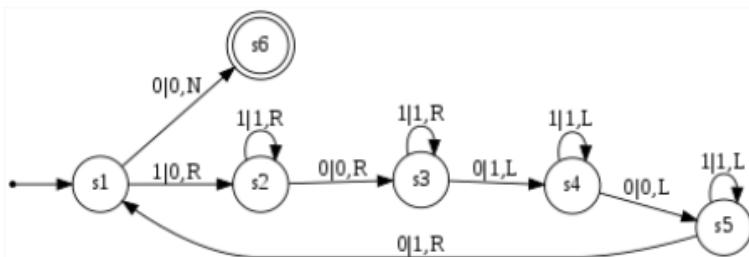


Fig. X1.7 : PSYCHIC INTERFACE, 2016



# HÖHERE PROGRAMMIERSPRACHEN

Große Auswahl ...

Javascript: im Browser, online ...

<http://digital-codes.de/zkm/jsonline.htm>

```
var a = "Open";
function zkm(a) {
    var d = "Codes";
    return a + d;
}
writeln(zkm(a));
```

Your work, select the text and copy it to an editor or email it to yourself.

Type JavaScript Examples:	Run (Ctrl-m)	Output	Timing: 0.001 s
Maximum element ▾			
<pre>var a = "Open";  function zkm(a) {     var d = "Codes";     return a + d; }  writeln(zkm(a));</pre>		OpenCodes	



# MAL AUSPROBIEREN

```
function istTeilbar(x,y) {  
    var rest = x % y; // Kommentar: % ist Rest (Modulo)  
    var ergebnis;  
    if (rest == 0) { ergebnis = "Teilbar";  
    } else { ergebnis ="Nicht Teilbar"; }  
    return ergebnis;  
}  
for (i = 1; i < 10; i++){  
    for (j=2; j<6;j++){  
        writeln("Ergebnis:" + i + " / " + j + ":" + istTeilbar(i,j));  
    }  
}
```

Flussdiagramm, Variablen, Funktionen, Grafik, ...





# SPRACHUMFANG UND INFOS

- Verschiedenen Datentypen
  - Zahlen, Text, zusammengesetzte Typen
- Übliche Operationen
  - Rechnen, Textoperationen, Ein-/Ausgabe
- Zugriff auf Funktionen des Systems
  - Graphische Anzeige, Datum/Zeit, Dateien, ...
- Erweiterbarkeit über Bibliotheken
  - Anwendungen, Spiele, eigene Funktionen
- Tutorial z.B.  
<https://www.w3schools.com/js/default.asp>



# BIBLIOTHEKEN

```

container.name = name;
container.add(elementMesh.clone());
container.scale.copy(v3(scale, scale, scale));
container.translateX(Xpos).translateY(Ypos);
container.length = container.children[0].length;
container.tween = new TWEEN.Tween();
return container;
}

function rotateTo(i = "a", time = 1000, start = true,
o = scene.getObjectByName("main"))
if (i == " ") o.visible = false;
else {
  o.tween.stop();
  let target = targetRotation(i).normalize();
  let start = o.quaternion.clone().normalize();
  let slerpI = {t: 0};
  o.tween = new TWEEN.Tween(slerpI).to({t: 1}, time)
    .interpolation(TWEEN.Interpolation.Bezier)
    .onUpdate(function(){
      THREE.Quaternion.slerp(start, target,
o.quaternion, Math.round(1000 * slerpI.t) / 1000);
    });
  if(start) o.tween.start();
}

function targetRotation(l) {
  let e = new THREE.Euler(D2r(abcP[l].x),
D2r(abcP[l].y), D2r(abcP[l].z), "XYZ");
  return new THREE.Quaternion().setFromEuler(e);
}

function updateLetter(letter) {
  if (alphabet.indexOf(letter) >= 0) {
    let matrixObject = scene.getObjectByName("matrix");
    let lPos = letter.charCodeAt(0) - "a".charCodeAt(0);
    blink(matrixObject.children[lPos], 500);
    let sum = 0;
    for (let j = 0; j < 4; j++) {
      sum += matrixObject.children[lPos].children[j].rotation.y;
    }
    matrixObject.children[lPos].rotation.y = sum / 4;
  }
}

function setIntia123 = B log2  $\left( \frac{S}{N} \right)$ 
"matrix",  $\frac{ti-d}{n} + d$   $\sum_{j=1, \dots, n}^{} \frac{PR_j}{c_j}$ 
var  $a_i = \frac{i-1}{n} + d$   $R_{xy}(\tau) = (x * y)(\tau) = \int_{-\infty}^{\infty} x(\tau) y(\tau + \tau)d\tau$ 

```



# 3D-GRAFIK (THREEJS)

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, w/h, 0.1, 1000 );
var renderer = new THREE.WebGLRenderer({ alpha: true });
document.body.appendChild( renderer.domElement );
var geometry = new THREE.BoxGeometry( 1, 1, 1 );
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
var material2 = new THREE.MeshBasicMaterial( { color: 0xff0000 } );
var cube = new THREE.Mesh( geometry, material );
var cube2 = new THREE.Mesh( geometry, material2 );
scene.add( cube ); scene.add( cube2 ); var l = false;
camera.position.z = 5; var c2x = 1.0; cube2.position.x = c2x;
var animate = function () { requestAnimationFrame( animate );
```



# ONLINE JAVASCRIPT MIT JSFIDDLE

<https://jsfiddle.net>

## Edit in JSFiddle

- [JavaScript](#)
- [HTML](#)
- [Result](#)

```
var gameOfLife = new terra.Terrarium(25, 25, {
  trails: 0.9, periodic: true, background: [22, 22, 22]
});
terra.registerCA({
  type: 'GoL',
  colorFn: function () { return this.alive ? this.color + ',1' : '0,0,0,0'; },
  process: function (neighbors, x, y) {
    var surrounding = neighbors.filter(function (spot) {
      return spot.creature.alive;
    }).length;
    this.alive = surrounding === 3 || surrounding === 2 && this.alive;
    return true;
  },
  function () {
    this.alive = Math.random() < 0.5;
  }
});
gameOfLife.grid = gameOfLife.makeGrid('GoL');
```

Beispiele:

<https://doc.jsfiddle.net/tutorial.html#first-fiddle-hello-world-goodbye-world>



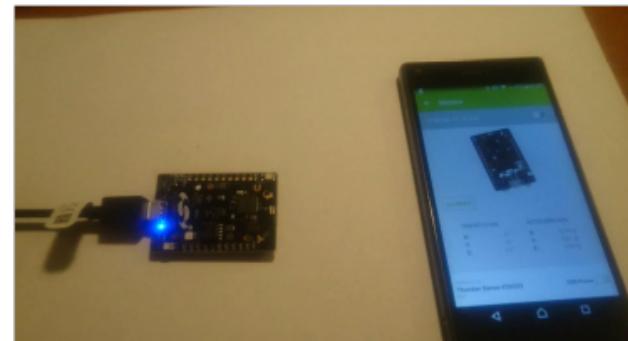
# MEHR SOFTWARE

- Alle Computer brauchen ein Betriebssystem
  - Linux, Android, Windows, MacOS, iOS
- Verwaltung der „Infrastruktur“
  - Internet, Display, Audio, Dateien, Touch/Maus, ...
  - Parallel Ausführung mehrerer Anwendungen (Programme)
    - Browser, Mails, Apps, Hintergunddienste, ...
  - (Un-)Sicherheit: Benutzerkonten, Speicherschutz
  - Wartungsaufgaben (Update)
- Weitere Programmiersprachen
  - Python (interpretiert. Entwicklung geht schneller)
  - C++ (compiliert. Programm läuft schneller)
- Spezielle Bibliotheken (Spiele, Wissenschaft,...)



# Hardware

- Übliche Computer
  - PC, Tablet
  - SW wie vor
- Smartphones
  - Android SDK
  - IOS SDK
- Embedded/IoT
  - Arduino
  - Raspberry
  - Sensor-Boards (div.)





# OpenSense

<https://sensebox.de/>

## senseBox:home



Bürger können mit der senseBox:home ihre eigenen lokalen Forschungsfragen stellen und die nötigen Daten selbst sammeln und sammeln lassen.

[Mehr Infos »](#)

## senseBox:edu



Für Schulen und Nachwuchsforscher gibt es die senseBox:edu als Experimentierkasten mit didaktischen Konzepten, Anleitungen und Projektideen.

[Mehr Infos »](#)

## openSenseMap



Die openSenseMap ist unsere Web-Plattform für die offenen Daten der senseBox. Hier werden die Daten visualisiert und jeder kann Analysen durchführen.

[Mehr Infos »](#)



# WIE WEITER?

- Programmierung
  - WWW Tutorials, Bücher
  - Entropia/CCC: <https://entropia.de>
  - Hackschool: <http://www.hackerstolz.de/hackschool/>
- Offene Daten
  - OKLabs: <https://okfn.de/>
- Hardware
  - Makerszene
    - Fablab KA: <https://fablab-karlsruhe.de/>
    - OKLabs, Sensebox: <https://sensebox.de/de/>
    - Board-Kosten ~ 10 – 100€
- Alles:
  - Meetups: <https://www.meetup.com/> (nach "tech" suchen)



# VERSCHLÜSSELUNG



Kein Video mit unterstütztem Format und MIME-Typ gefunden.



# OKLABS

<https://codefor.de/karlsruhe/>

