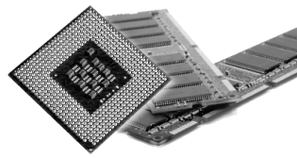
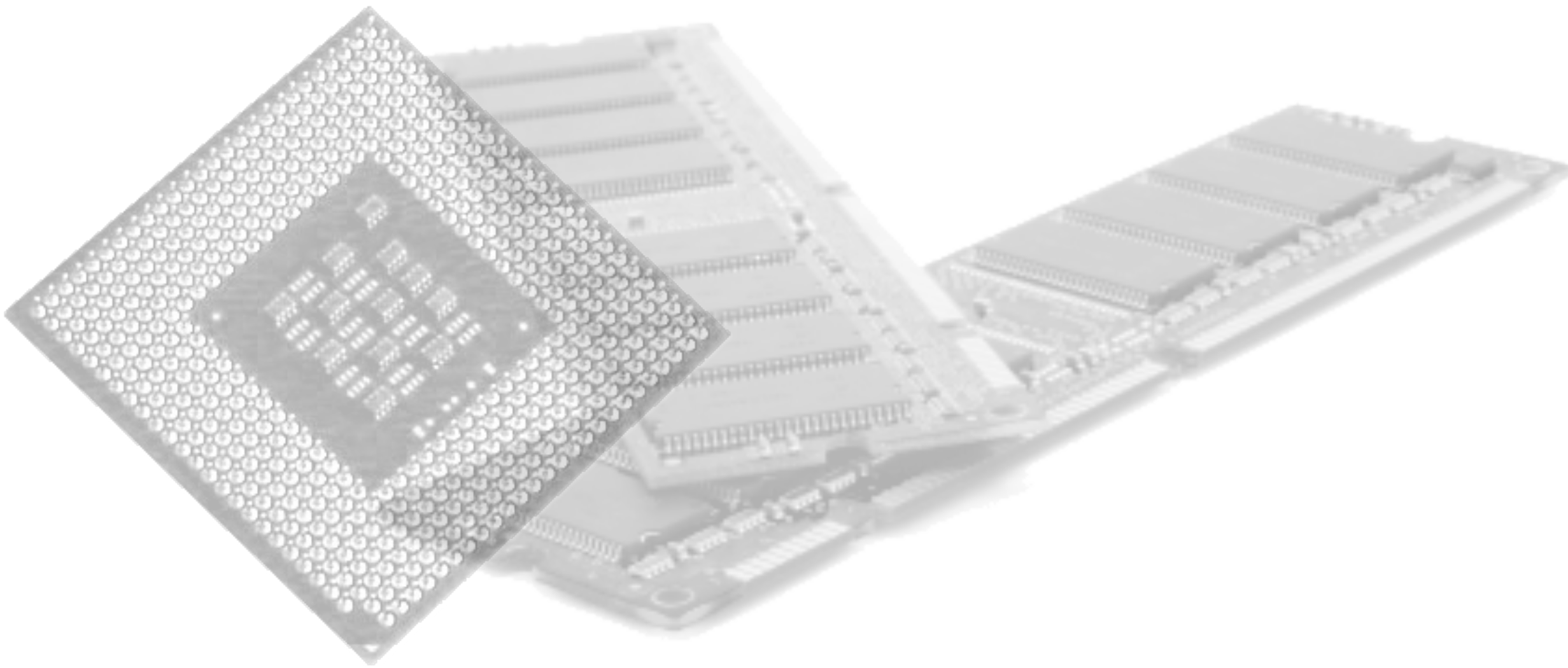
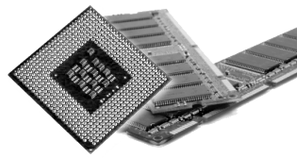


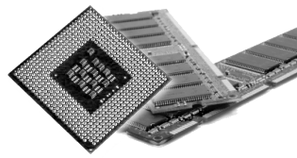
Logische Operationen



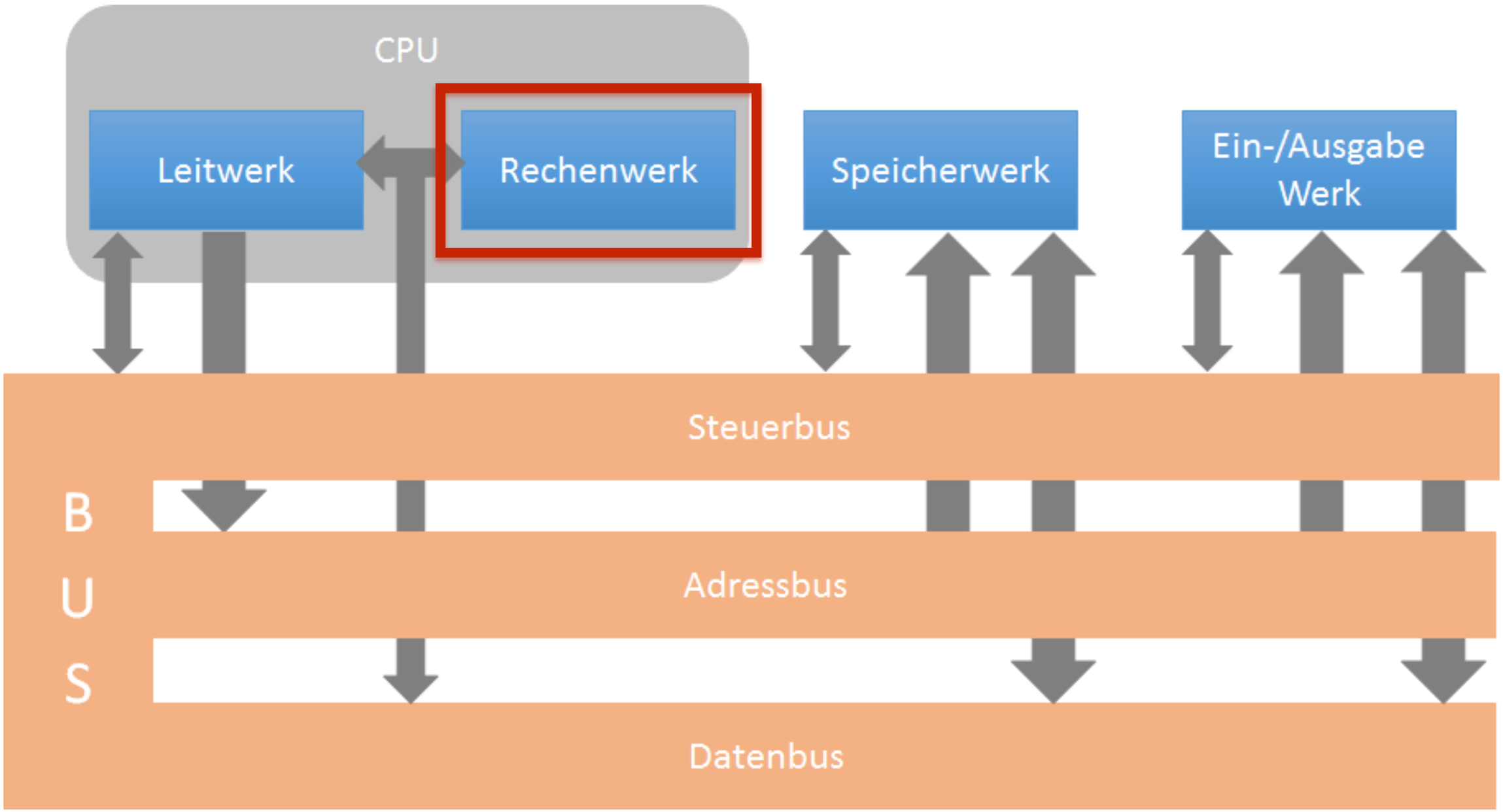


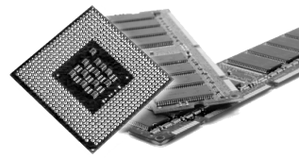
Logische Operationen

- Aus einem oder mehreren Signalen (Eingabe) werden durch logische Verknüpfung ein oder mehrere andere Signale (Ausgabe)
- In der Regel:
  - eine Eingabe - eine Ausgabe
  - zwei Eingaben - eine Ausgabe
- Funktionsprinzip der ALUs



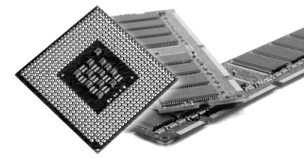
Der von-Neumann Rechner





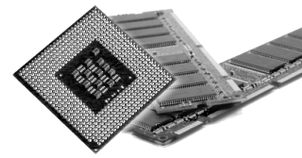
Arithmetisch-logische Einheit

- elektronisches Rechenwerk
- berechnet arithmetische und logische Funktionen
  - meistens mindestens:
    - Addition (ADD)
    - Negation (NOT)
    - Konjunktion (AND)
  - In der Regel zusätzlich: SUB, MUL, CMP, OR, SHR, …



Gattertypen

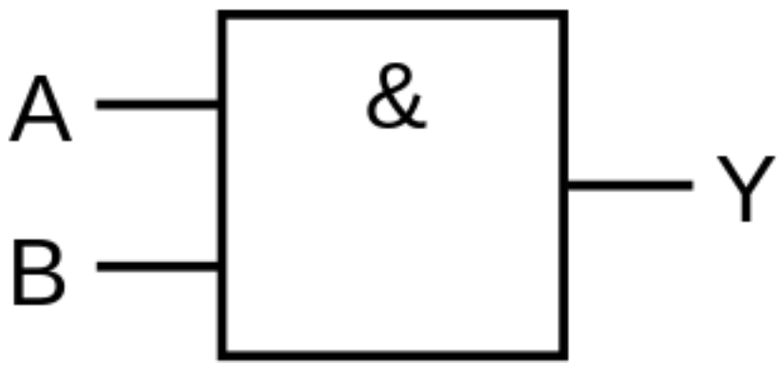
- AND
- OR
- NOT
- XOR
- NOR
- NAND
- XNOR



AND

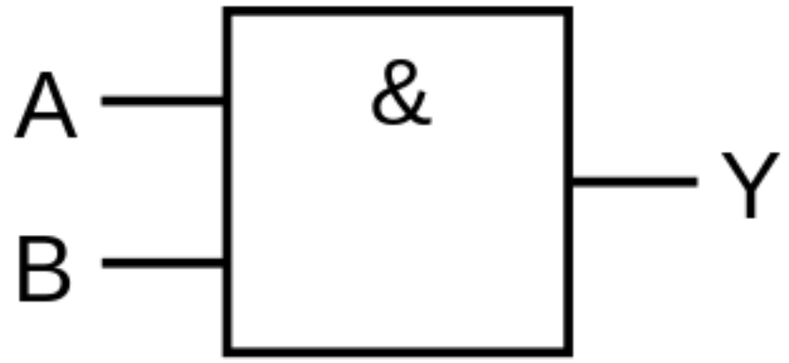
- $Y = a \wedge b$
- Y = a * b
- Y = ab
- Wenn alle Eingänge 1 sind, dann ist auch Y 1

a	b	Y
0	*0*	***0***
0	*1*	***0***
1	*0*	***0***
1	*1*	***1***

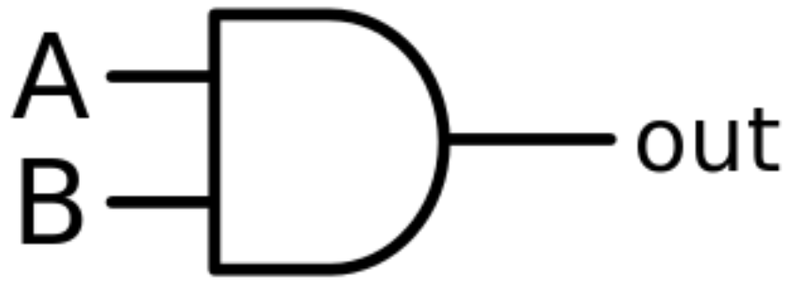


Schaltsymbole

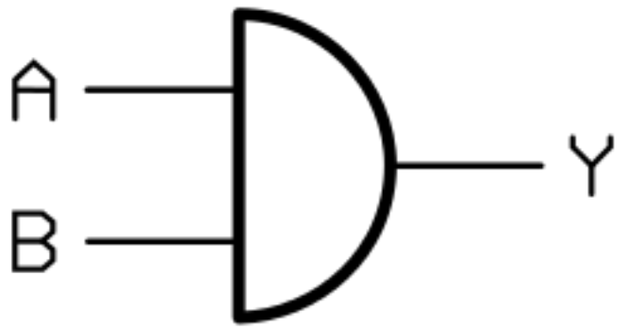
- Bsp.: AND

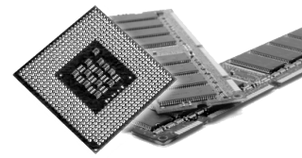


- US ANSI 91-1984:



- DIN 40700 (vor 1976):



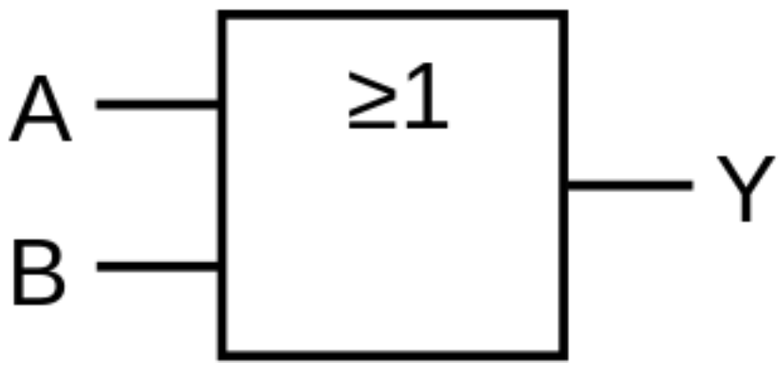


OR

- Y = a ∨ b
- Y = a + b

a	b	Y
0	*0*	***0***
0	*1*	***1***
1	*0*	***1***
1	*1*	***1***

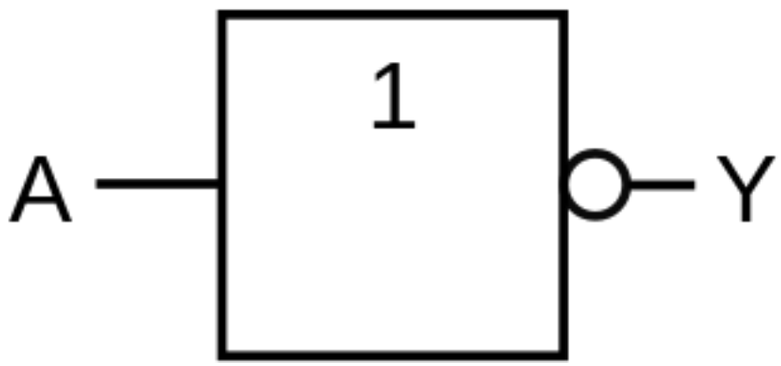
- Wenn mindestens ein Eingang 1 ist, dann ist Y auch 1

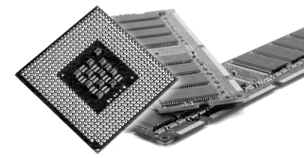


NOT

- Y = ¬a
- Y = !a
- Y = $\bar{a}$
- Der Eingang wird negiert (umgekehrt)

a	Y
0	***1***
1	***0***



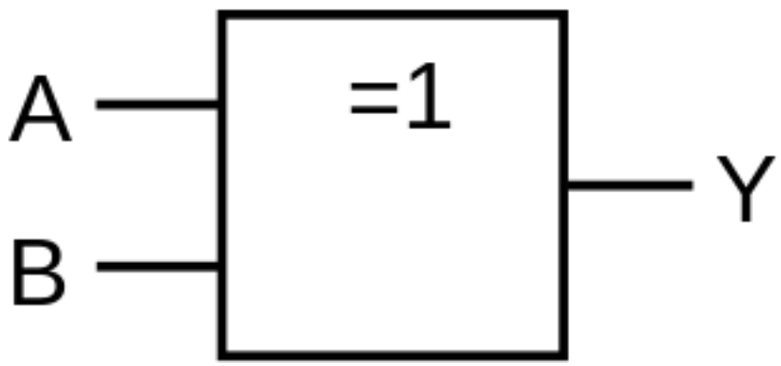


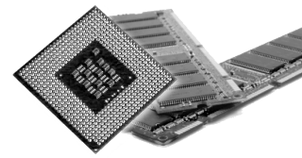
XOR

- $Y = a \veebar b$
- $Y = a \oplus b$

a	b	Y
0	*0*	***0***
0	*1*	***1***
1	*0*	***1***
1	*1*	***0***

- Wenn genau ein Eingang 1 ist, dann ist Y auch 1



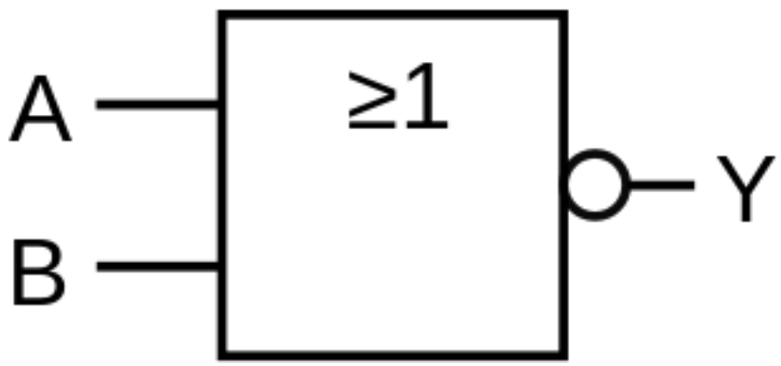


NOR

- $Y = \overline{a \vee b}$
- $Y = \overline{a + b}$

a	b	Y
0	*0*	***1***
0	*1*	***0***
1	*0*	***0***
1	*1*	***0***

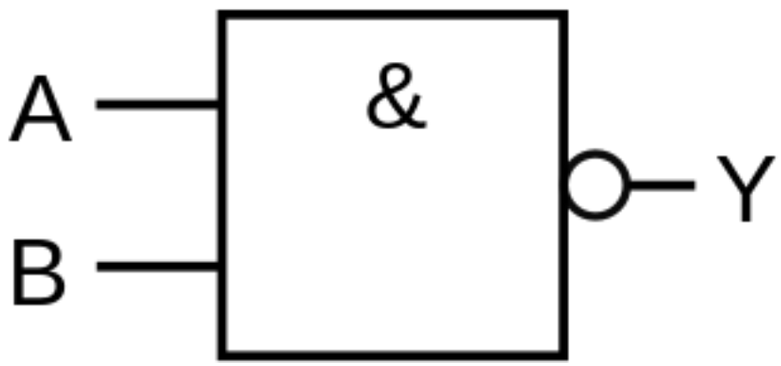
- Nur wenn alle Eingänge 0 sind, ist Y 1



NAND

- $Y = a \barwedge b$
- $Y = \overline{a \wedge b}$
- Wenn nicht alle Eingänge 1 sind, ist Y auch 1

a	b	Y
0	*0*	***1***
0	*1*	***1***
1	*0*	***1***
1	*1*	***0***

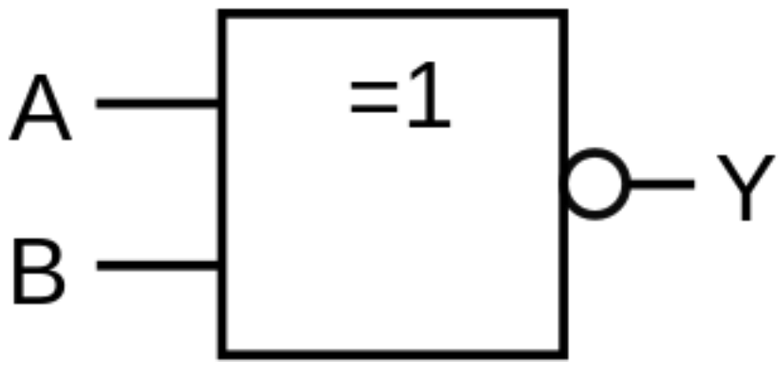


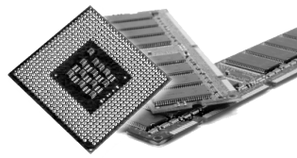
XNOR

- $Y = \overline{a \veebar b}$
- $Y = \overline{a \oplus b}$

a	b	Y
0	*0*	***1***
0	*1*	***0***
1	*0*	***0***
1	*1*	***1***

- Wenn entweder alle Eingänge 0 oder 1 sind, ist Y auch 1



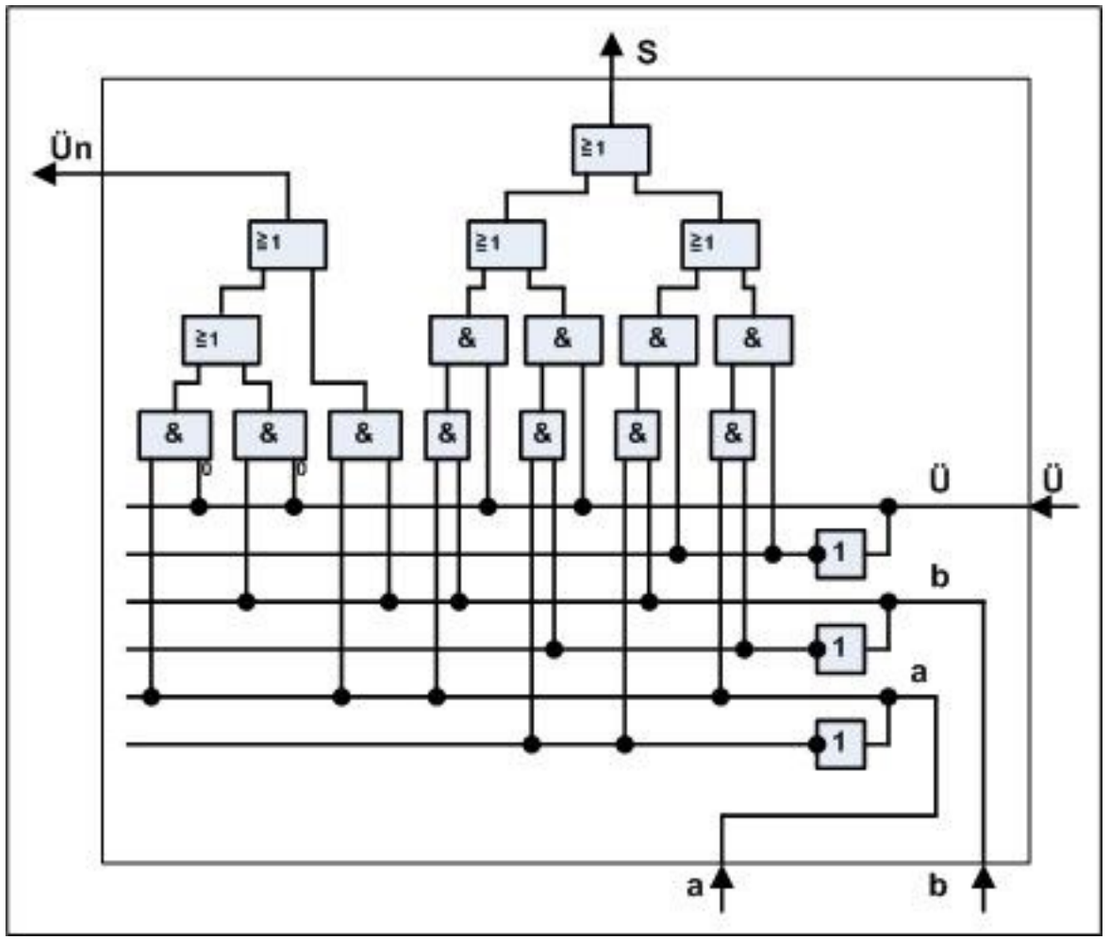


Logische Operationen

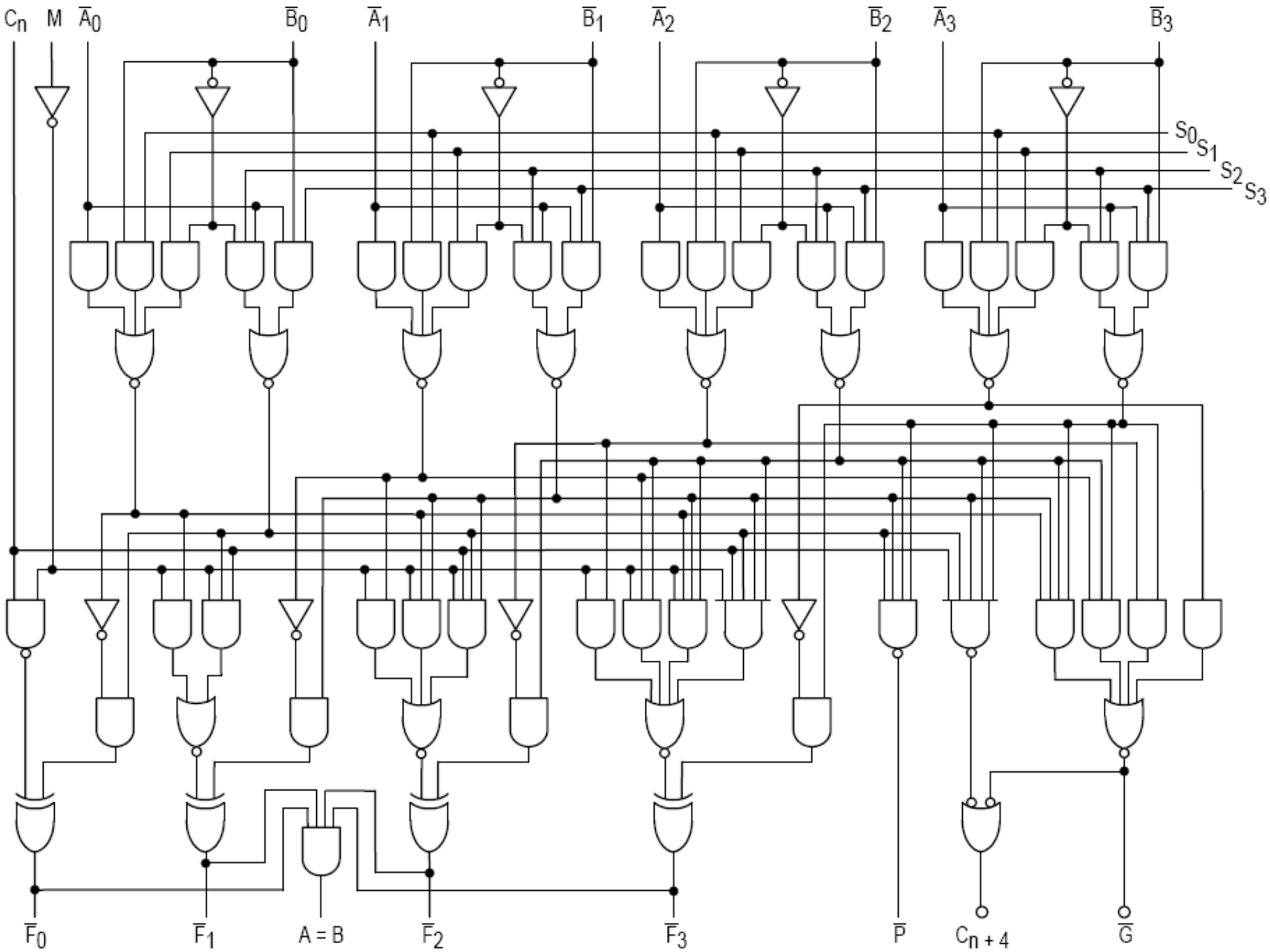
- Alle binären Funktionen lassen sich auf AND, OR und NOT zurückführen
- OR und NOT kann man mit NAND darstellen
- AND und NOT kann man mit NOR darstellen
- Damit lässt sich jede logische Schaltung allein durch NAND oder NOR darstellen

Logische Operationen

- Addierer aus logischen Schaltungen:
- 1 Bit

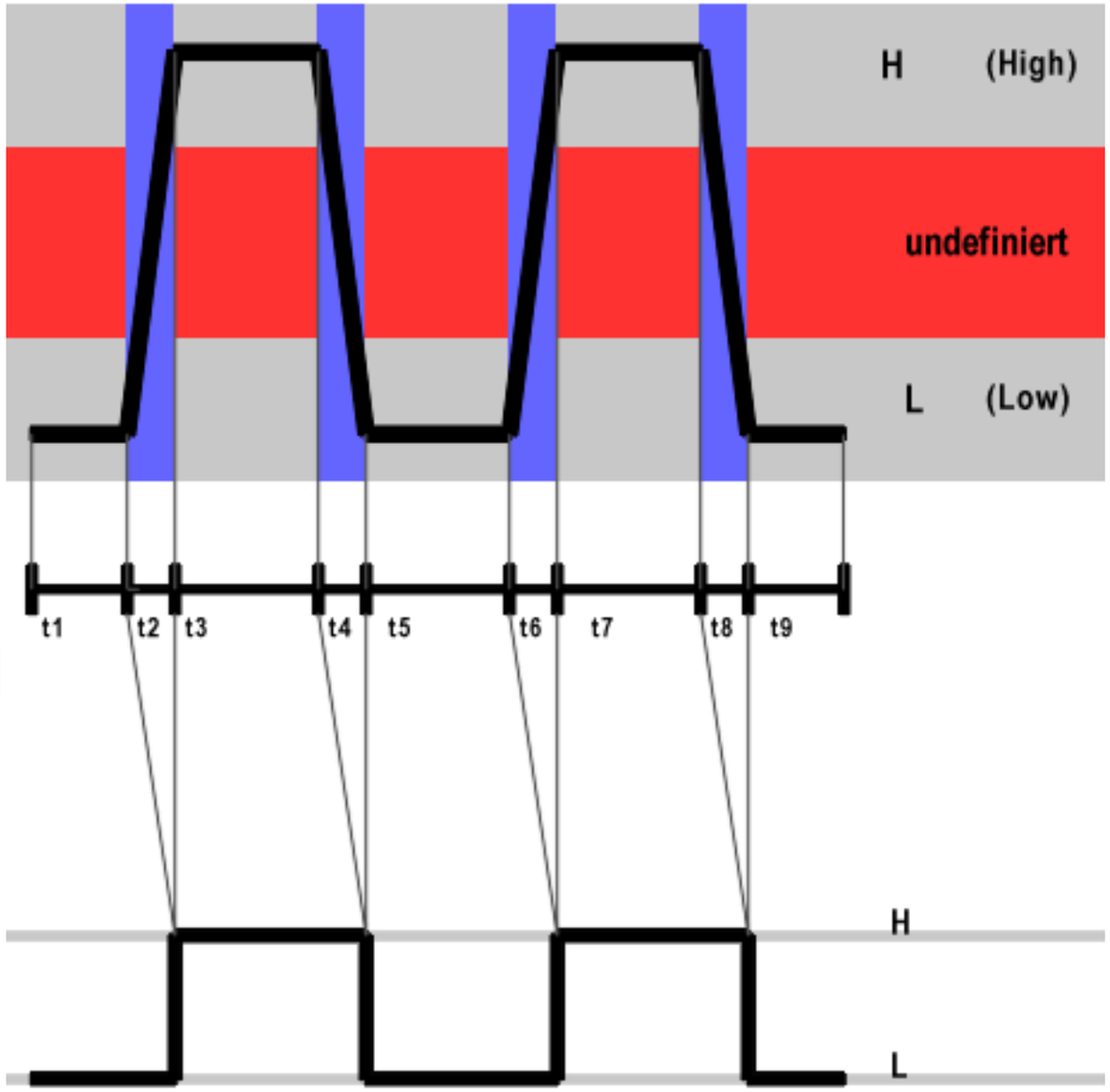


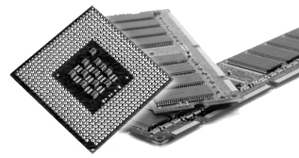
Einfache 4-Bit ALU



Logikpegel

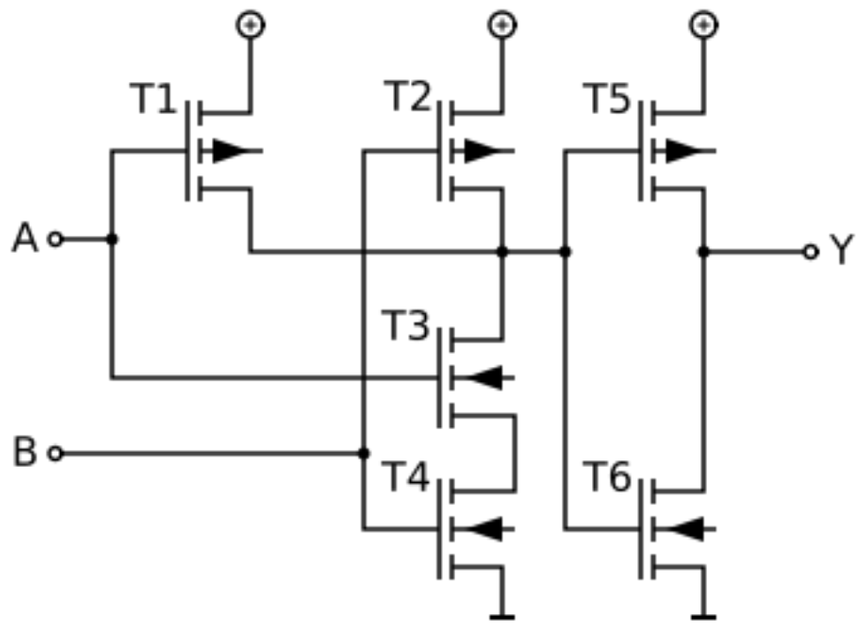
- Der Logikpegel ist die elektrische Spannung, welche den Logikwerten zugeordnet wird
- Binär Codiert:
  - High-Pegel (z.B. >4,44 V) = 1
  - Low-Pegel (z.B. < 0,5 V) = 0
  - bei positiver Logik





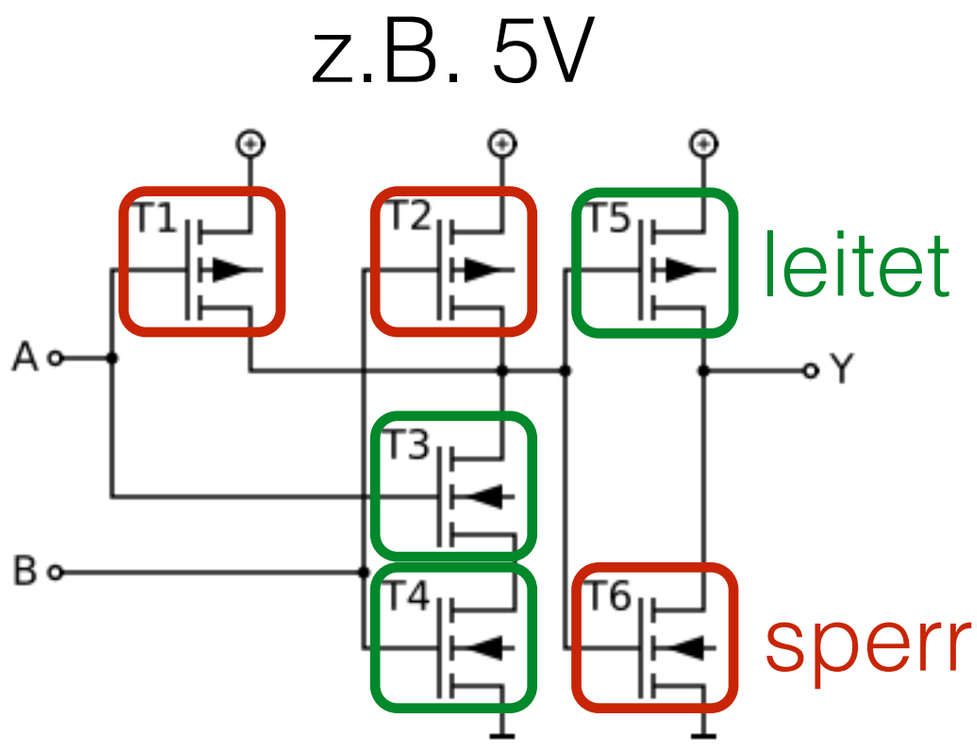
AND

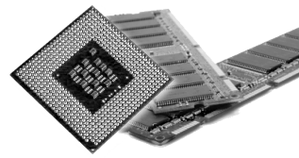
- in CMOS Technologie
- T = Transistor



AND

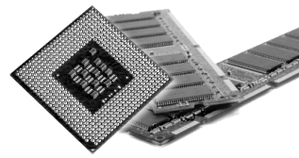
- in CMOS Technologie
- T = Transistor
- z.B.
  - A = 1 (High)
  - B = 1 (High)





Assembler

- AND <Ziel>, <Quelle>
- OR <Ziel>, <Quelle>
- XOR <Ziel>, <Quelle>
- NOT <Ziel>

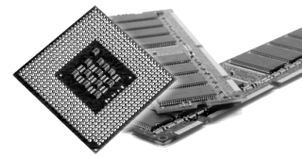


Assembler

- EAX: 1100 0111 0001 0001
- 05511h: 0101 0101 0001 0001
- AND: 0100 0101 0001 0001

```
;write your code here

mov eax, 0C711h
and eax, 05511h

PRINT_HEX 4, eax
```

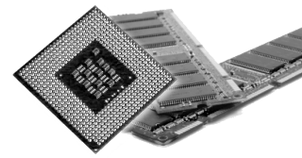
```
Output

4511
```



Assembler

- XOR eax, eax
- eax wird auf 0 gesetzt
- Bsp. 8 Bit:
- 0100 0111
- 0100 0111
- 0000 0000

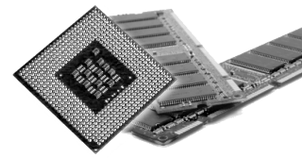
a	b	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**



Flags verändern

- Carry (erstes Flag löschen)
  - PUSHF
  - POP ax
  - AND ax, 0FFFEh
  - PUSH ax
  - POPF
- alle anderen Flags bleiben unverändert:
  - 0000 0010 1000 0011
  - 1111 1111 1111 1110
  - AND: 0000 0010 1000 0010

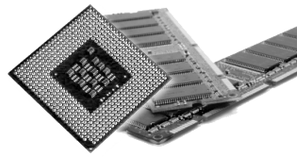
a	b	Y
0	*0*	***0***
0	*1*	***0***
1	*0*	***0***
1	*1*	***1***



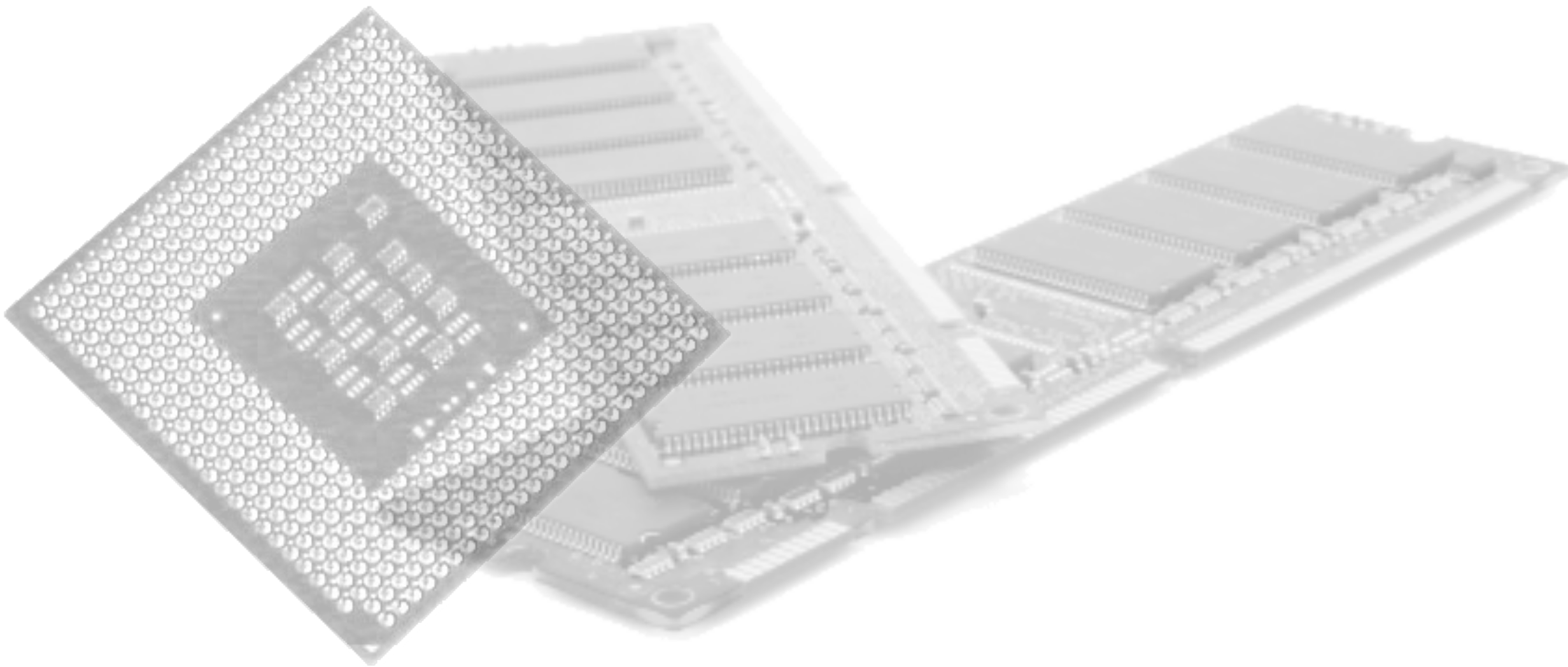
Flags verändern

- Carry (erstes Flag setzen)
  - PUSHF
  - POP ax
  - OR ax, 0001h
  - PUSH ax
  - POPF
- alle anderen Flags bleiben unverändert:
  - 0000 0010 1000 0010
  - 0000 0000 0000 0001
  - OR: 0000 0010 1000 0011

a	b	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**



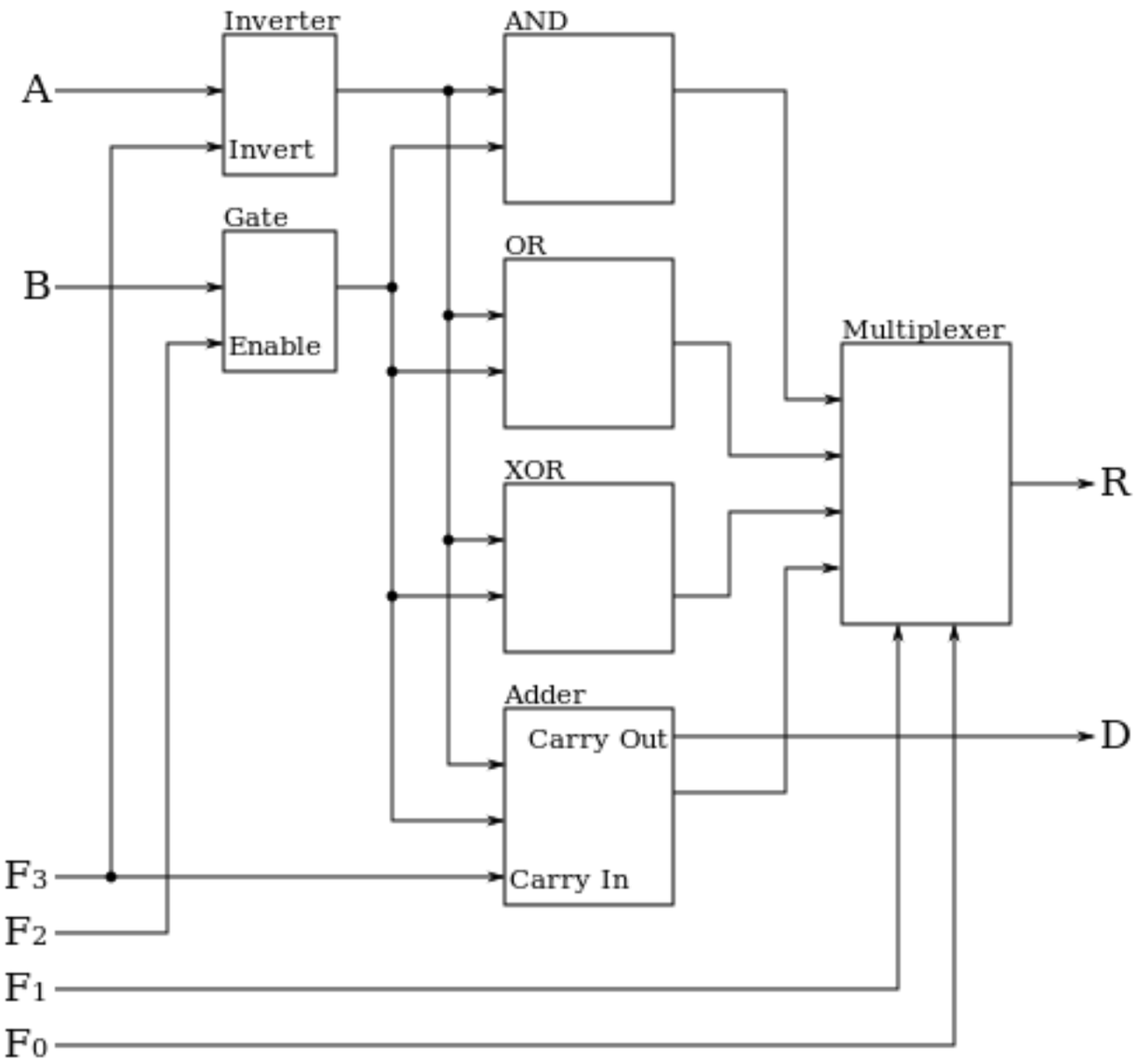
Aufbau ALU



Aufbau ALU

in CMOS Technologie

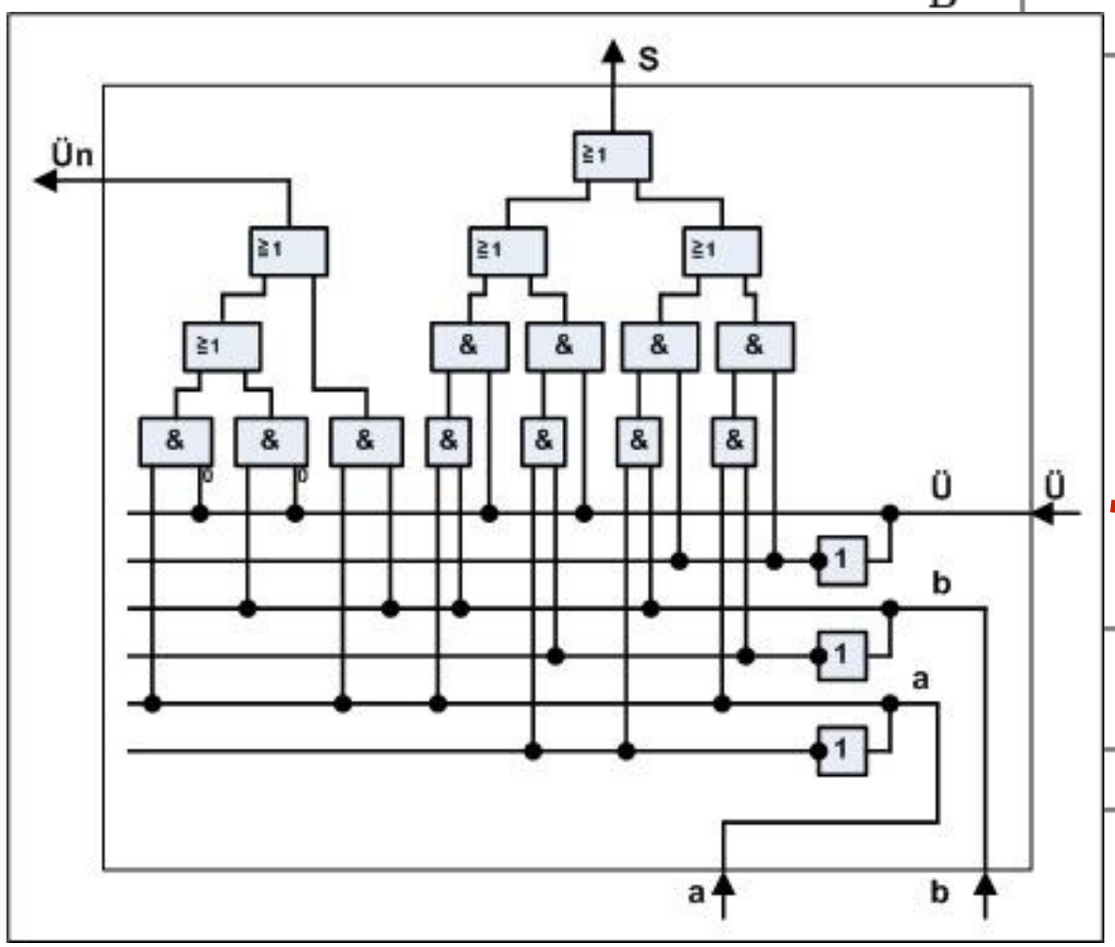
T = Transistor

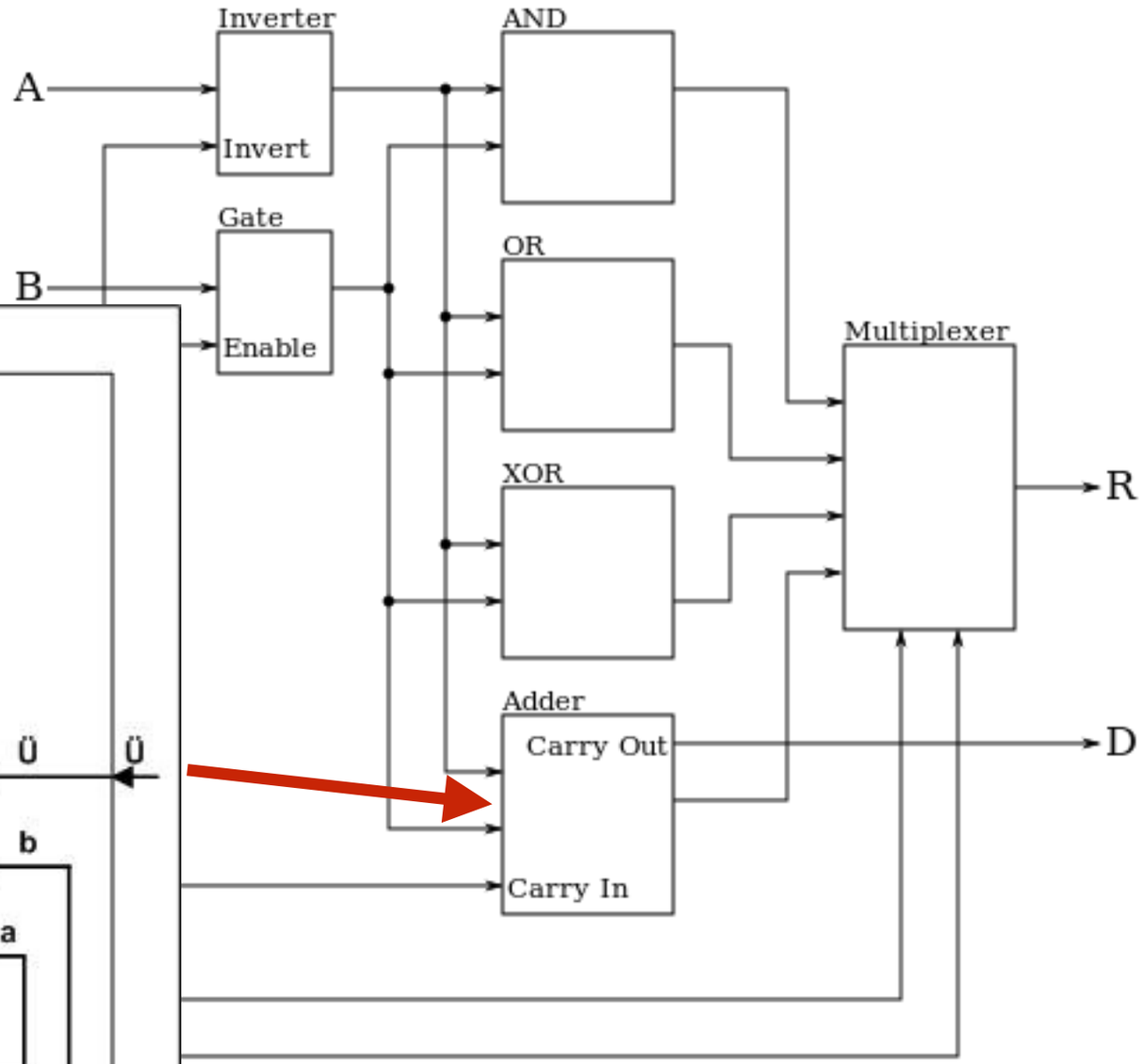


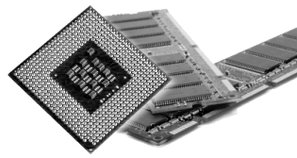
Aufbau ALU

in CMOS Technologie

T = Transistor



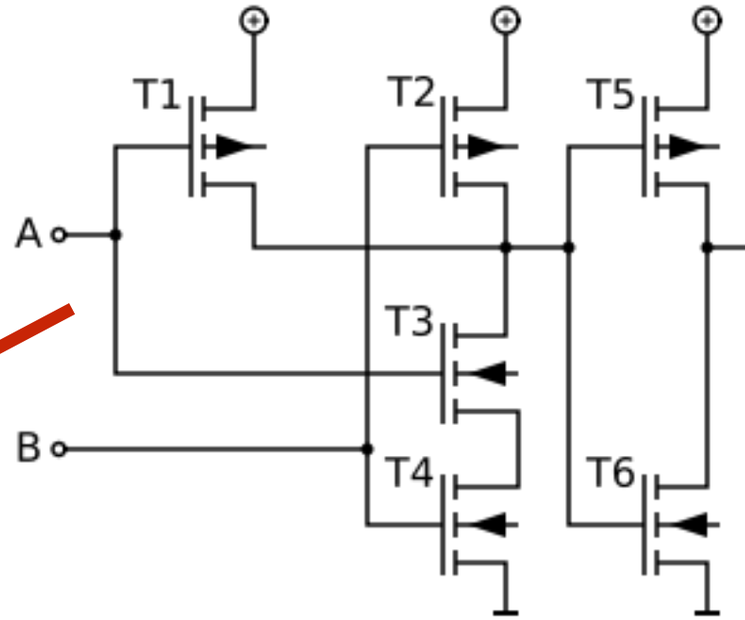


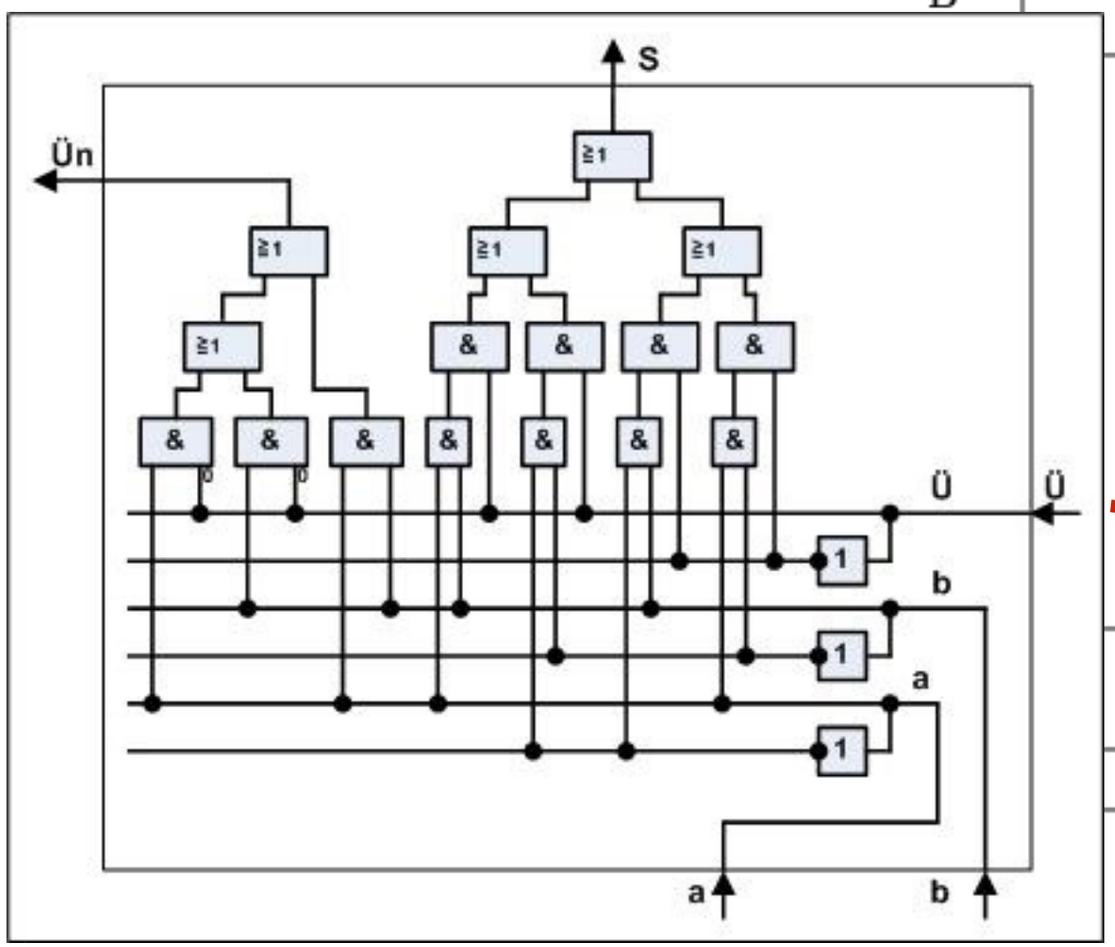


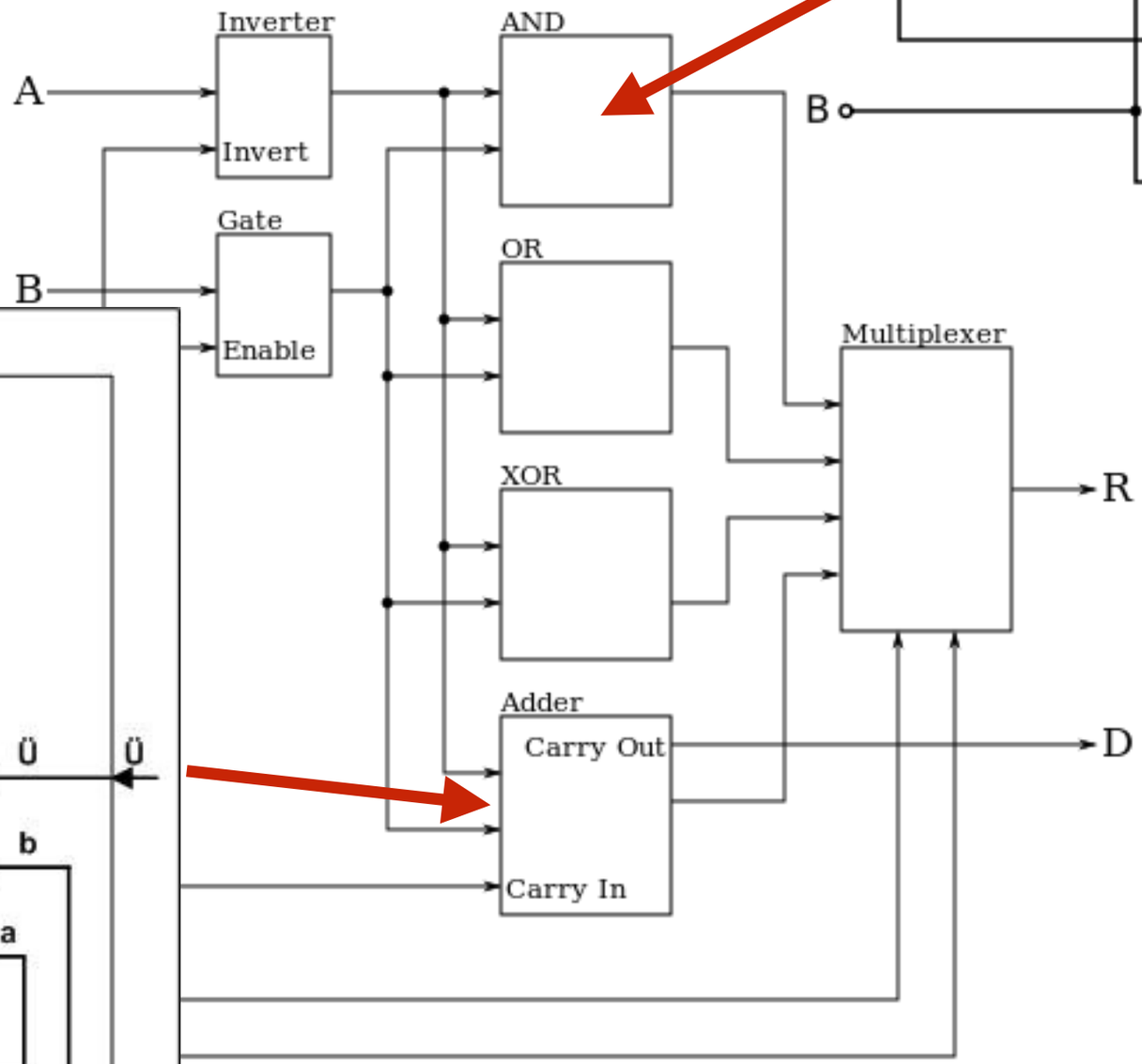
Aufbau ALU

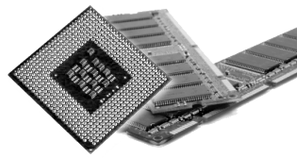
in CMOS Technologie

T = Transistor





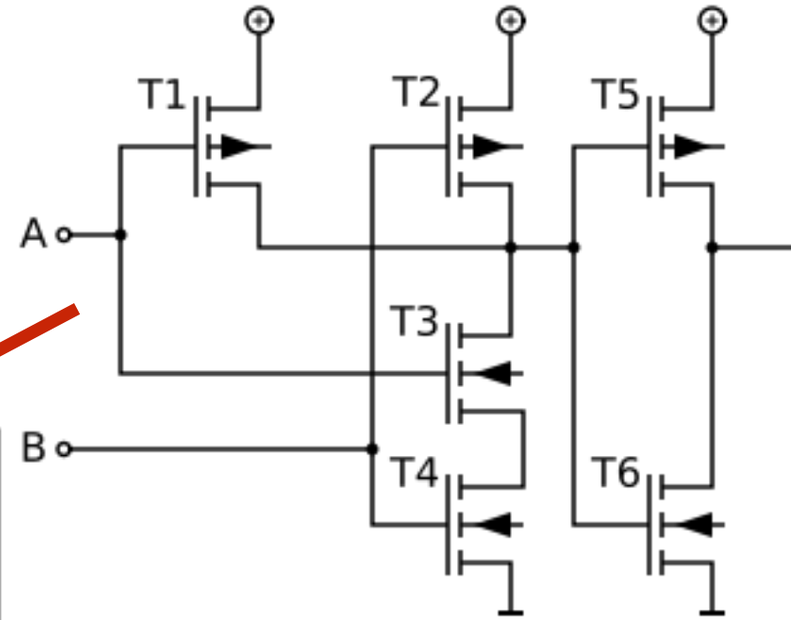


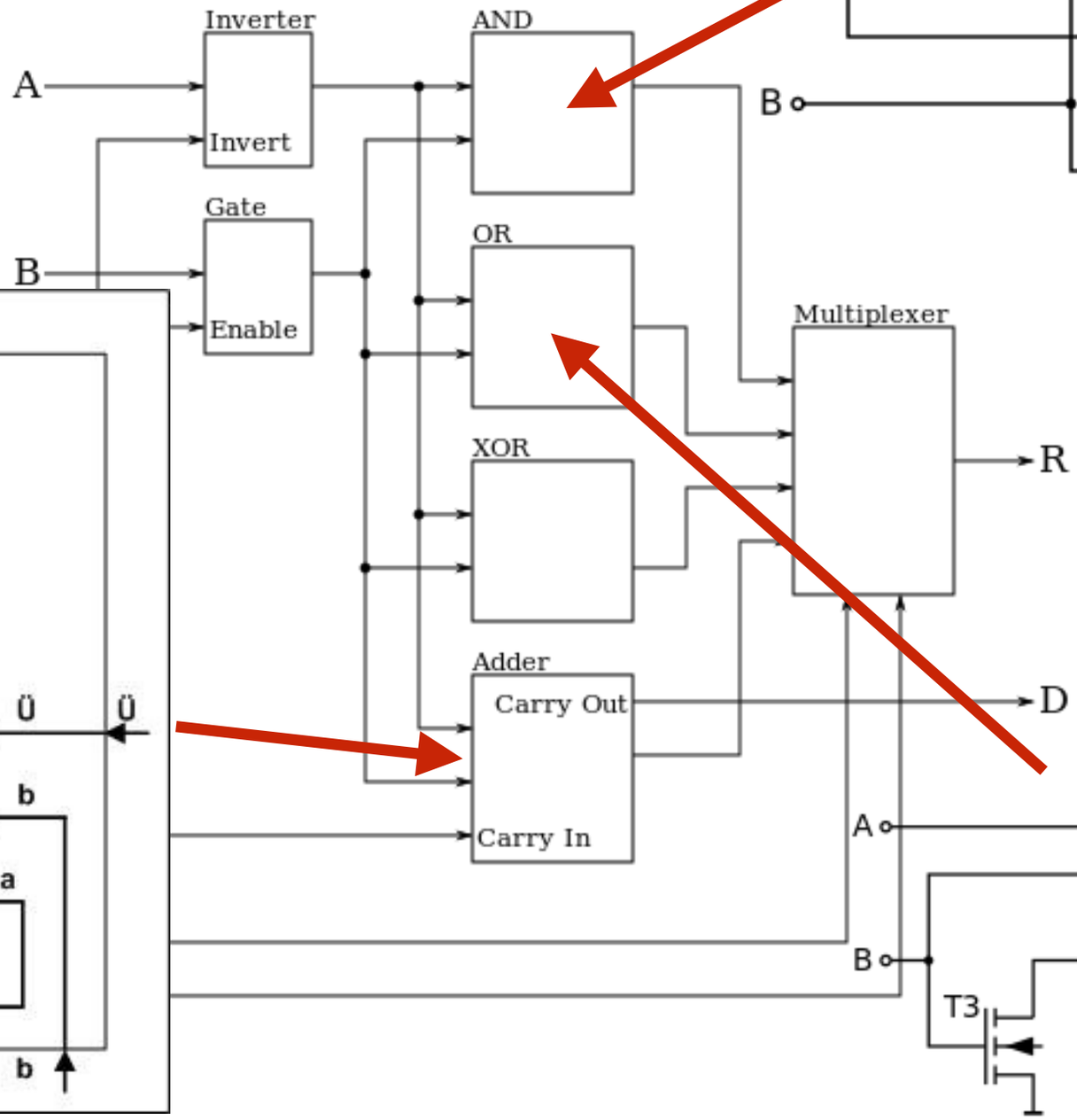


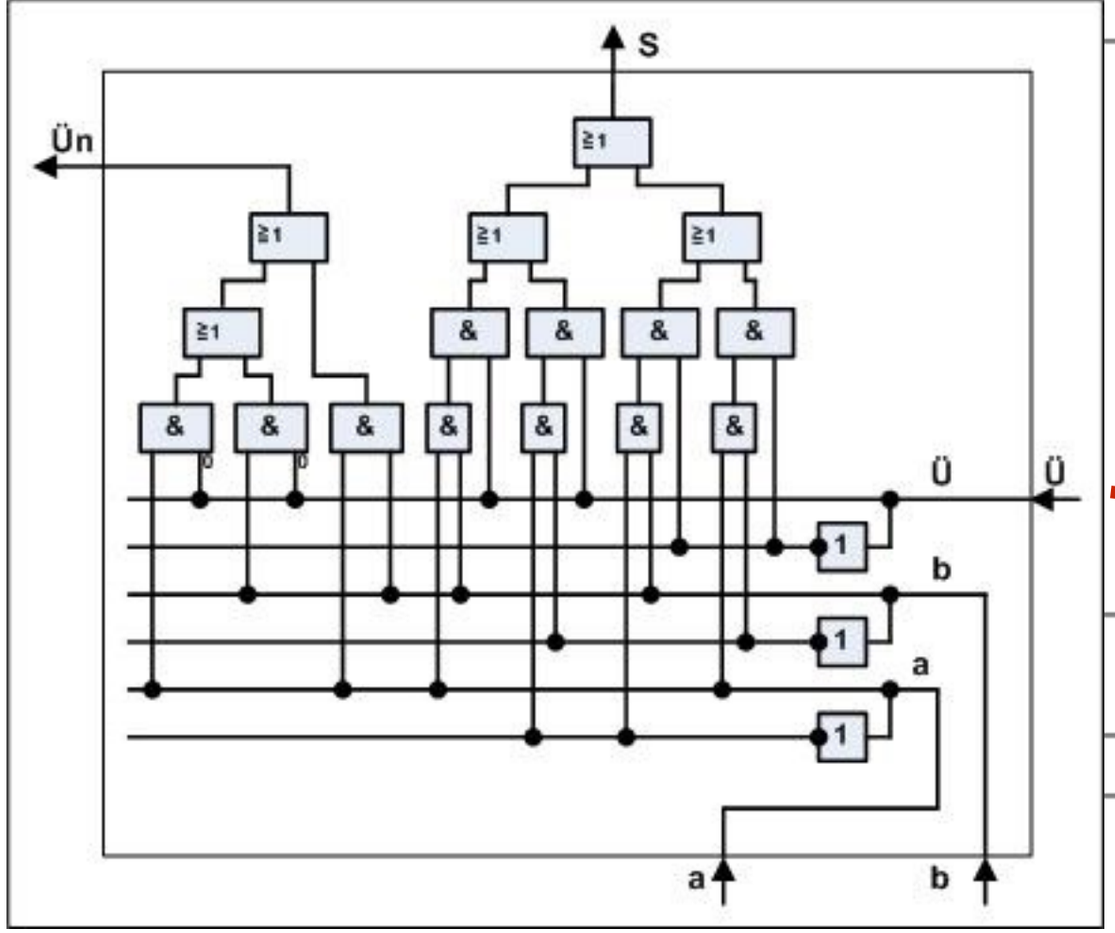
Aufbau ALU

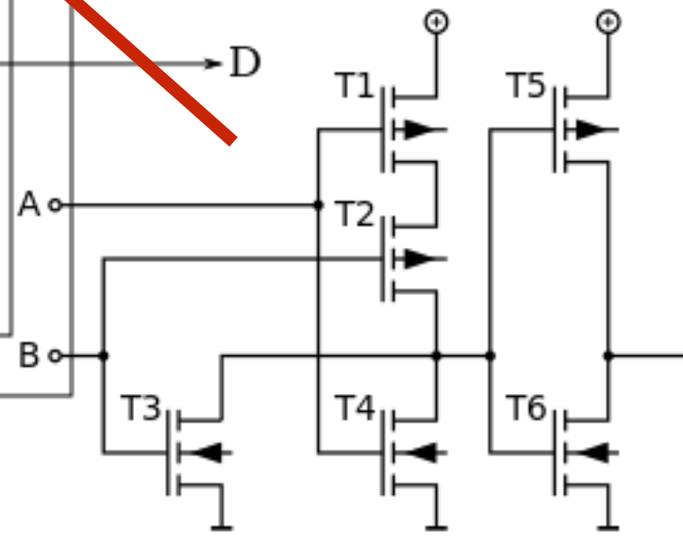
in CMOS Technologie

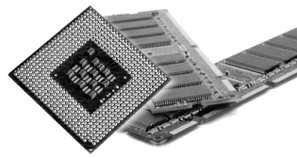
T = Transistor











Aufbau ALU

in CMOS Technologie

T = Transistor