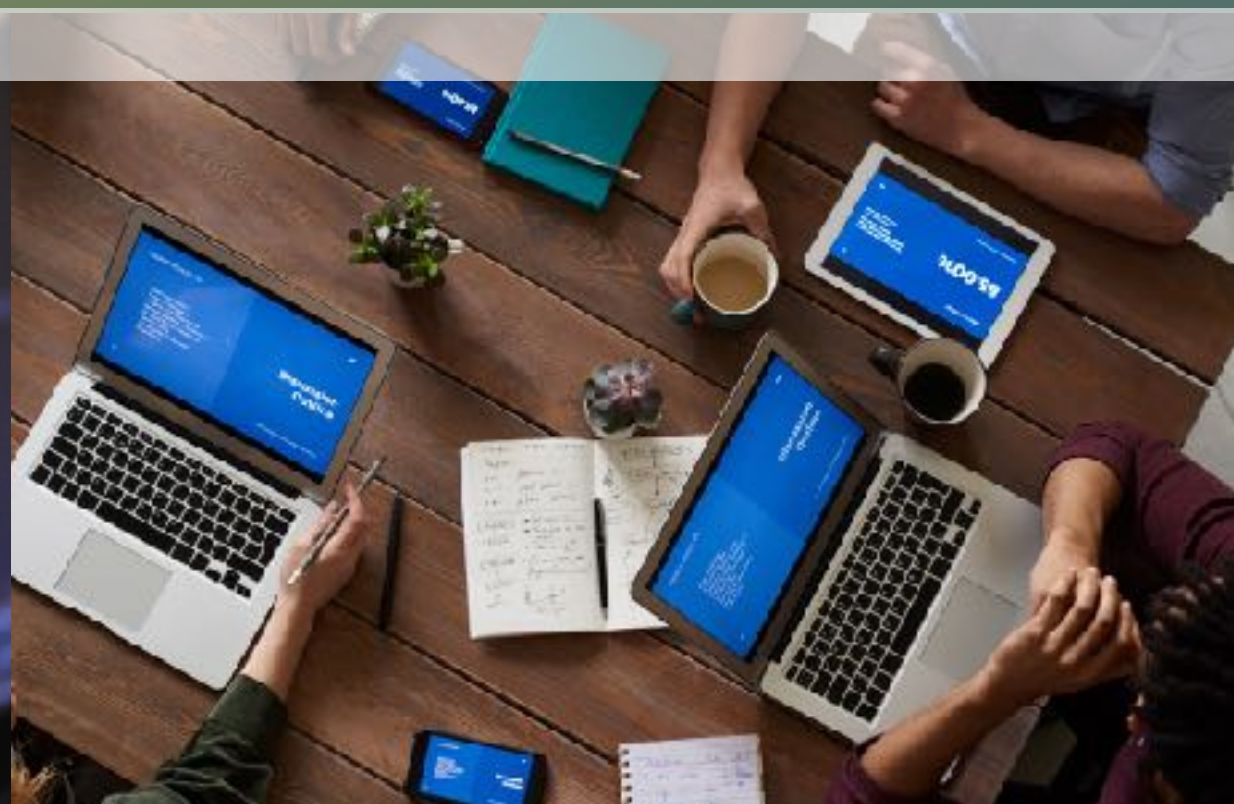
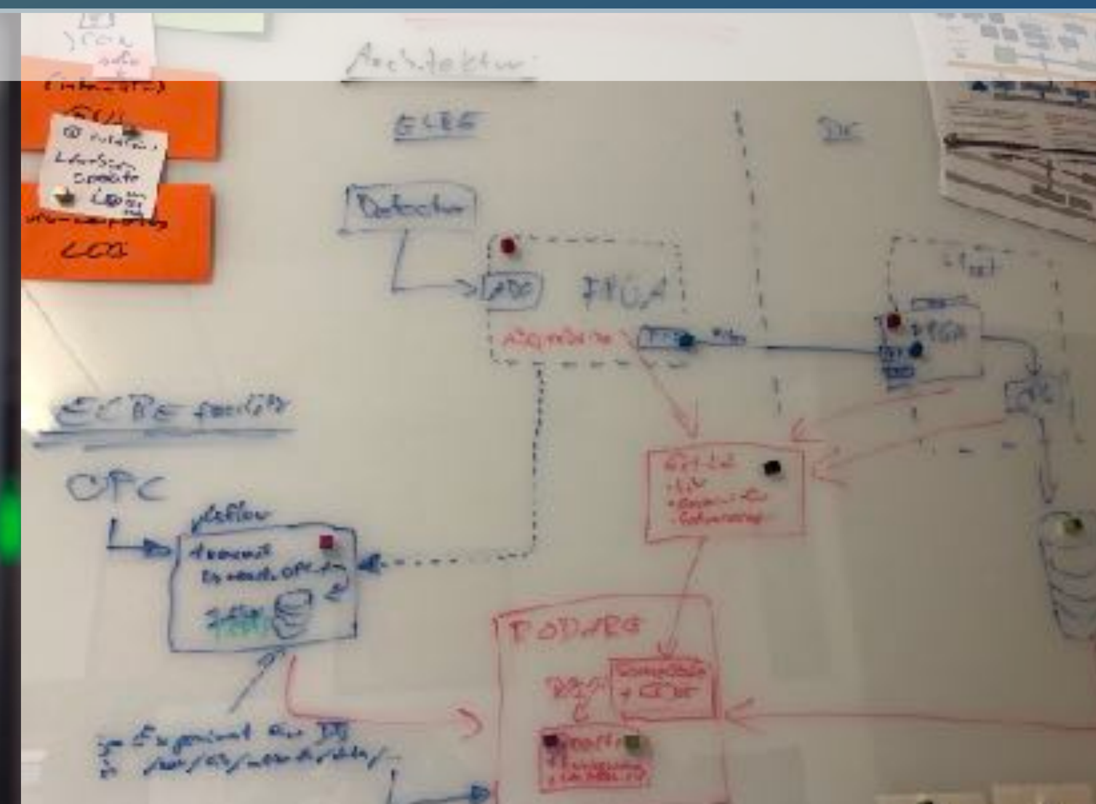
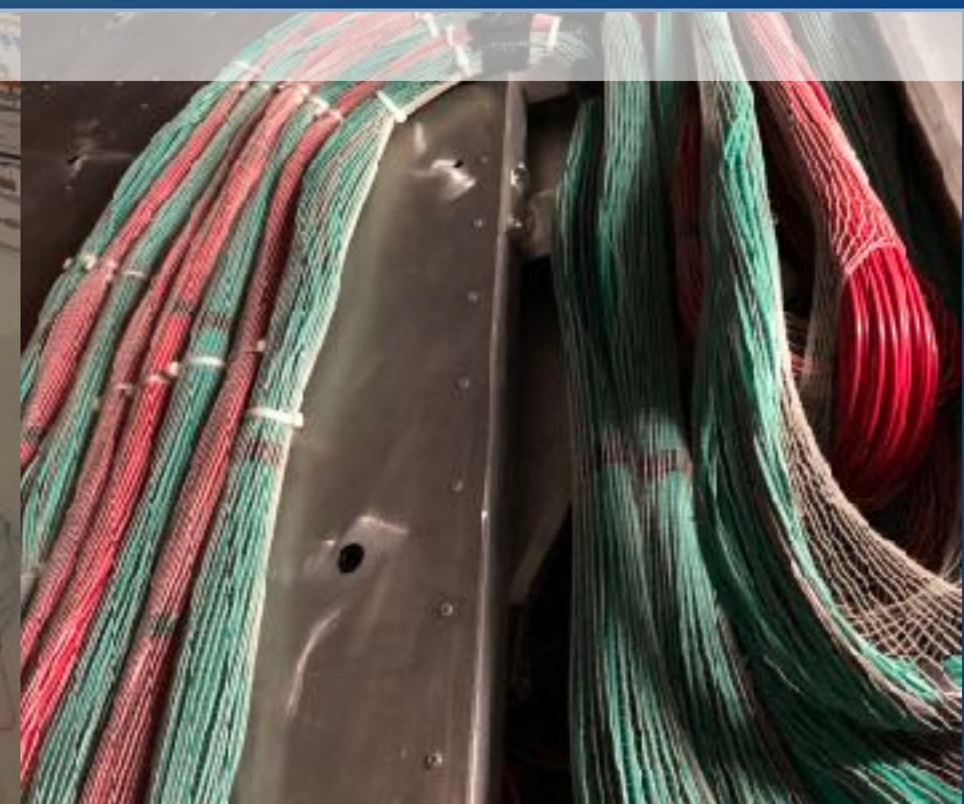
HZDR HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

Dr.-Ing. Oliver Knodel

Rechenwerk

Dresden // Mai, 2024



Rechenwerk — Komponenten

Verarbeitungseinheit – ALU (ALU – Arithmetic Logical Unit):

Realisiert arithmetische -, logische -, Vergleichs- und Verschiebeoperationen von Operanden 1 und Operend 2 (Akkumulator) zum Resultat (Akkumulator). Daten zur Operation und zum Ergebnis werden im Statusregister gepeichert.

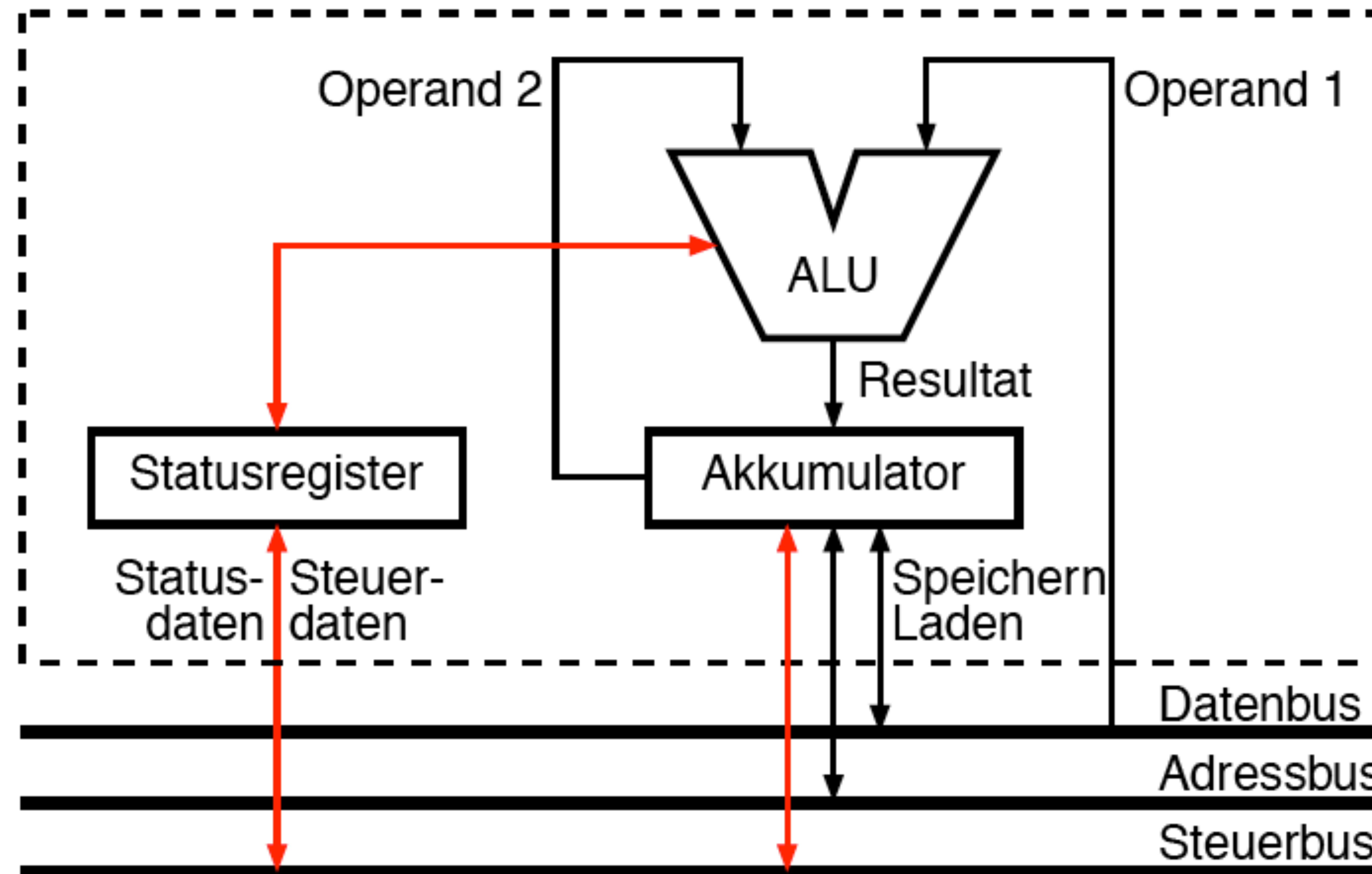
Akkumulator Register – AKKU (ACCU – Accumulator Register):

Universalregister zur Abspeicherung der Ergebnisdaten (Resultat). Gleichzeitig Hilfsregister für Operand 2 und für den Speichertransfer, zum Laden von Operand 2 bzw. zum Speichern des Resultates (Load/Store).

Statusregister – SR (SR – Status Register):

Dient der Zwischenspeicherung der Steuerdaten des Steuerwerkes und der Statusdaten des Rechenwerkes (Bedingungskode, Fehler, . . .). Realisierung Daten-bedingter Programmverzweigungen durch Abfrage der Statusdaten, bedingte Sprünge ($\Rightarrow$ Mikroprogrammsteuerung).

Rechenwerk — Arithmetisch-Logische Einheit (ALU)

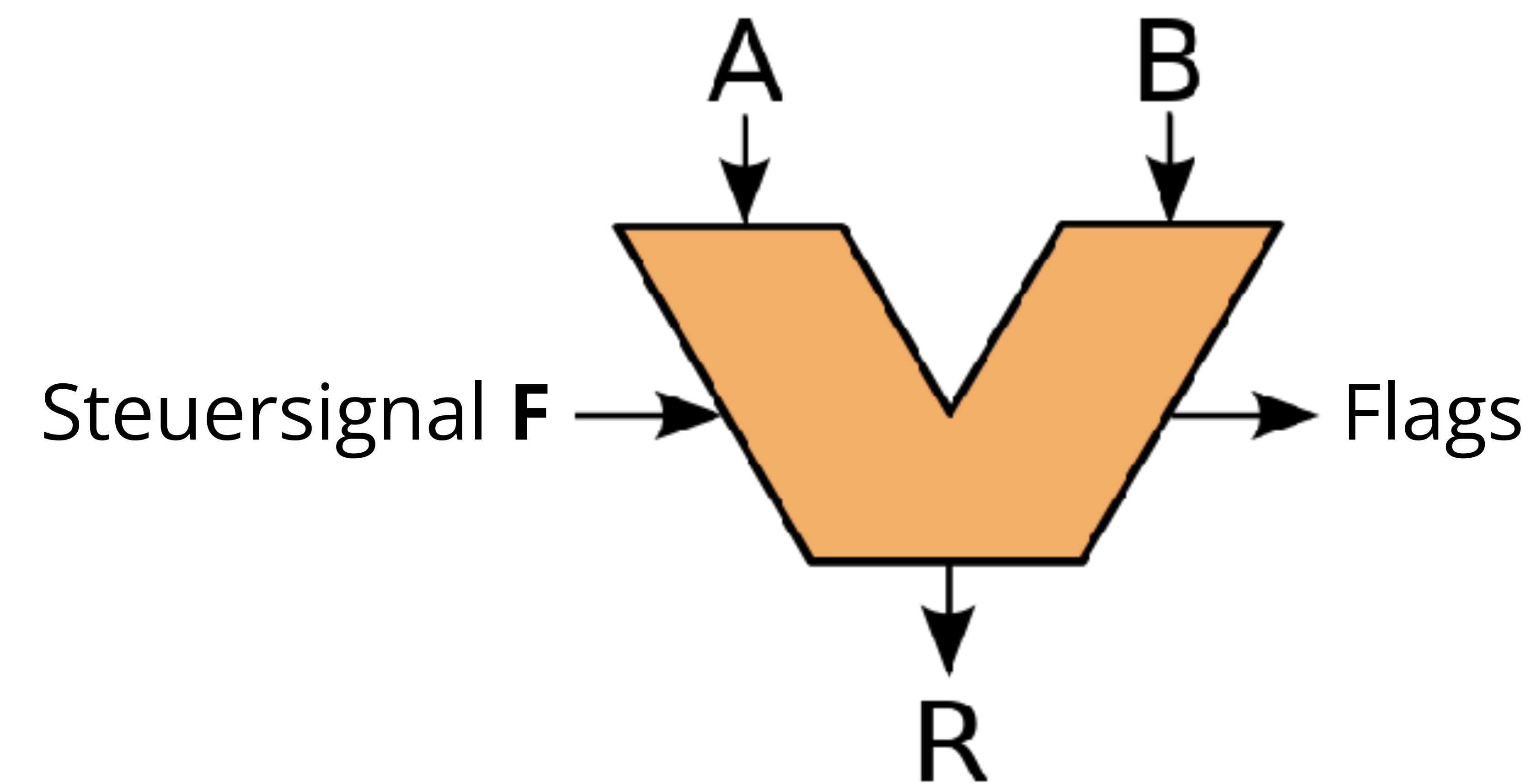


Funktionen der Arithmetisch Logischen Einheit

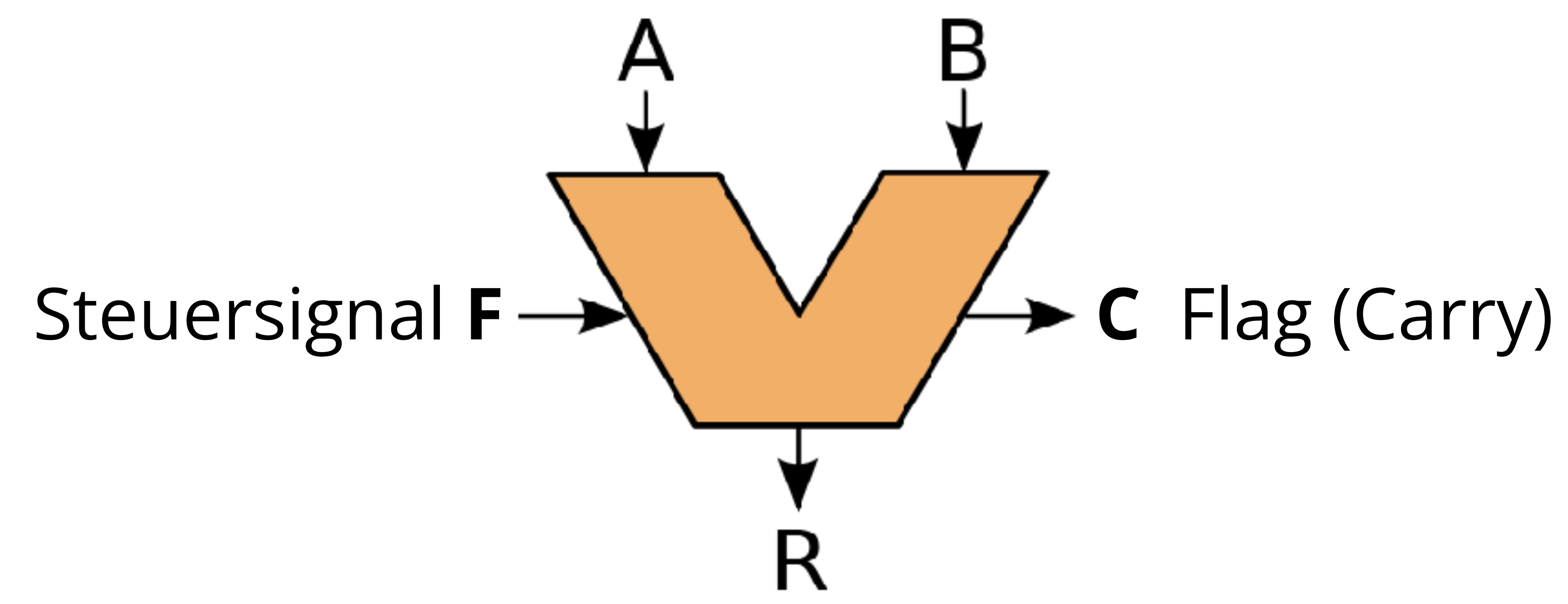
— Arithmetische Operationen:
- Addtition (ADD), Addition mit Übertrag (ADC) , Subtraktion (SUB), Subtraktion mit Übertrag (SBB)
- Multiplikation (MUL), Division (DIV)
- Inkrement (INC), Dekrement (DEC)
- Vergleiche (CMP)

— Logische Operationen:
- AND, OR, XOR, ...
- Schiebeoperationen
- Rotationen
- Registertransfers
- Bit-Manipulation

— Steuerbefehle:
- Interrupts, I/O

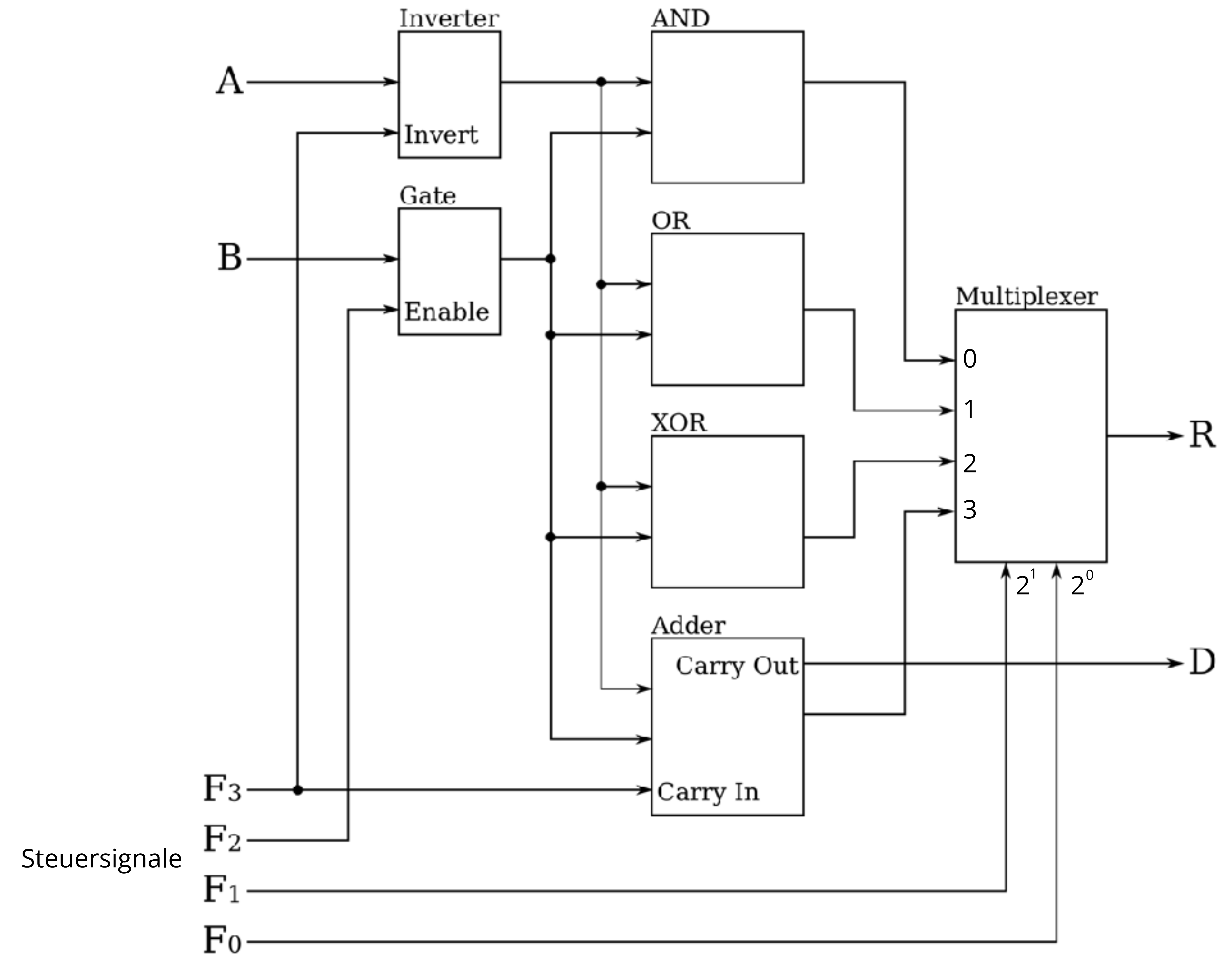


Aufbau eines einfachen Rechenwerkes



Befehlskodierung

F_3	F_2	F_1	F_0	Befehl
0	0	0	0	Null
0	0	0	1	A
1	0	0	1	NOT A
0	1	0	0	A AND B
0	1	0	1	A OR B
0	1	1	0	A XOR B
0	1	1	1	A + B
1	1	1	1	B – A



Bedeutung der Flags

Nullflag (Zero) Z:
— Z = 1, wenn Ergebnis = 0.

Vorzeichenflag (Sign) S, (N):
— S = 1, wenn Vorzeichenbit (MSB) = 1 (d.h. negativ).

Übertragsflag (Carry) C:
— C = 1, wenn Übertrag cn aus dem höchstwertigen Bit vorliegt.

Überlaufflag (Overflow) V, (OF, OV):
— V = 1, wenn $c_n \neq c_{n-1}$.
— cn: Übertrag aus dem höchstwertigen Bit.
— cn-1: Übertrag in das höchstwertige Bit.

Rechenwerk – Flags

Dateneigänge X und Y

X_n X_{n-1} X_1 Y_n Y_{n-1} Y_1

f_1 f_2 f_3 f_4

Steuersignale

ALU

XOR

c_{n-1} c_n

=1

≥1

Z S C V

S_n S_{n-1} S_1

Datenausgänge

C_n	C_{n-1}	V
0	0	0
0	1	1
1	0	1
1	1	0

Flags im Statusregister

Halbaddierer (HA)

Wahrheitstabelle Halbaddierer:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Summe:

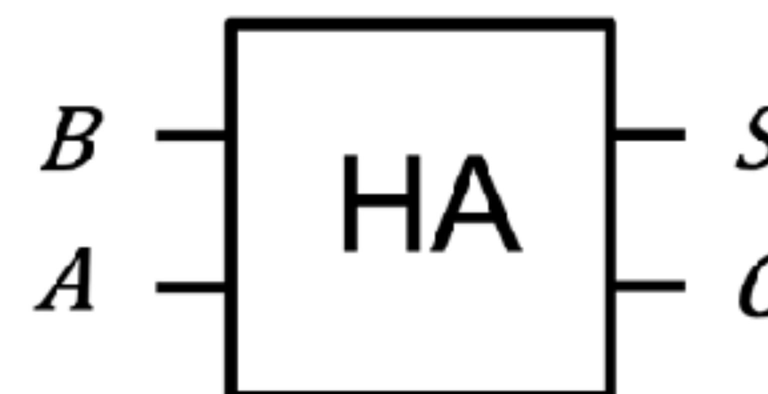
$$S = \overline{A}\,B + A\,\overline{B}$$

(Exklusiv-Oder)

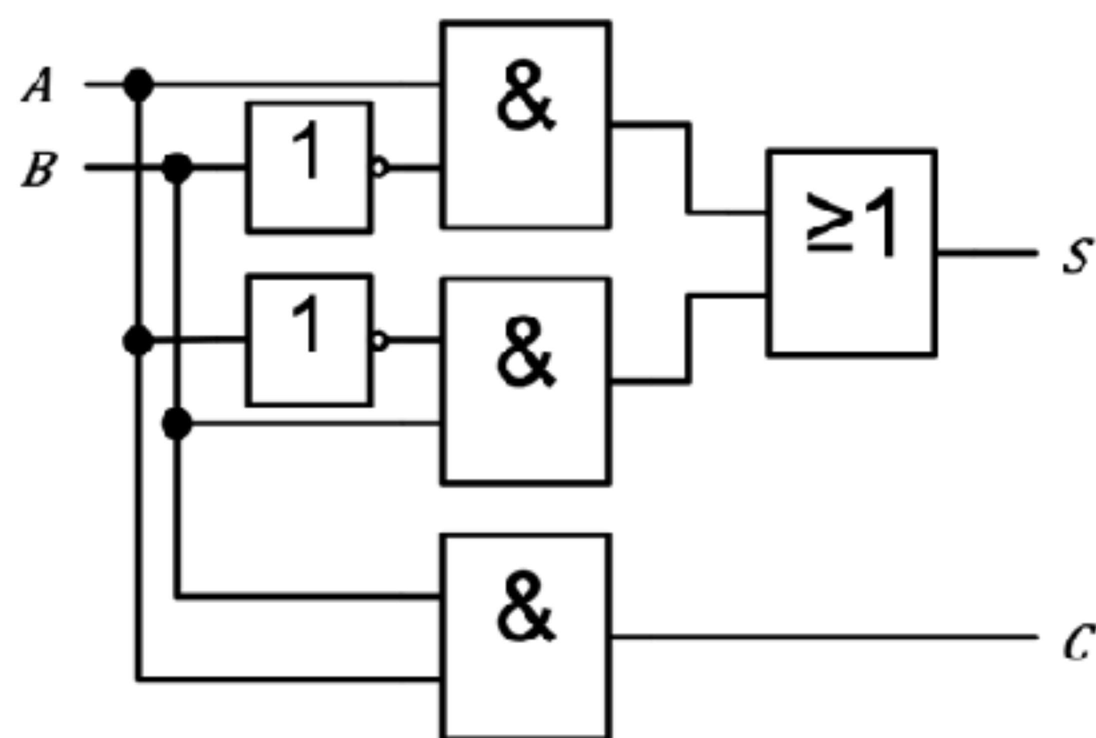
Carry:

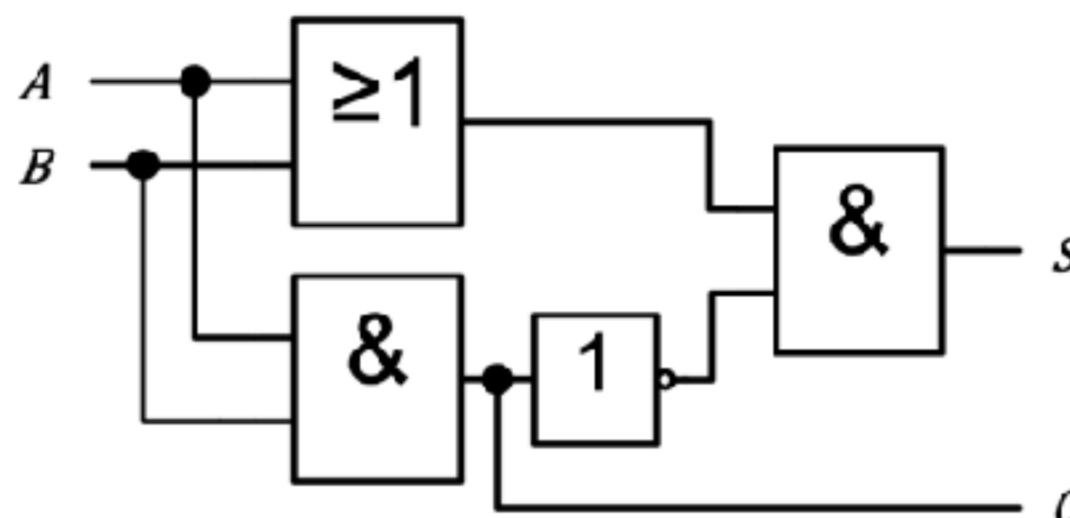
$$C = A\,B$$

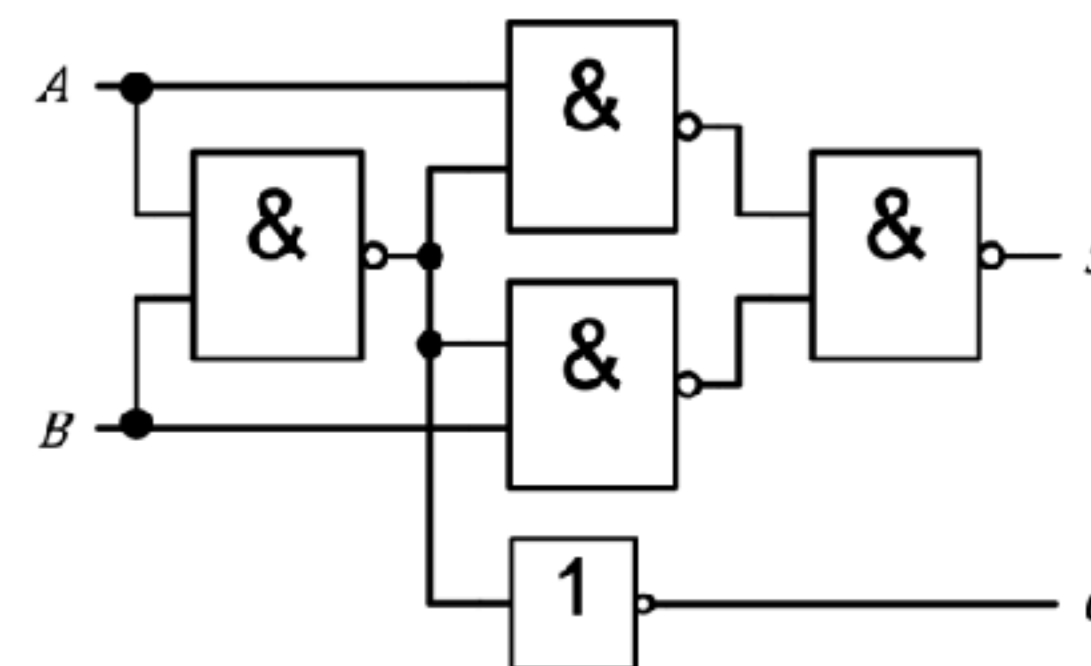
Symbol:



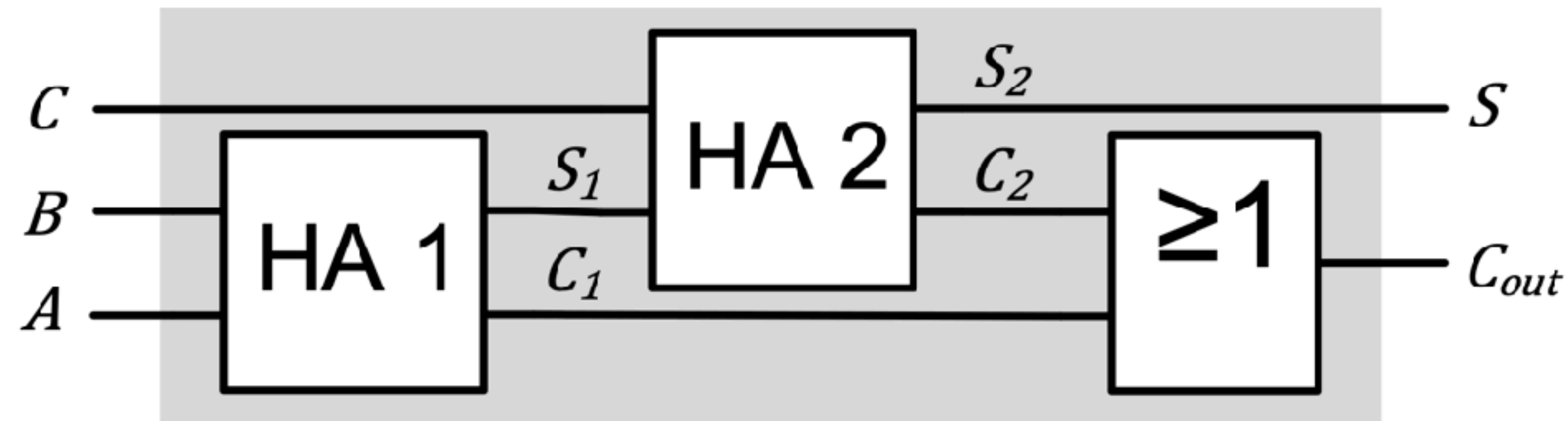
Implementierung:



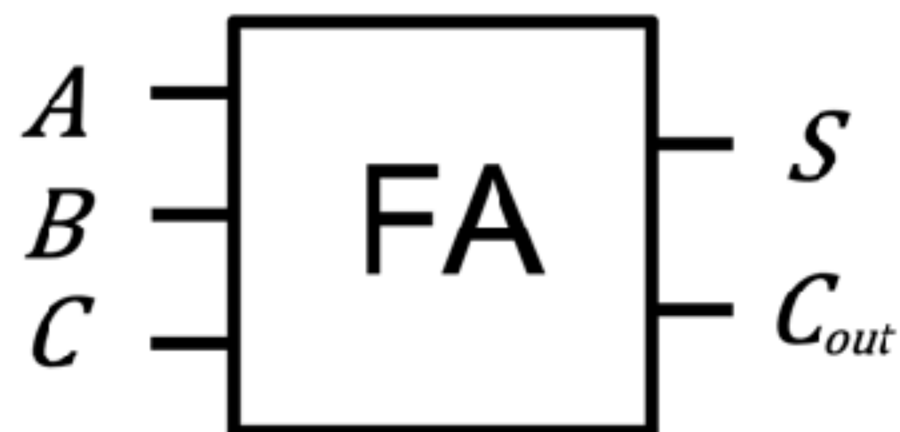




Volladdierer/Full Adder (FA)



Implementierung eines Volladdierers mittels zweier Halbaddierer

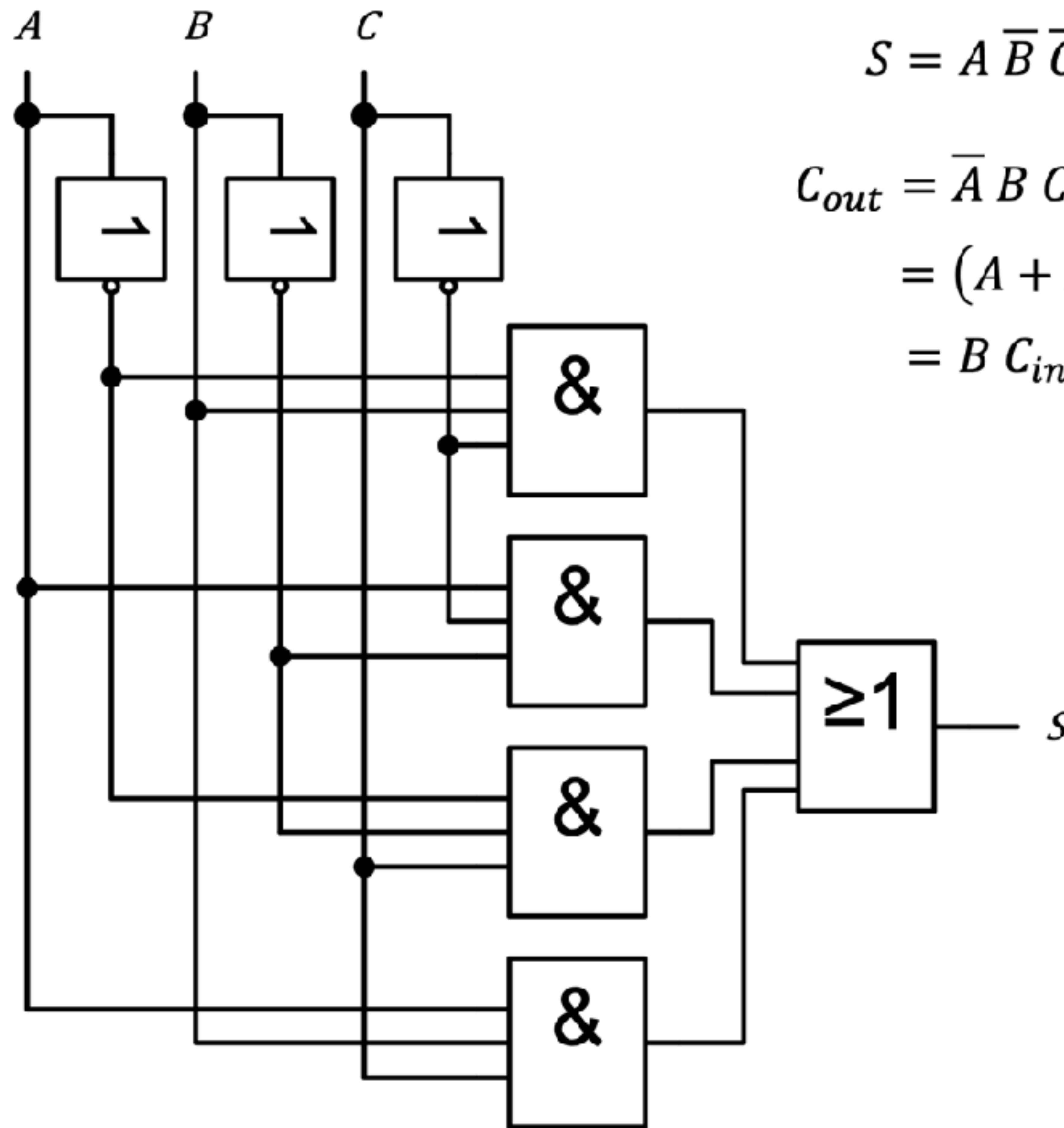


Symbol

Wahrheitstabelle des Volladdierers

C_{in}	A	B	S	C_{out}
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

Optimerte Schaltung für einen Volladdierer

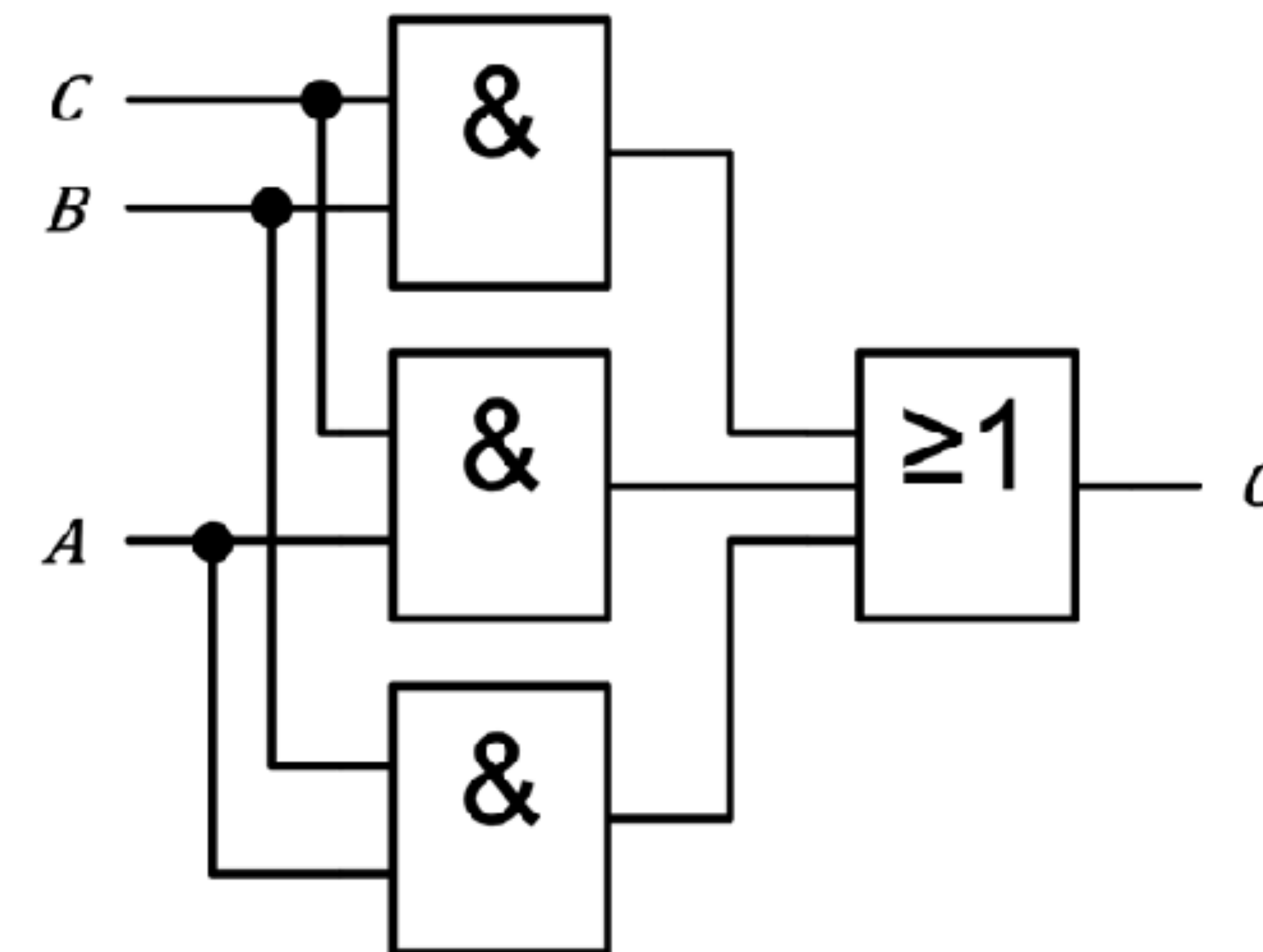


$$S = A\,\overline{B}\,\overline{C}_{in} + \overline{A}\,B\,\overline{C}_{in} + \overline{A}\,\overline{B}\,C_{in} + A\,B\,C_{in}$$

$$C_{out} = \overline{A}\,B\,C_{in} + A\,\overline{B}\,C_{in} + A\,B\,\overline{C}_{in} + A\,B\,C_{in}$$

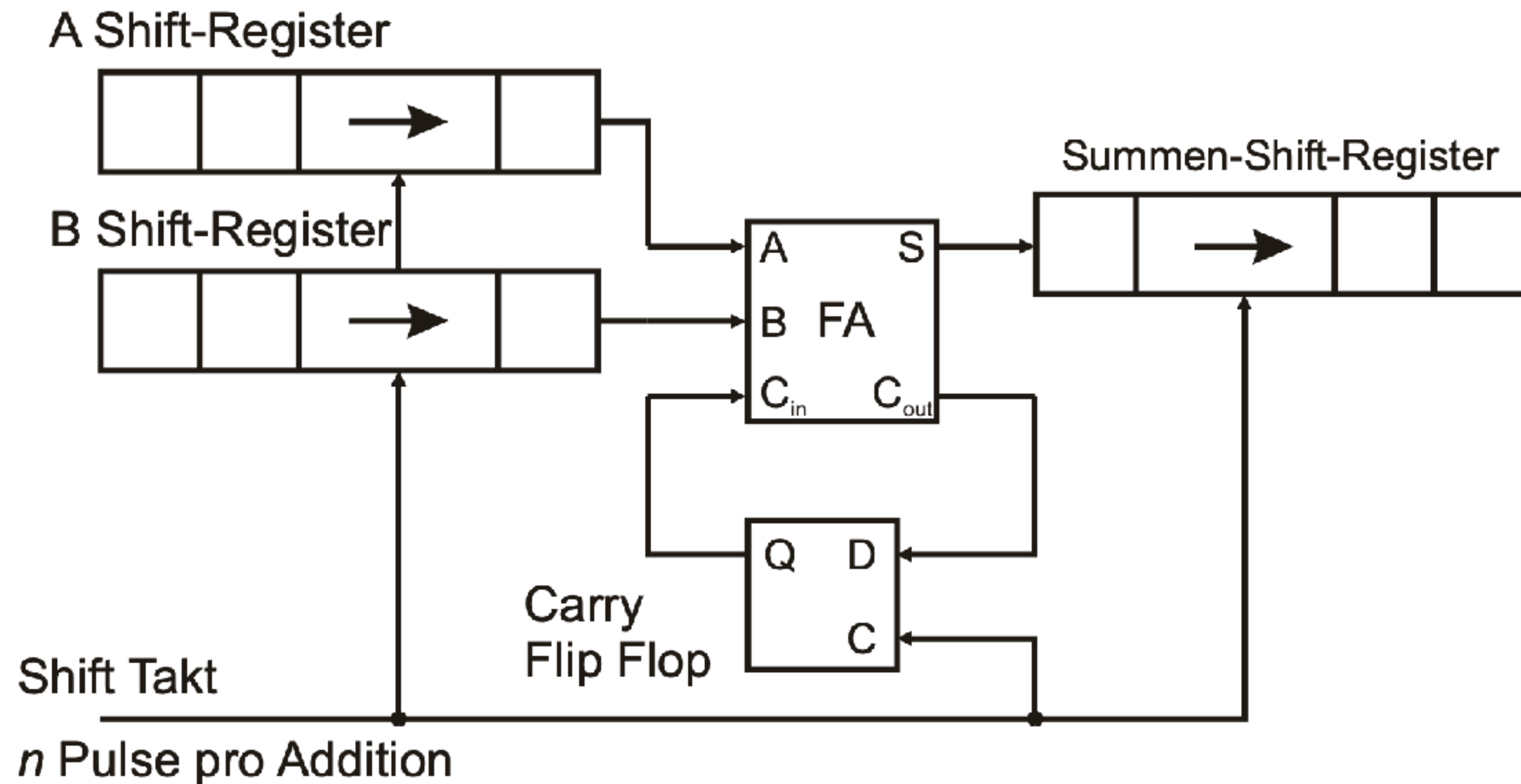
$$= (A + \overline{A})\,B\,C_{in} + A\,(B + \overline{B})\,C_{in} + A\,B\,(C_{in} + \overline{C}_{in})$$

$$= B\,C_{in} + A\,C_{in} + A\,B$$



Laufzeit: 3 x Gatterlaufzeit (GLZ) 2 x GLZ Gesamt: **3** x GLZ

Serieller Addierer

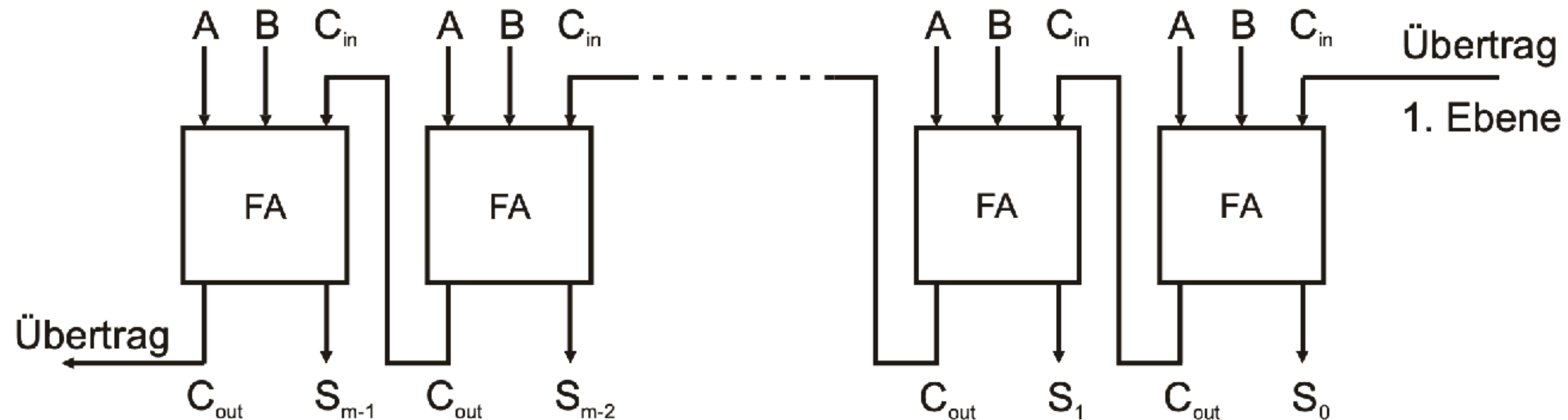


Rechenzeit: 3 GLZ pro Bit-Addition im FA x Anzahl Bits Datenwort

Bei 32 Bit Operanden gesamt 96 GLZ

Aufbau eines n-stelligen Parallel-Addierers

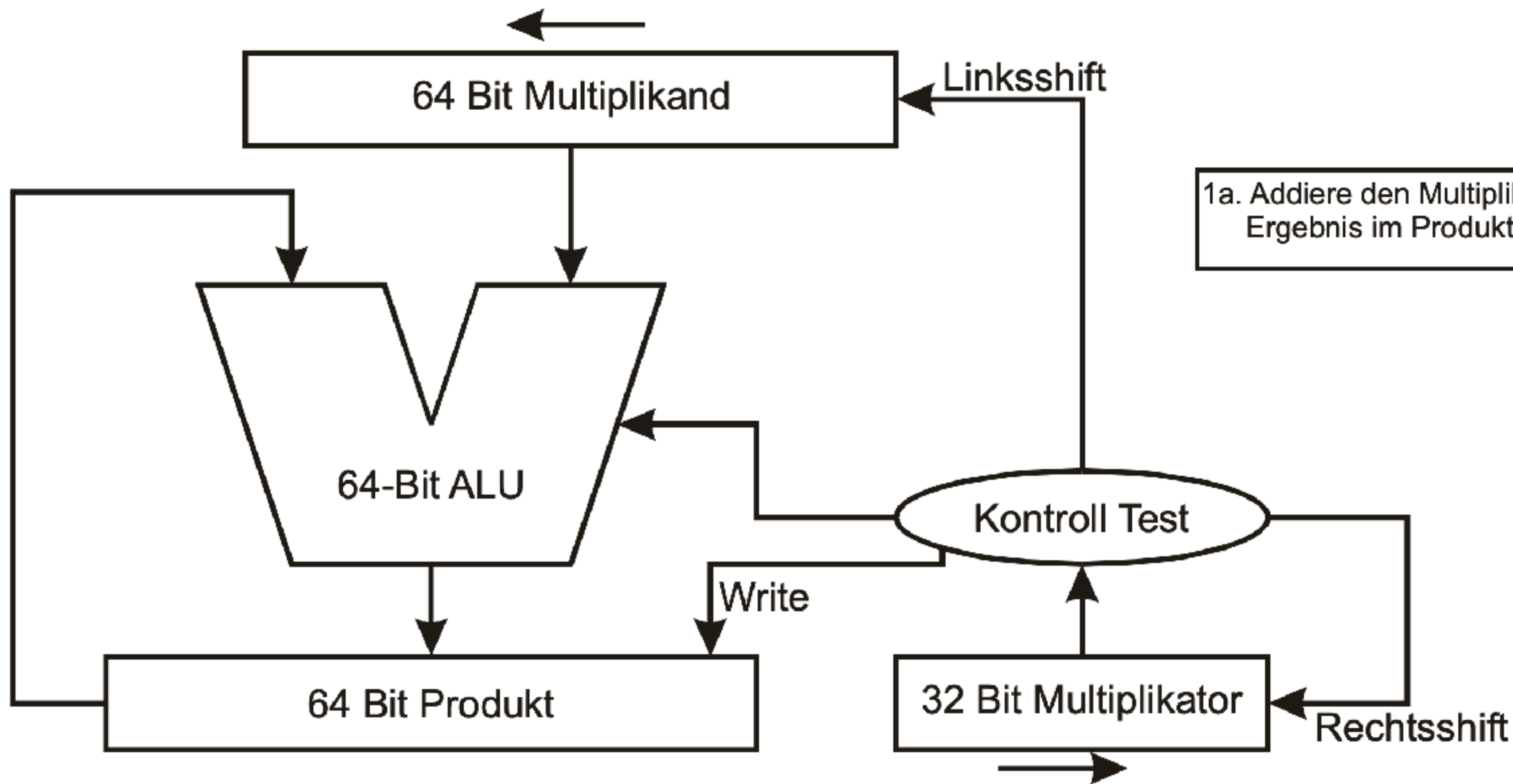
Auch Ripple Carry Adder genannt, da das Carry Bit durch die gesamte Breite durchläuft und Geschwindigkeit aufgrund Laufzeit vorgibt.

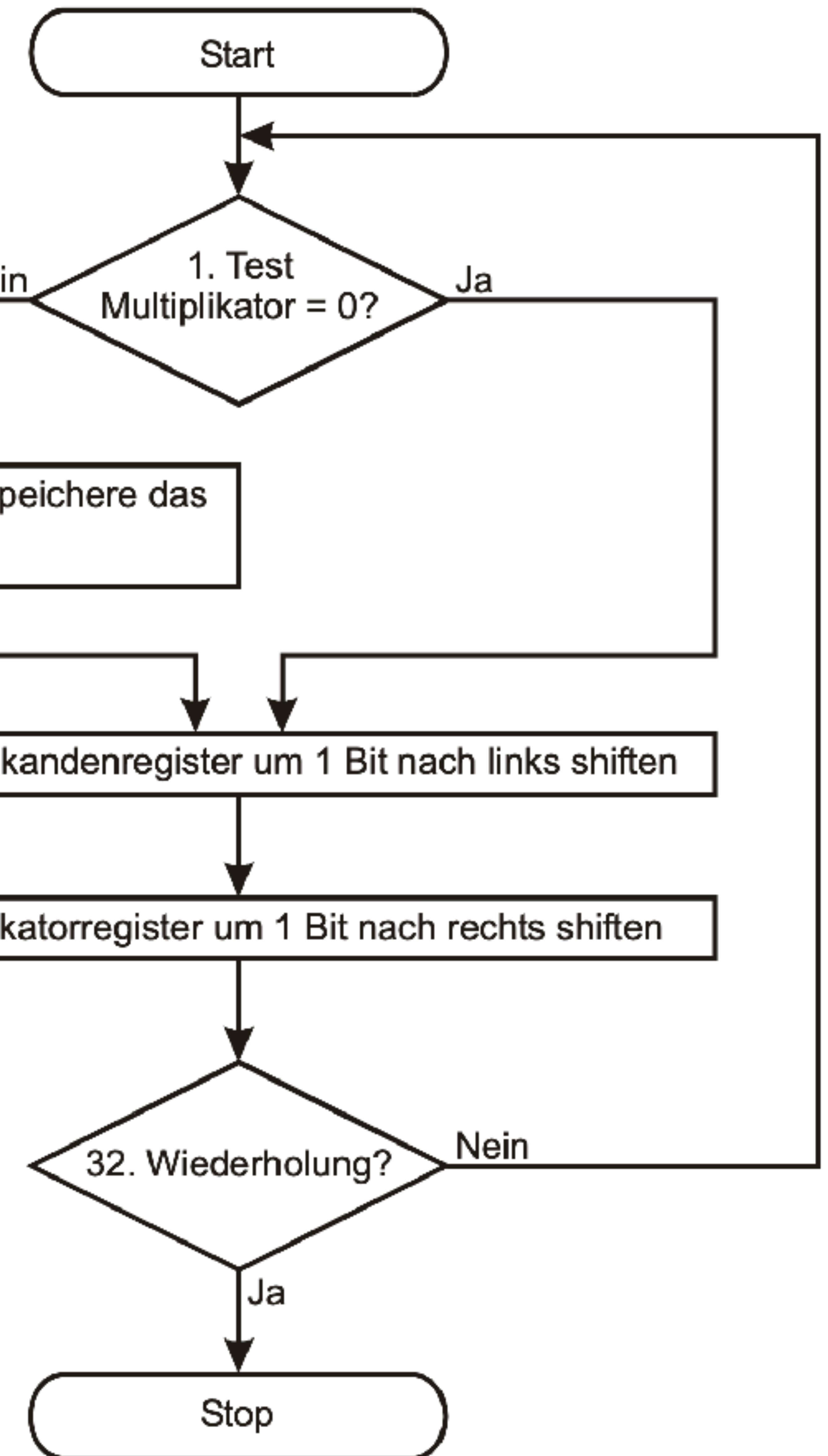


Rechenzeit: Stufen x 2 GLZ (Ripple Carry) + 1 x 3 GLZ (letzte Addition)

Bei 32 Bit Operanden gesamt 65 GLZ

Multiplikation





Multipliziererblock des MSP430

15 rw 0
MPY 130h
MPYS 132h
MAC 134h
Operand 1
(Address Determines Operation)
Operand 1
Operand 2 138h
Mode
16 × 16 Multiplier
Accessible Register
31 0
Product Register
32-Bit Adder
MPY, MPYS
MAC
32-Bit Multiplexer
Mode
000
MPY MPYS MAC
Mode
Multiplexer
SumExt 13 Eh
C
S
SumHi 13 Ch
SumLo 013 Ah
15 r 0
15 rw 0
15 rw 0

MPY: vorzeichenlose Multiplikation
MPYS: Multiplikation mit Vorzeichen
MAC: Multiply-and-Accumulate

Division

64 Bit Divisor
Rechtsshift
64-Bit ALU
32 Bit Quotient
Linksshift
64 Bit Rest
Schreiben
Kontroll Test

Aufgabe — ALU

1. Eine einfache ALU ist mit Logikgattern zu entwerfen:
   a) Nutzen Sie zuerst mehrere Volladdierer (VA) zur Realisierung der Addition zweier 4-Bit Operanden.
   b) Erweitern Sie den 4-Bit Addierer zu einer 4-Bit ALU, welche auch eine Subtraktion ausführen kann. Über das Steuersignal **`command = add/sub`** ($\overline{\text{add}}$/sub) soll die entsprechende Operation ausgeführt werden.
   c) Fügen Sie der Schaltung ein Statusregister mit den Flags **V**, **C** und **S** hinzu.
2. Entwerfen Sie ausgehend vom schriftlichen Multiplikationsschema eine Schaltung die zwei vorzeichenlose 4-Bit Zahlen multipliziert:
   a) Wie funktioniert die schriftliche Multiplikation Binärer Zahlen? Wie viele Bits kann das vollständige Produkt umfassen?
   b) Entwerfen Sie eine geeignete Multipliziererzelle (MC), die die Operation für eine Ziffer in der Produktmatrix realisiert.
   c) Entwerfen Sie einen kompletten 4×4 Feldmultiplizierer unter Nutzung dieser Multipliziererzelle.

Aufgabe — ALU

3. Gegeben sei eine **8-Bit**-Akkumulator-Architektur. Das Rechenwerk arbeitet im 2er-Komplement und bildet die üblichen Flags (Zero, Sign, Carry, Overflow).

   a) Die Ergebnisse und die Flags nach der Addition folgender Wertpaare mit Hilfe des ADD-Befehls sind zu berechnen:

      i. $27_{16} + 6C_{16}$

      ii. $E6_{16} + 1C_{16}$

      iii. $43_{16} + 84_{16}$

      iv. $9A_{16} + 66_{16}$

   b) Wann sind Ergebnisse in vorzeichenloser oder in vorzeichen-behafteter Interpretation von einer Bereichsüberschreitung betroffen? Wie kann das an den Flags abgelesen werden?

Aufgaben — ALU

4. Gegeben sei eine **8-Bit-Akkumulator-Architektur**, die über einen byteweise adressierten Speicher mit 10 Bit Adressraum verfügt. Das Rechenwerk arbeitet im 2er-Komplement und bildet die üblichen Flags. Datenworte werden in **Big Endian** abgelegt.

Mnem.	Maschinencode	Beschreibung	Aktualisierte Flags
JPf L	00<f:2><O:4>	Bedingter Sprung bei gesetztem Bedingungsflag.	—
JNf L	01<f:2><O:4>	Bedingter Sprung bei nicht gesetztem Bedingungsflag.	—
JMP L	1000 00<O.hi:2>, <O.lo:8>	Unbedingter Sprung.	—

- Die Assemblermnemoniken verwenden benannte Sprungmarken (Label).
- Die Sprungziele sind PC-relativ und bestimmen sich aus dem im 2er-Komplement vorliegenden, vorzeichenbehafteten Sprungoffset O relativ zur Adresse des regulären Folgebefehls.
- Das den Sprung bedingende Flag ist wie folgt kodiert:

Z	00	–	Zero	Null als Ergebnis.
C	01	–	Carry	Auslaufender Übertrag.
V	10	–	Overflow	Arithmetischer Überlauf (vorzeichenbehaftet).
S	11	–	Sign	Negatives Ergebnis.

Mnem.	Maschinencode		Beschreibung	Aktualisierte Flags
NOT	1000 0100	$A \leftarrow \bar{A}$	Invertiere Akkumulator bitweise.	Z, S
INC	1000 0110	$A \leftarrow A+1$	Inkrementiere Akkumulator.	Z, C, V, S
LD [P]	1100 00<P.hi:2>, <P.lo:8>	A←<P>	Lade Akkumulator mit Speicherwert an Adresse P.	—
ST [P]	1100 01<P.hi:2>, <P.lo:8>	<P>←A	Kopiere Akkumulator an Speicherplatz mit Adresse P.	—
ADD [P]	1100 10<P.hi:2>, <P.lo:8>	A←A+<P>	Addiere Speicherwert an Adresse P zum Akkumulator.	Z, C, V, S
ADC [P]	1100 11<P.hi:2>, <P.lo:8>	A←A+<P>+C	Addiere Speicherwert und Carry-Flag zum Akkumulator.	Z, C, V, S

Aufgaben — ALU

Fortsetzung Aufgabe 4:

a) Wie groß ist der maximal adressierbare Speicherbereich?

b) Disassemblieren Sie den Maschinencodebefehl 00_{16}. Welche Auswirkungen hat er? Welche alternative Bezeichnung für den Befehl bietet sich an?

c) Die **16-Bit** Variablen a und b liegen ab der Adresse 0x200, bzw. 0x202 im Speicher. Schreiben Sie ein Programm, das $c = a + b$ berechnet und das Ergebnis c im Speicher auf Adresse 0x204 ablegt. Geben Sie sowohl den Befehl als auch den hexadezimalen Maschinencode an.

d) Wie kann der Befehl ADC [0x202] durch andere Befehle abgebildet werden, wenn er nicht zur Verfügung steht?

e) Erweitern Sie die Lösung so, dass $c = |a + b|$ berechnet wird.

END