Bits and Bytes

Stellenwertsysteme

Steinzeit ||||| ||||| .... |||| |||

Römer MM XX III sehr ungeeignet zum Rechnen

Dezimalsystem 2023 = $2 \cdot 1000 + 0 \cdot 100 + 2 \cdot 10 + 3 \cdot 1$

= $2 \cdot 10^3 + 0 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$

Zehnersystem

Stellenwertsystem zur Basis 10: zehn Ziffer 0, 1, 2, 3, 4, ..., 9

↳ Wert einer Ziffer hängt von ihrer Stelle ab

- z.B. die Ziffer 2 in 2023
- Nullen sind wichtig: 2023 ≠ 223
- Warum 10er-Potenzen? 10 Finger

Fünfersystem: das Stellenwertsystem zur Basis 5,

fünf Ziffern: 0, 1, 2, 3, 4

3	2	0	1

= $3 \cdot 125 + 2 \cdot 25 + 0 \cdot 5 + 1 \cdot 1$

= 426

Anzahl der 1er

Anzahl der 5er

Anzahl der 25er

Anzahl der 125er

$1 = 5^0$ ·5

$5 = 5^1$ ·5

$25 = 5^2$ ·5

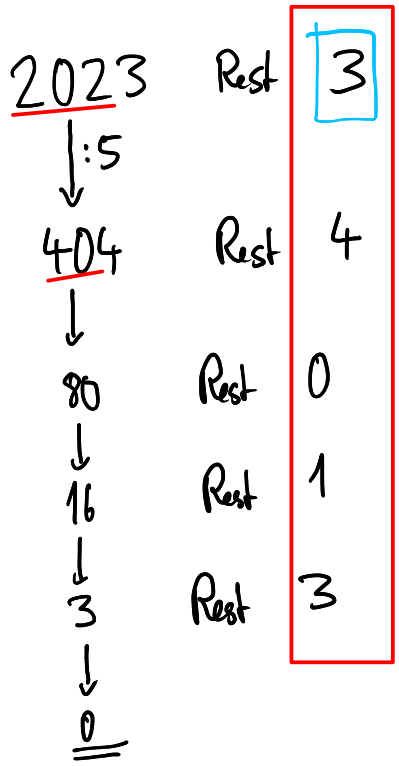
$125 = 5^3$

$(3201)_5 = (426)_{10}$

Aufgabe: $(2023)_{10} = (31043)_5$

systematischer Algorithmus / Verfahren

„dividiere solange durch 5 mit Rest, bis Null herauskommt"



$(3104)_5 = (2023)_{10}$

Idee dahinter: Die letzte Ziffer einer Zahl im Dezimalsystem ist der Rest der Division dieser Zahl durch 10.

2023**3**

↑ Rest der Division von 2023 durch 10

Aufgabe: $(1296)_{10} = (\quad)_5$

per Algorithmus + Probe

1296	1
259	4
51	1
10	0
2	2
0	0
0	0
0	

0020141

$1.1 + 4 \cdot 5 + 1.25 + 0.125 + 2 \cdot 625$

$= 1296$ ✓

Primarschule

Zähle im 5er-System von 0 bis 32

Kleines Einspluseins

Kleines Einmaleins

schriftliche Addition und Multiplikation
Dasselbe dann auch im Zweiersystem, alles an Tafel von SuS angeschrieben.

Logische Schaltungen

Bit = binary digit = Binärziffer, d.h. 0 oder 1

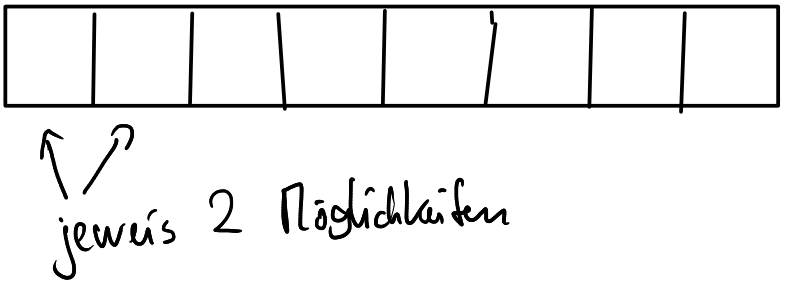
Byte = Folge von 8-Bit bzw. die Möglichkeit, eine solche Folge zu speichern

= 8-stellige Binärzahl, z.B. 00101101

Wie viele verschiedene Zahlen kann man in einem Byte speichern?

z.B. 0000 0000, 00000001, 1111 1111

Man kann $2^8 = 256$ Zahlen speichern:



Ziel: Entwurf einer logischen Schaltung, die zwei Byte addieren kann.

Boolesche / logische Verknüpfungen

True = wahr = AN = ON = 1

False = falsch = AUS = OFF = 0

Logisches Und (Konjunktion)

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

Logisches Gatter

Mathematische Notation

$x \wedge y$ statt x AND y

↑ stilisiertes A für AND.

Logisches Oder (Disjunktion)

x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

$x \vee y$

↑ Lateinisch vel = oder

Logisches Nicht (Negation)

x	NOT x
0	1
1	0

$\bar{x}$

Aufgabe: Fülle aus!

a	b	$\overline{a \wedge b}$	$\overline{a \vee b}$	$\bar{a} \wedge \bar{b}$	$\bar{a} \vee \bar{b}$	$a \wedge b$	$a \vee b$	$\bar{a}$	$\bar{b}$
0	0	1	1	1	1	0	0	1	1
0	1	1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1	0	1
1	1	0	0	0	0	1	1	0	0

gleich

gleich

$$\overline{a \wedge b} = \bar{a} \vee \bar{b}$$

$$\overline{a \vee b} = \bar{a} \wedge \bar{b}$$

} de Morgansche Gesetze

Es gibt einige ähnliche Logik-Gesetze, die man genauso beweist, vgl. Wikipedia

Wegen der Assoziativgesetze darf

man $a \vee b \vee c$ statt $(a \vee b) \vee c = a \vee (b \vee c)$

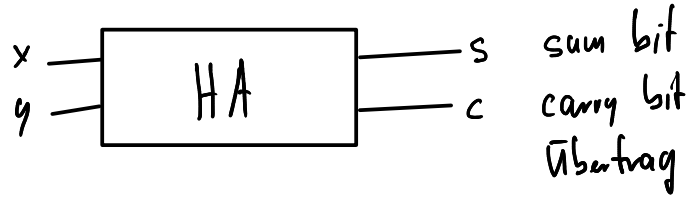
und $a \wedge b \wedge c$ statt $(a \wedge b) \wedge c = a \wedge (b \wedge c)$

schreiben.

Ziel: 4-Bit-Addierer bauen

dazu benötige

(1) Halbaddierer (HA) addiert 2 Bit (dh. zwei einstellige Binärzahlen)



x	y	c	s	x v y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

$$c = x \wedge y$$

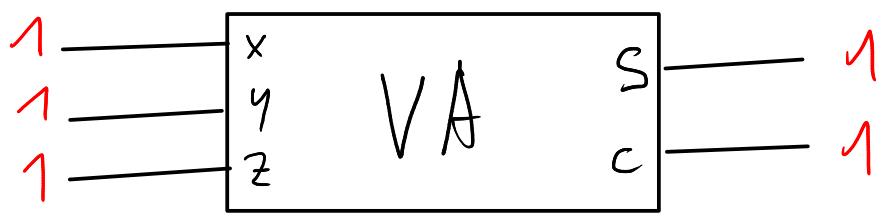
$$s = x \text{ XOR } y$$

exklusives Oder
Entweder-Oder

$$= (\bar{x} \wedge y) \vee (x \wedge \bar{y})$$

disjunktive Normalform.

(2) Volladdierer (VA) addiert 3 Bit.



1+1+1 = 11