

# Rechnerarchitektur

Arithmetik I

Univ.-Prof. Dr.-Ing. Rainer Böhme

Wintersemester 2020/21 · 11. November 2020

# Gliederung heute

1. **Addition und Subtraktion**
2. Arithmetisch-logische Einheit

# Addieren von Binärzahlen

**Addition** positiver  $n$ -stelliger Binärzahlen  $a$  und  $b$  stellenweise von rechts nach links:

- An jeder Stelle  $i$  kann ein **Übertrag** (engl. *carry*)  $c_i = 1$  auftreten.
- Falls Summe  $y = a + b \geq 2^n$ , reichen  $n$  Bit nicht mehr für die Darstellung des Ergebnisses aus. Das  $(n + 1)$ -te Summenbit wird als **Überlauf** (engl. *overflow*) bezeichnet.

## Beispiele

(für  $n = 8$ )

$$\begin{array}{r} 0001\ 0111 \quad (23)_{10} \\ + 0101\ 0110 \quad (86)_{10} \\ \hline \end{array} \qquad \begin{array}{r} 0011\ 0111 \quad (55)_{10} \\ + 1101\ 0110 \quad (214)_{10} \\ \hline \end{array}$$

Übertrag („1 gemerkt“)  $\rightarrow$

$$\begin{array}{r} 1\ 11 \\ 0110\ 1101 \quad (109)_{10} \end{array} \qquad \begin{array}{r} 1\ 111\ 11 \\ 1\ 0000\ 1101 \quad (269)_{10} \end{array}$$

8-Bit-Summe

$$0000\ 1101 \quad (13)_{10}$$



# Vergleich am Zahlenstrahl

Binärdarstellung vorzeichenbehafteter Ganzzahlen mit  $n$  Stellen

## Wertebereiche

(Beispiele für  $n = 8$  Bit)

positive Binärzahlen

$0, 1, \dots, 2^{n-1} - 1, 2^{n-1}, \dots, 2^n - 1$   
0 1 127 128 255

Zweierkomplement

$-2^{n-1}, \dots, -1, 0, 1, \dots, 2^{n-1} - 1$   
-128 -1 0 1 127

## Arithmetische Negation in Zweierkomplementdarstellung

1. Invertiere alle Bits.  $(127)_{10} = (0111\ 1111)_2 \Rightarrow (1000\ 0000)_2 = (-128)_{10}$
2. Addiere 1.  $(-128)_{10} + 1 = (1000\ 0000)_2 + 1 = (1000\ 0001)_2 = (-127)_{10}$

# Fallunterscheidung

1. Zahlen  $a$  und  $b$  **positiv** (d. h. „Vorzeichenbits“  $a_{n-1} = b_{n-1} = 0$ )

Arithmetischer Überlauf bei  $y_{n-1} = 1$  ( $\Leftrightarrow c_{n-1} = 0$  und  $c_{n-2} = 1$ )

2. Zahlen  $a$  und  $b$  **negativ** (d. h. „Vorzeichenbits“  $a_{n-1} = b_{n-1} = 1$ )

$\Rightarrow a' = -a$  und  $b' = -b$  positiv, also gilt:

$$y' = a + b = (2^n - a') + (2^n - b') = 2 \cdot 2^n - (a' + b')$$

Das korrekte Ergebnis  $y = 2^n - (a' + b') = y' - 2^n$  wird durch Abschneiden und Ignorieren des Übertragsbits  $c_{n-1}$  erreicht.

Arithmetischer Überlauf bei  $y_{n-1} = 0$  ( $\Leftrightarrow c_{n-1} = 1$  und  $c_{n-2} = 0$ )

3. Vorzeichen von  $a$  und  $b$  **unterschiedlich** (z. B. sei  $b$  negativ)

$$y' = a + b = a + (2^n - b') = 2^n - (b' - a) \text{ ist korrekt für } |b| > |a|.$$

Das korrekte Ergebnis für  $|b| \leq |a|$  ist  $y = a - b' = y' - 2^n$ .

Auch hier: Abschneiden und Ignorieren des Übertragsbits  $c_{n-1}$ .

Kein arithmetischer Überlauf möglich !



# Hörsaalfrage



24 82 94 16

Welche Bitfolge entspricht der Zweierkomplement-Darstellung der Zahl  $-4$  bei  $n = 8$  Bit?

- 0000 1011
- 0000 1100
- 1011 1111
- 1101 1111
- 1111 1011
- **1111 1100** denn:  $0000\ 0100 \Rightarrow 1111\ 1011 + 1 = 1111\ 1100$

Zugang: <https://arsnova.uibk.ac.at> mit Zugangsschlüssel **24 82 94 16**. Oder scannen Sie den QR-Kode.

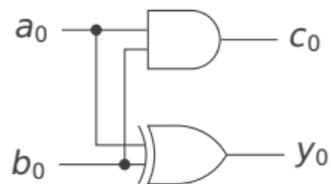
# Halbaddierer (engl. Half Adder)

- Ermittelt aus  $a_0$  und  $b_0$  Summe  $y_0$  und Übertrag  $c_0$
- Einsatz für niederwertigste Bits
- Verzögerung  $\tau$  für  $c_0$  und  $2\tau$  für  $y_0$

## Wahrheitstabelle

| $a_0$ | $b_0$ | $y_0$ | $c_0$ |
|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     |
| 0     | 1     | 1     | 0     |
| 1     | 0     | 1     | 0     |
| 1     | 1     | 0     | 1     |

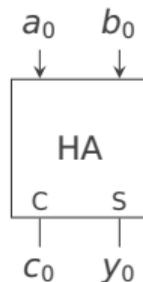
## Realisierung



$$c_0 = a_0 \cdot b_0$$

$$y_0 = a_0 \oplus b_0$$

## Symbol



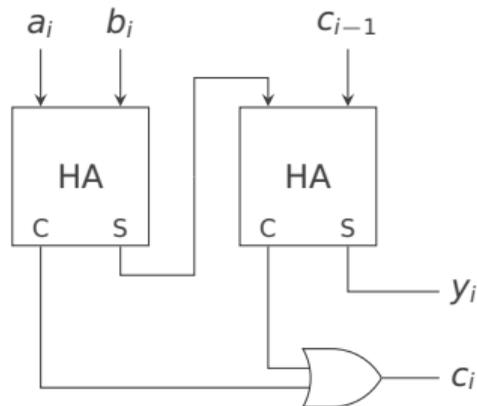
# Volladdierer (engl. Full Adder)

- Addiert  $a_i$ ,  $b_i$  und  $c_{i-1}$  an Bitpositionen  $i = 1, \dots, n - 1$ .
- Gibt Summe  $y_i$  und Übertrag  $c_i$  aus.

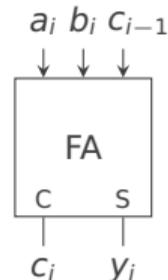
## Wahrheitstabelle

| $a_i$ | $b_i$ | $c_{i-1}$ | $y_i$ | $c_i$ |
|-------|-------|-----------|-------|-------|
| 0     | 0     | 0         | 0     | 0     |
| 0     | 0     | 1         | 1     | 0     |
| 0     | 1     | 0         | 1     | 0     |
| 0     | 1     | 1         | 0     | 1     |
| 1     | 0     | 0         | 1     | 0     |
| 1     | 0     | 1         | 0     | 1     |
| 1     | 1     | 0         | 0     | 1     |
| 1     | 1     | 1         | 1     | 1     |

## Realisierung



## Symbol

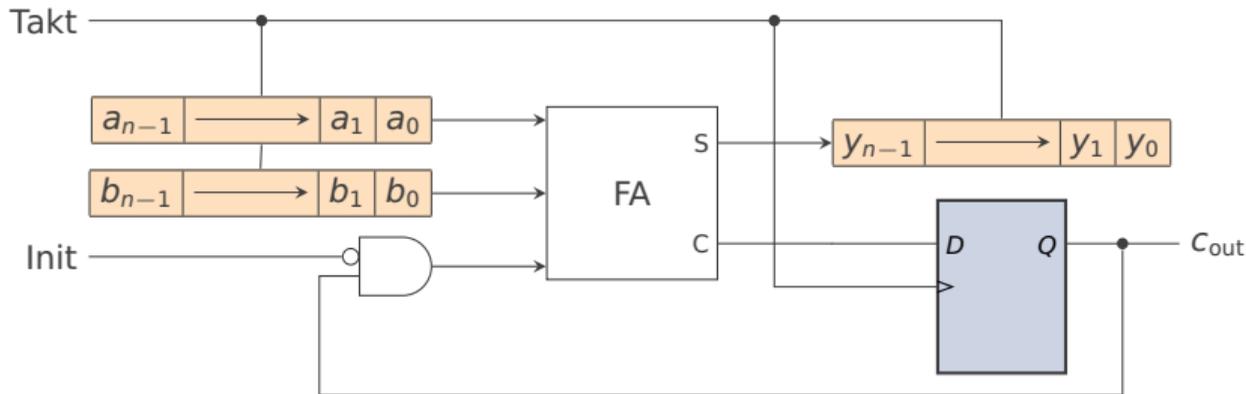


$$c_i = a_i \cdot b_i + a_i \cdot c_{i-1} + b_i \cdot c_{i-1}$$

$$y_i = a_i \oplus b_i \oplus c_{i-1}$$

# Serielles Addierwerk

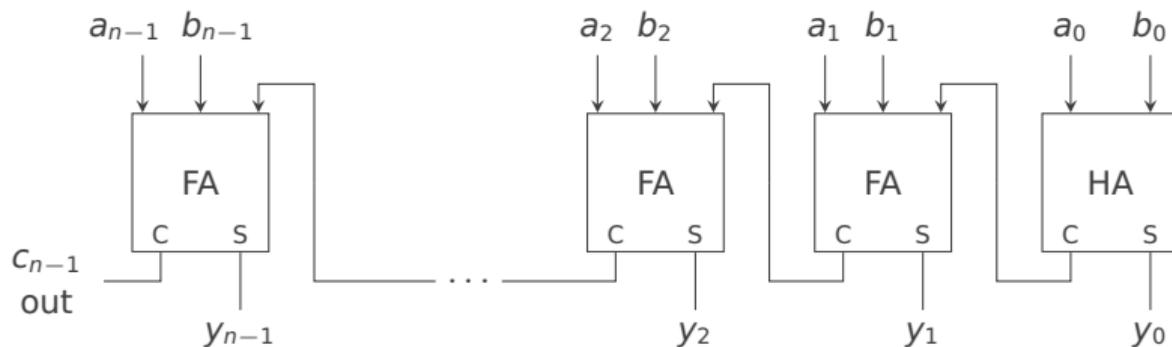
Konstruktion eines **synchronen Schaltwerks** aus einem Volladdierer, einem Flipflop und drei  $n$ -Bit-Schieberegistern:



- Der Init-Eingang dient zum Löschen des Übertrags, falls das Flipflop nicht initialisiert ist.
- In Takt  $t$  wird Ergebnisbit  $y_t$  aus  $a_t$ ,  $b_t$  und  $c_{t-1}$  bestimmt.
- Die Addition von zwei  $n$ -Bit-Zahlen benötigt  $n$  Taktzyklen.

# Paralleles Addierwerk

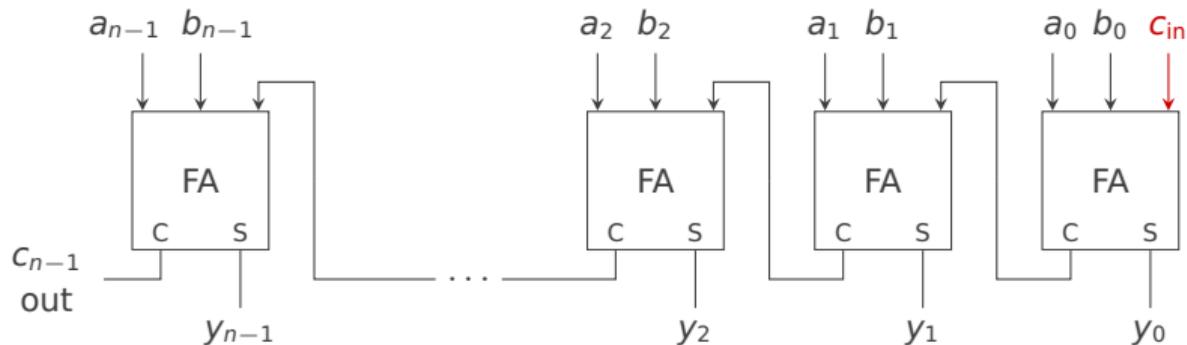
Konstruktion aus  $n - 1$  Voll- und einem Halbaddierer:



- Übertrag an Position  $i = 0$  kann alle Bitstellen 1 bis  $n - 1$  durchlaufen, daher: "Ripple Carry"-Addierer (RCA)
- Maximale Verzögerung:  $2n \cdot \tau$
- Verbesserung in der Praxis: "Carry Look-Ahead"-Addierer (CLA) addieren mit konstanter Verzögerung.

# Paralleles Addierwerk

Konstruktion aus  $n$  Volladdierern mit Übertrag-Eingang:



- Übertrag an Position  $i = 0$  kann alle Bitstellen 1 bis  $n - 1$  durchlaufen, daher: “Ripple Carry”-Addierer (RCA)
- Maximale Verzögerung:  $2n \cdot \tau$
- Verbesserung in der Praxis: “Carry Look-Ahead”-Addierer (CLA) addieren mit konstanter Verzögerung.

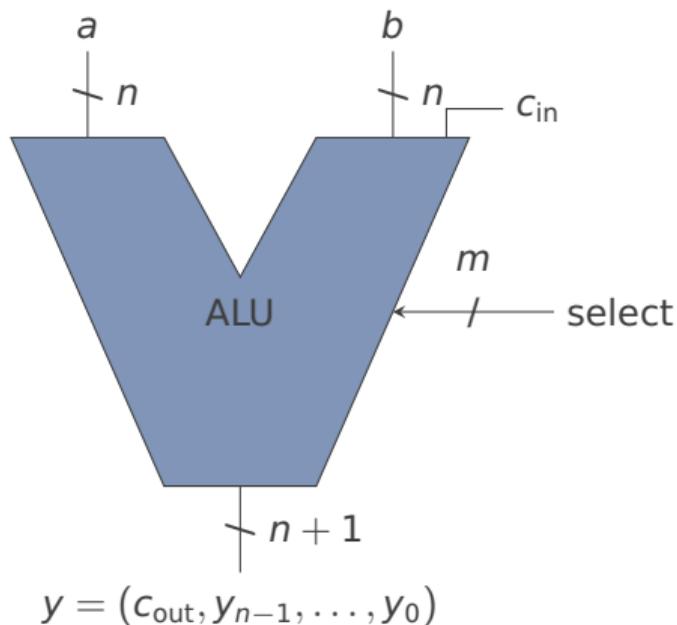


# Gliederung heute

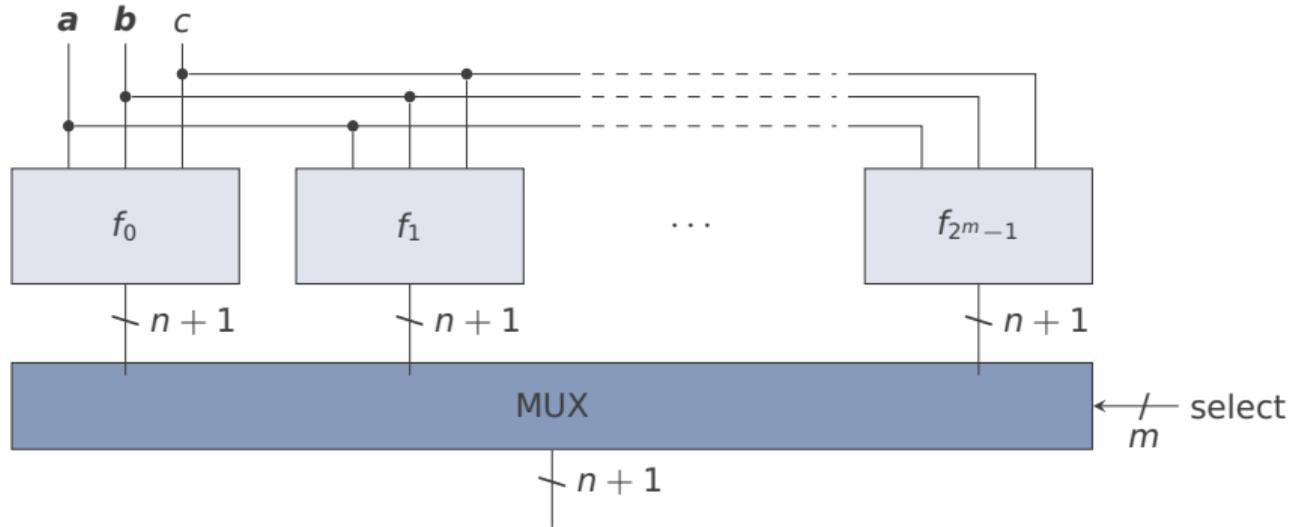
1. Addition und Subtraktion
2. **Arithmetisch-logische Einheit**

# Arithmetisch-logische Einheit (ALU)

Multifunktionsmodul für Verknüpfungen zwischen  $n$ -Bit-Registern



# Schematischer Schaltungsaufbau



Weitere Strukturierung der Select-Eingänge  $s_i$  sinnvoll, z. B.

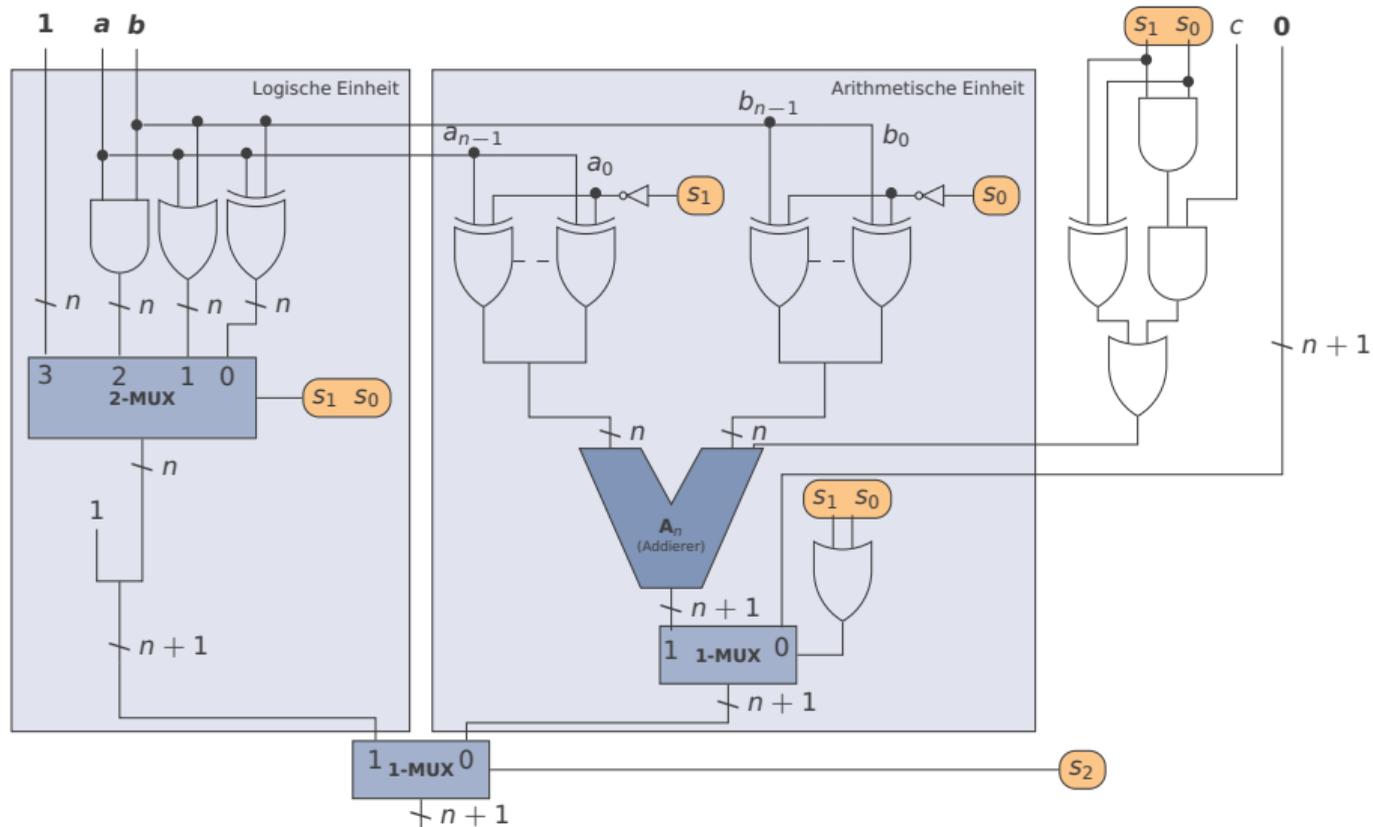
- $s_2$  entscheidet zwischen arithmetischen und logischen Operationen
- $s_1$  und  $s_0$  wählen konkrete (arithmetische oder logische) Operation

# Wahl der Operation

Beispiel für die Belegung des Select-Eingangs:

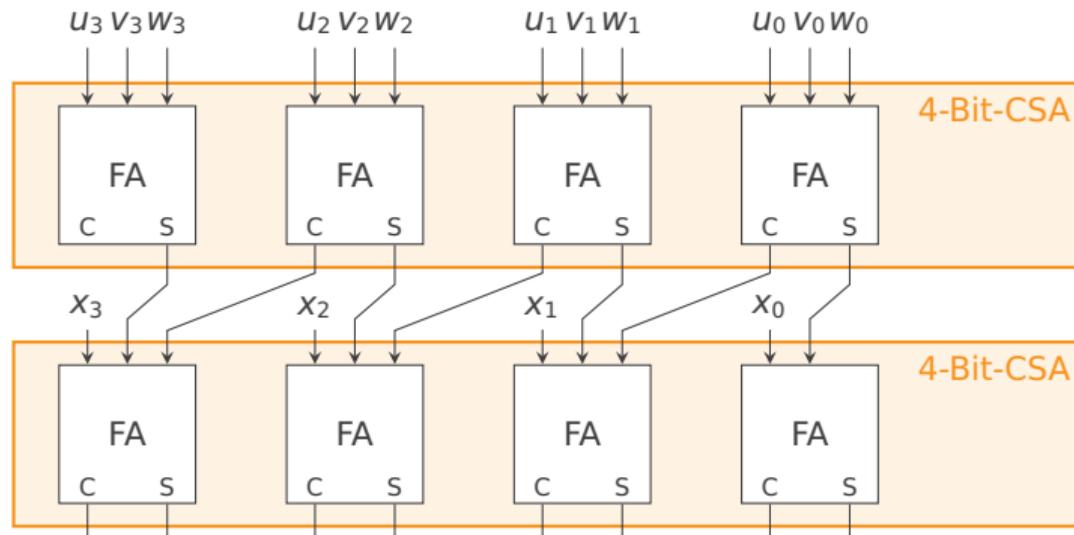
| $s_2$ | $s_1$ | $s_0$ | Funktion                       |
|-------|-------|-------|--------------------------------|
| 0     | 0     | 0     | <b>0</b>                       |
| 0     | 0     | 1     | <b><math>b - a</math></b>      |
| 0     | 1     | 0     | <b><math>a - b</math></b>      |
| 0     | 1     | 1     | <b><math>a + b + c</math></b>  |
| 1     | 0     | 0     | <b><math>a \oplus b</math></b> |
| 1     | 0     | 1     | <b><math>a \vee b</math></b>   |
| 1     | 1     | 0     | <b><math>a \wedge b</math></b> |
| 1     | 1     | 1     | <b>1</b>                       |

# Schaltungstechnische Realisierung einer ALU



# Ausblick: Carry-Save-Addierer (CSA)

Anordnung zur **partiellen Addition**: Ein CSA-Baustein integriert  $n$  Volladdierer mit **separaten** Carry-Ausgängen



→ Kaskadierung zur schnellen Addition mehrerer Summanden

# Syllabus – Wintersemester 2020/21

|                 |                                    |   |
|-----------------|------------------------------------|---|
| 07.10.20        | 1. Einführung                      |   |
| 14.10.20        | 2. Kombinatorische Logik I         |   |
| 21.10.20        | 3. Kombinatorische Logik II        |   |
| 28.10.20        | 4. Sequenzielle Logik I            |   |
| 04.11.20        | 5. Sequenzielle Logik II           |   |
| 11.11.20        | 6. Arithmetik I                    |   |
| 18.11.20        | 7. Arithmetik II                   |   |
| 25.11.20        | 8. Befehlssatzarchitektur (ARM) I  |   |
| 02.12.20        | 9. Befehlssatzarchitektur (ARM) II | danach<br>→ <b>inday students</b> virtual edition |
| 09.12.20        | 10. Prozessorarchitekturen         |   |
| 16.12.20        | 11. Ein-/Ausgabe                   |   |
| 13.01.21        | 12. Speicher                       |   |
| 20.01.21        | 13. Leistung                       |   |
| <b>27.01.21</b> | <b>Klausur (1. Termin)</b>         |   |