Computerarithmetik (1)

Fragen:

- Wie werden Zahlen repräsentiert und konvertiert?
- Wie werden negative Zahlen und Brüche repräsentiert?
- Wie werden die Grundrechenarten ausgeführt?
- Was ist, wenn das Ergebnis einer Operation größer ist als die größte darzustellende Zahl?

Hauptunterschied zwischen Computer- und menschlicher Arithmetik:

- Computer arbeiten mit einer anderen Zahlendarstellung
- Genauigkeit der sowie Platzbedarf für die Darstellung von Zahlen sind beim Computer endlich und begrenzt.

Rechner speichern die Information (Zahlen) in Einheiten festgesetzter Bitlänge, genannt **Worte.** So dargestellte Zahlen heißen **„Zahlen mit begrenzter Genauigkeit.“**

Prozessortyp	Wortlänge (in Bits)
8085, Z80, 6809	8
8086, 68000	16
80386, 68020	32
Pentium, PowerPC (Sun SPARC, IBM AIX)	32
typischer Mikrocontroller	4
Cray-1 Supercomputer	64

Computerarithmetik (2)

Beispiel für Zahlendarstellung mit unterschiedlichen Basen:

binär

$$\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 \cdot 2^{10} & + 1 \cdot 2^{9} & + 1 \cdot 2^{8} & + 1 \cdot 2^{7} & + 1 \cdot 2^{6} & + 1 \cdot 2^{5} & + 1 \cdot 2^{4} & + 1 \cdot 2^{3} & + 1 \cdot 2^{2} & + 1 \cdot 2^{1} & + 1 \cdot 2^{0} \\ 1024 & + 512 & + 256 & + 128 & + 64 & + 0 & + 16 & + 0 & + 0 & + 0 & + 1 \end{array}$$

oktal

$$\begin{array}{cccc} 3 & 7 & 2 & 1 \\ 3 \cdot 8^{3} & + 7 \cdot 8^{2} & + 2 \cdot 8^{1} & + 1 \cdot 8^{0} \\ 1536 & + 448 & + 16 & + 1 \end{array}$$

dezimal

$$\begin{array}{cccc} 2 & 0 & 0 & 1 \\ 2 \cdot 10^{3} & + 0 \cdot 10^{2} & + 0 \cdot 10^{1} & + 1 \cdot 10^{0} \\ 2000 & + 0 & + 0 & + 1 \end{array}$$

hexa-dezimal

$$\begin{array}{ccc} 7 & D & 1 \\ 7 \cdot 16^{2} & + 13 \cdot 16^{1} & + 1 \cdot 16^{0} \\ 1792 & + 208 & + 1 \end{array}$$

Computerarithmetik (3)

Kollektion von Zahlendarstellungen mit den 4 verschiedenen Basen:

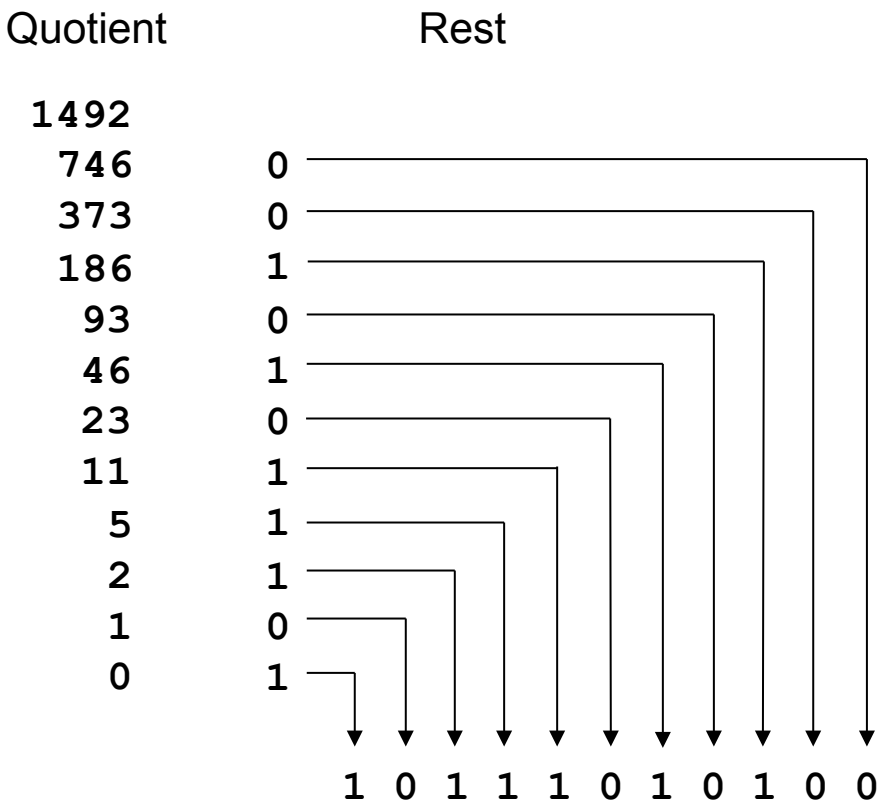
dezimal	binär	oktal	hexadezimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
7	111	7	7
8	1000	10	8
10	1010	12	A
11	1011	13	B
12	1100	14	C
15	1111	17	F
16	10000	20	10
20	10100	24	14
50	110010	62	32
60	111100	74	3C
80	1010000	120	50
100	11001000	144	64
1000	1111101000	1750	3E8
2989	101110101101	5655	BAD

Computerarithmetik (4)

Tabelle für Umwandlung binär - hexadezimal:

Hexa-dezimal	binär	Hexa-dezimal	binär
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Beispiel für Konversion einer Dezimalzahl in eine Binärzahl:



Computerarithmetik (5)

BCD (Binary Coded Decimal):
weitere Möglichkeit der Zahlendarstellung mit Hilfe von nur 2 Ziffern, aber im Dezimalsystem verbleibend.

Prinzip:
Jede Dezimalziffer wird für sich in die entsprechende Binärzahl konvertiert.
(Analogie zum Binärblock mit den Basen 2, 8, 16)

Vorteil:
sehr einfache Konvertierung von dezimaler zu binärer Darstellung

Nachteile:

- komplexere Arithmetik
- verschwenderische (ineffiziente) Ausnutzung der zur Verfügung stehenden Wortbreite und damit des gesamten Speichers

Konsequenz:
Einsatz nur in Applikationen mit sehr geringem Speicherbedarf

Beispiele:
Taschenrechner, Digitaluhr

Computerarithmetik (6)

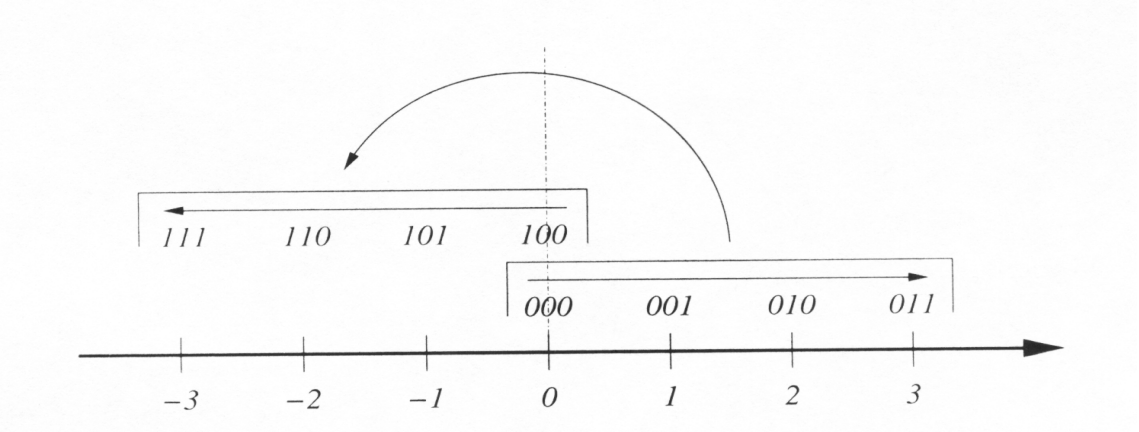
Darstellung ganzer Zahlen (signed numbers)

1. Die Vorzeichen/Betrags - Darstellung (sign and magnitude):

Das höchstgewichtete Bit wird exklusiv für die Angabe des Vorzeichens genutzt.
(Exklusiv heißt: Das Vorzeichenbit ist nicht Teil der Zahl)

Sei S (=d_n) das Vorzeichenbit und N der Betrag (Größe) einer ganzen Zahl Z,
dann ist ihr Wert gegeben durch: $Z =: (-1)^{d_n} N$

Beispiel für n=3:



Der Wertebereich bei einem gegebenen n-bit-Wort liegt im Intervall $[-(2^{n-1}-1), 2^{n-1}-1]$
---> Der Zahlenbereich ist symmetrisch bzgl. des Nullpunkts
---> keine eindeutige Darstellung der Null (-0, +0)

Weitere Nachteile:

- erfordert separates Subtrahierwerk
- erfordert zusätzliche Logik, um zu entscheiden, welches Vorzeichen das Ergebnis der Operation hat

2. Die Komplement - Darstellung

Das höchstgewichtete Bit wird weiterhin (aber nicht exklusiv) für die Angabe des Vorzeichens genutzt, d.h. das Vorzeichenbit ist Teil des Summanden und wird in eine arithmetische Operation mit eingeschlossen

- → Subtraktion wird auf die Addition zurückgeführt
- → Keine Notwendigkeit für ein zusätzliches Subtrahierwerk
- → Keine zusätzliche Logik zur Bestimmung des Vorzeichens

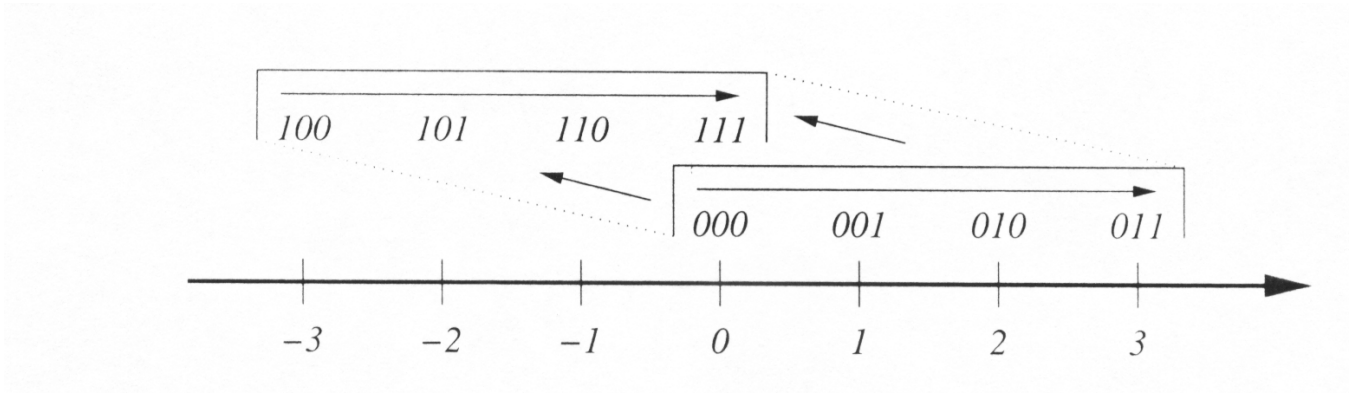
2a. Einer - Komplement

Sei N der Betrag (Größe) einer negativen ganzen Zahl Z (d.h. d_n=1). Dann gilt:
$-N = N - (2^n - 1)$

Das Einer - Komplement - N einer positiven binären Zahl N aus $[0 , 2^{n-1}-1]$ erreicht man durch bitweises Invertieren von N ---> -N aus $[-0, -2^{n-1}-1]$

Computerarithmetik (7)

Beispiel für n=3:



Subtraktion:= Addition + end-around-carry, d.h. zu der Summe wird das Übertragsbit aufaddiert.

Vorteil: zusätzliches Subtrahierwerk überflüssig

Nachteile:
- keine eindeutige Darstellung der Null
- kein echtes Komplement, da -x + x $\neq$ 0 sondern = - 0, also 111....1

Computerarithmetik (7b)

2b. Zweier - Komplement

Sei N der Betrag (Größe) einer negativen ganzen Zahl Z (d.h. d_n=1). Dann gilt:

$-N = N - 2^n$

→ -N = Einer - Komplement + 1

→ -N = (bitweises Invertieren von N) + 1

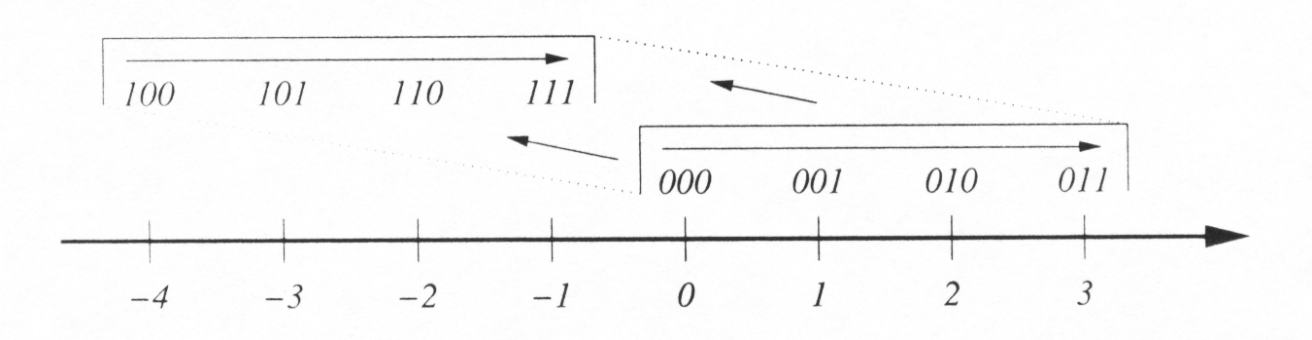
→ Es gibt eine eindeutige Darstellung der Null (0.....0)

→ -N aus $[-1, -2^{n-1}]$

→ Der Wertebereich des Zweier - Komplements ist $[-2^{n-1}, 2^{n-1}-1]$

→ Das Zweier - Komplement ist ein echtes Komplement: $N+(-N) = 2^n = (0.....0)$

Beispiel für n=3:



Computerarithmetik (8a)

Einfache Additions (Subtraktions-) Regeln

x+y: Addition der entsprechenden 2er - Komplemente ergibt korrekte Summe im 2er - Komplement, solange der Wertebereich nicht überschritten wird.

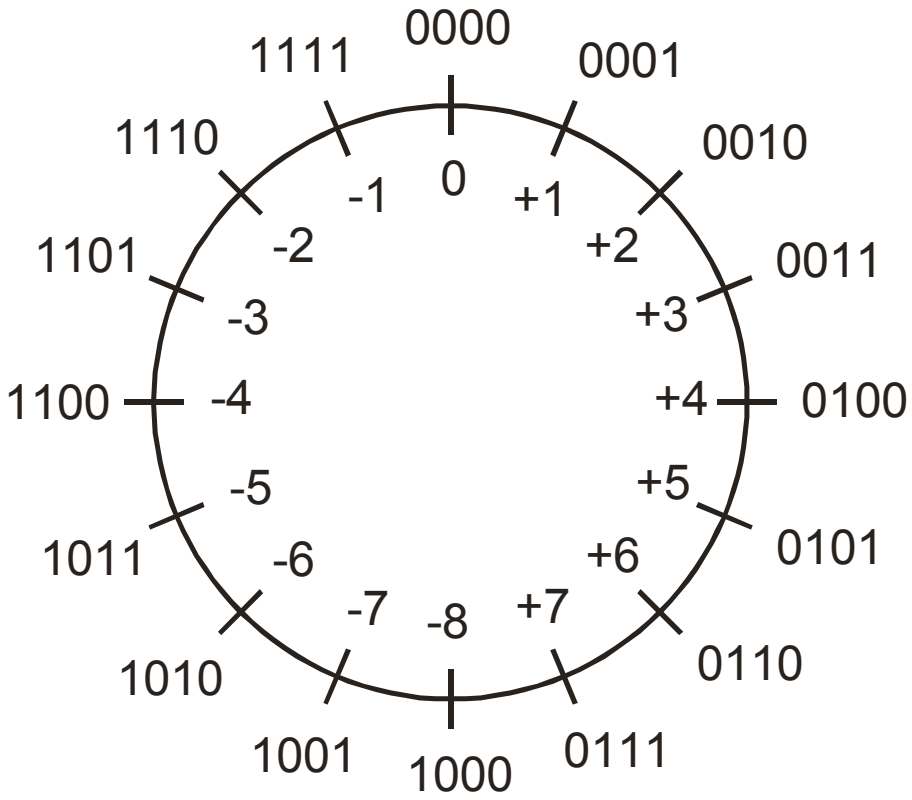
x-y = x+(-y): Bilde das 2er - Komplement von y und führe Addition wie oben aus.

Konsequenz:

Die logische Einfachheit und die daraus resultierende Geschwindigkeit (arithmetische Operation erfolgt immer in einem Schritt) führt dazu, dass das Zweier - Komplement in den ALU´s moderner Rechner eingesetzt wird.

Computerarithmetik (8)

Visualisierung des Zweier - Komplements sowie der Addition



Computerarithmetik (9)

Overflow (Summe liegt außerhalb des Wertebereiches):

Wichtig: Erkennung des Overflows

- Bei Integer-Addition dient das carry-out-Bit als Overflow-Indikator.
- Bei Addition ganzer Zahlen (signed numbers) gilt dies nicht
- Addition von Summanden mit unterschiedlichem Vorzeichen ergibt **nie** einen Overflow (Absoluter Wert ihrer Summe ist immer kleiner als der absolute Wert von einem der beiden Summanden)

Folgerung: Overflow nur möglich, wenn beide Summanden das gleiche Vorzeichen haben

Prüfung auf Overflow:

$$O = a_{n-1}\, b_{n-1}\, \overline{s_{n-1}} + \overline{a_{n-1}}\, \overline{b_{n-1}}\, s_{n-1}$$

(Die Faktoren repräsentieren die Vorzeichenbits der Summanden a und b sowie der Summe s)

Gilt O = 1 $\longrightarrow$ Es existiert ein Overflow!

Computerarithmetik (27)

Kriterien für die Qualität der Zahlendarstellung:

- Größe des darstellbaren Zahlenbereichs (**range**)
- Genauigkeit (**precision**) der Zahlendarstellung

Diese beiden Kriterien sind prinzipiell **unabhängig** voneinander.

Wissenschaftliche Notation: $d = a \times r^E$

a Mantisse (Argument), r Radix (Basis), E Exponent (Charakteristik)

Parameter für mögliche Darstellungen von Floating point - Zahlen :

- Anzahl der insgesamt verfügbaren Bits (Wortlänge bzw. Worte)
- Anzahl der verfügbaren Bits jeweils für Mantisse bzw. Exponent (Trade-off!)
- Darstellung von Mantisse und Exponent
- Lokalisierung (Mantisse vor Exponent oder umgekehrt)

Mantissendarstellung in normierter Form:

$d = (-1)^s \times a \times 2^E$ mit s als Vorzeichenbit und $1 \leq a < 2$

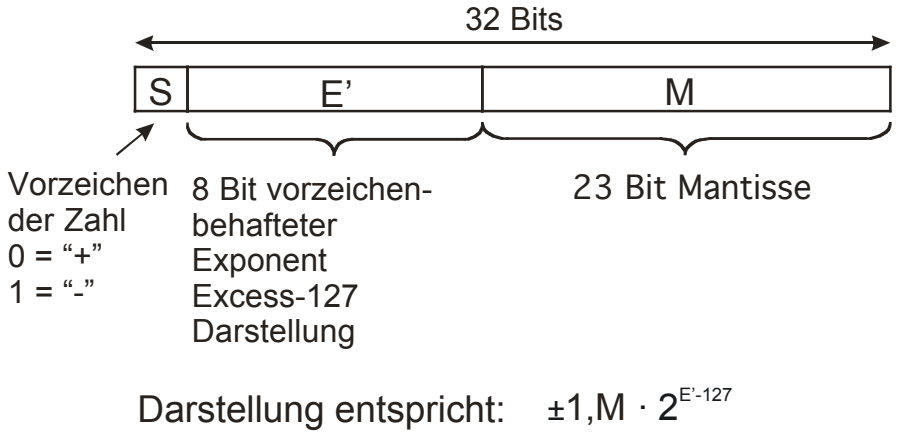
Exponentendarstellung mit Bias:

$d = (-1)^s \times a \times 2^{E'}$ mit $E' := E + 127$

Computerarithmetik (27a)

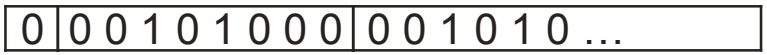
Darstellung im IEEE Standard 754:

Einfache Genauigkeit:



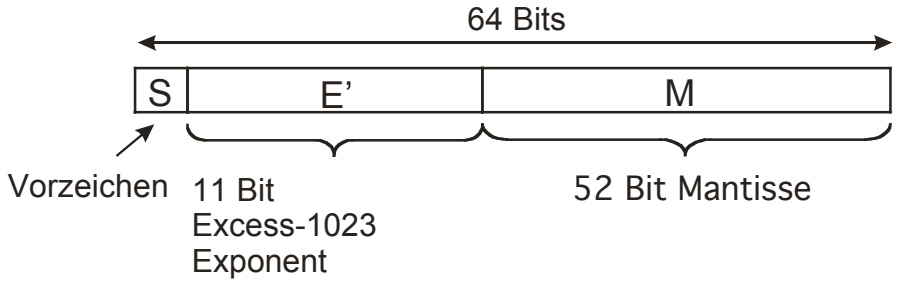
Darstellung entspricht: $\pm 1{,}M \cdot 2^{E'-127}$

Beispiel mit
Einfacher Genauigkeit:



Darstellung entspricht: $1{,}001010...0 \cdot 2^{-87}$

Doppelte Genauigkeit:



Darstellung entspricht: $\pm 1{,}M \cdot 2^{E'-1023}$

Computerarithmetik (28)

Verallgemeinerter Additions/Subtraktions - Algorithmus:

- Rechtsshift auf der Mantisse des kleineren Operanden zur Angleichung der Exponenten
  ----> Exponent der Summe/Differenz := Exponent des größten Operanden
- Addition/Subtraktion der Mantissen und Bestimmung des Vorzeichens
- Wenn nötig, Normalisierung des Ergebnisses

Verallgemeinerter Multiplikations/Divisions- Algorithmus:

- Multipliziere/Dividiere die Mantissen und bestimme das Vorzeichen
- Wenn nötig, normalisiere das Ergebnis
- Addiere/Subtrahiere die Exponenten und subtrahiere/addiere 127_{10}

Rundung: Kappung überzähliger Bits durch

- Abspalten (chopping)
- von Neuman - runden
- runden

Computerarithmetik (28a)

(vorläufige) Zusammenfassung:

- Computerarithmetik ist endlich und kann folglich **nicht** übereinstimmen mit der natürlichen Arithmetik

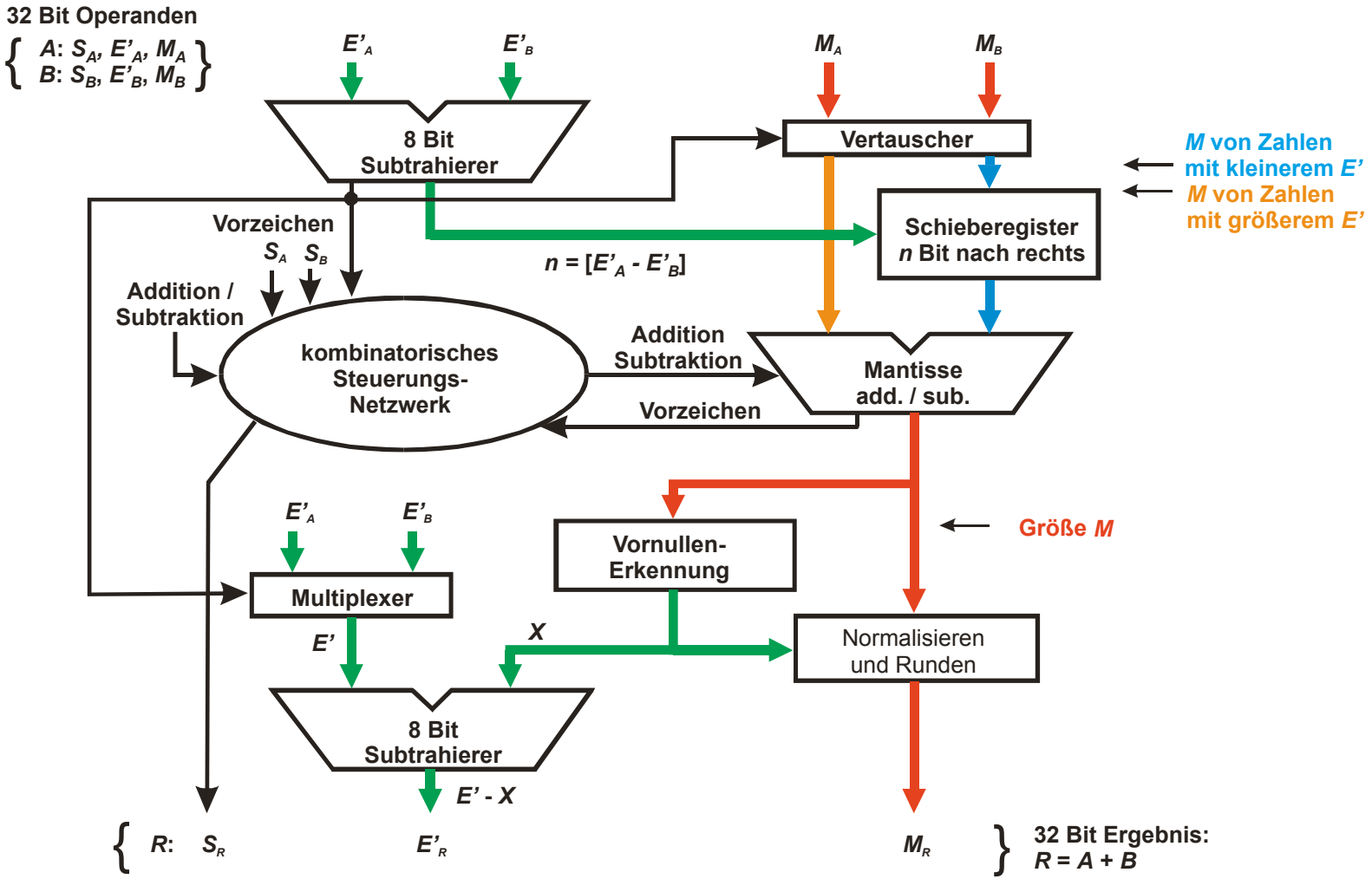
- Selbst der IEEE 754 - Standard für die Fließkomma - Darstellung, wie jede andere auch, ist fast immer eine **Approximation** der realen Zahlen.

- Rechnersysteme müssen dafür sorgen, den daraus resultierenden Unterschied zwischen Computerarithmetik und Arithmetik in der realen Welt möglichst zu minimieren.

- Informatiker sollten sich dieser Zusammenhänge bewusst sein.

Computerarithmetik (29)

Beispiel für die HW - Implementierung einer Addition/Subtraktion:



Computerarithmetik (10)

Wahrheitstabelle Halbaddierer:

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

Summe:

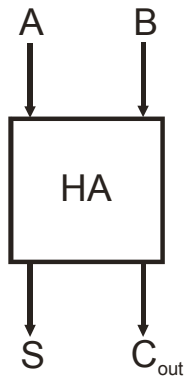
$S = \overline{A}B + A\overline{B} = A \oplus B$

(Exklusiv-Oder)

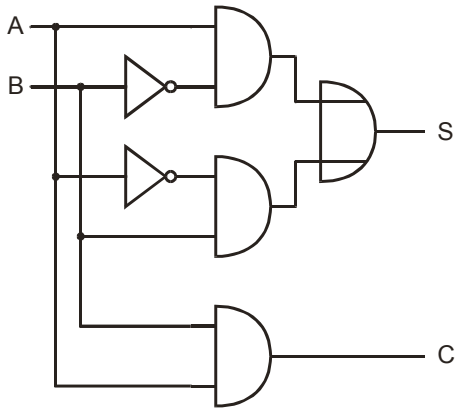
Carry:

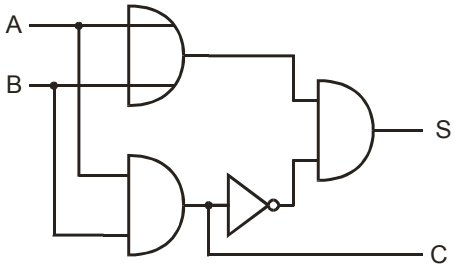
$C = AB$

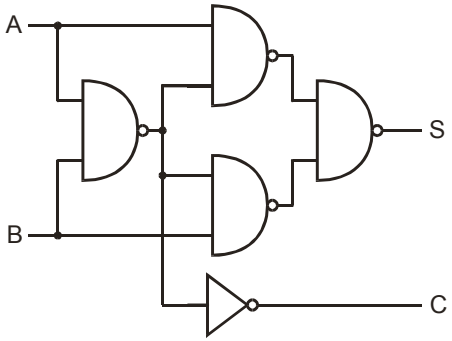
Symbol:



Implementierung:





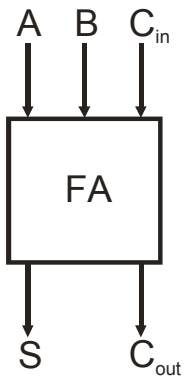


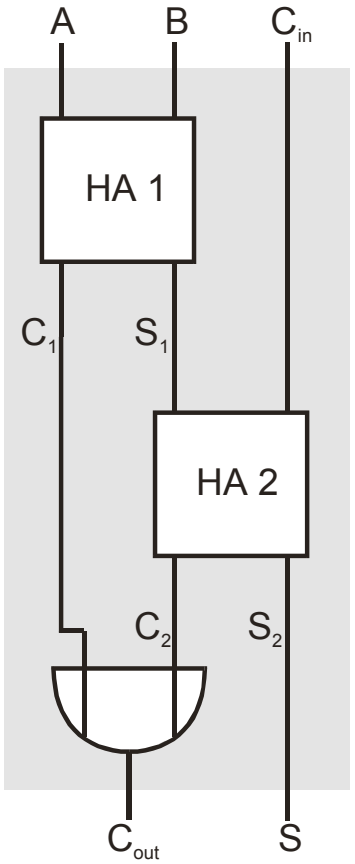
Computerarithmetik (11)

Wahrheitstabelle Volladdierer:

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

Symbol:

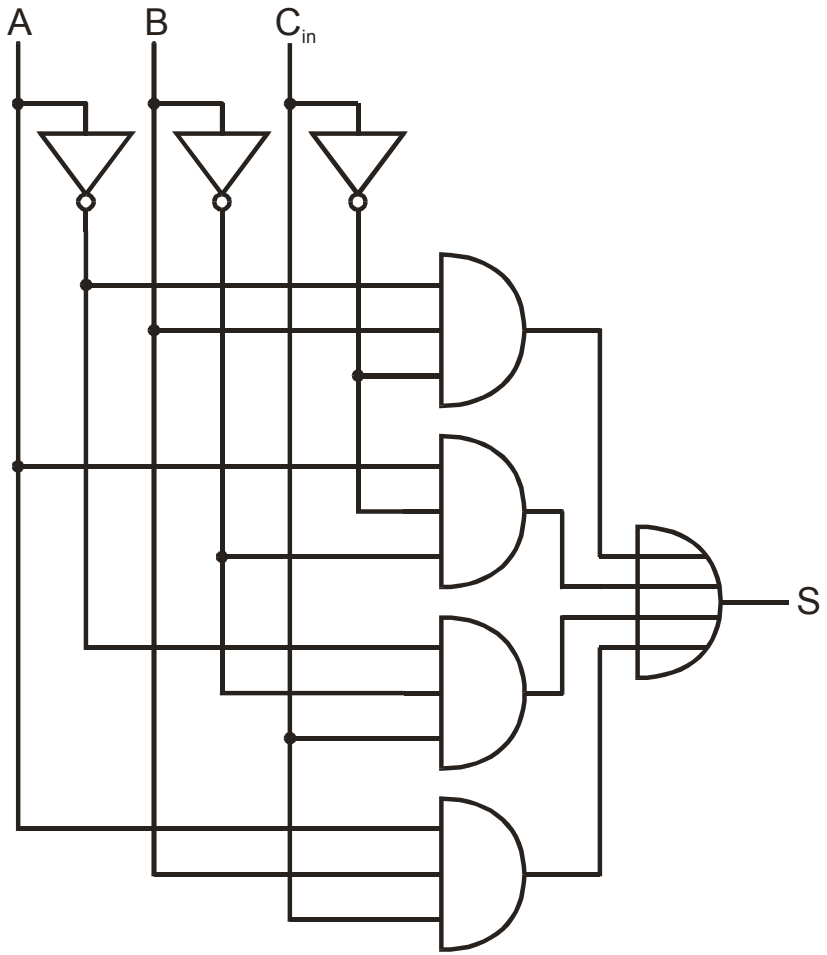


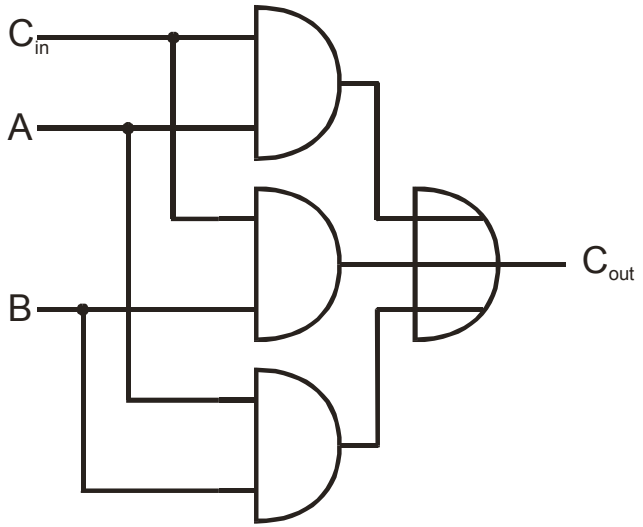


Implementierung eines Volladdierers mittels zweier Halbaddierer

Computerarithmetik (12)

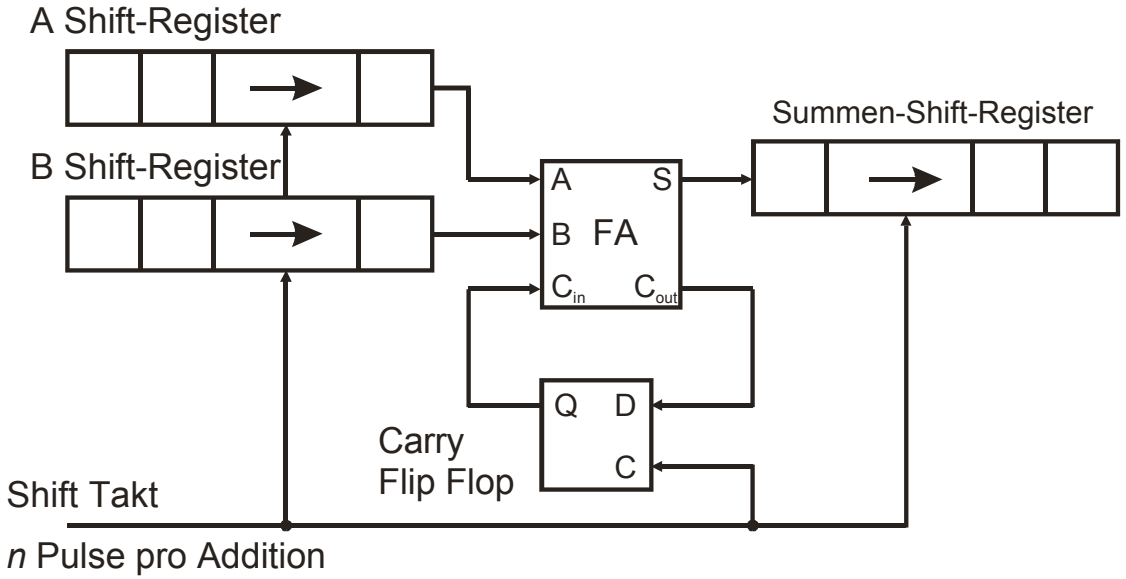
Schaltung für einen Volladdierer:



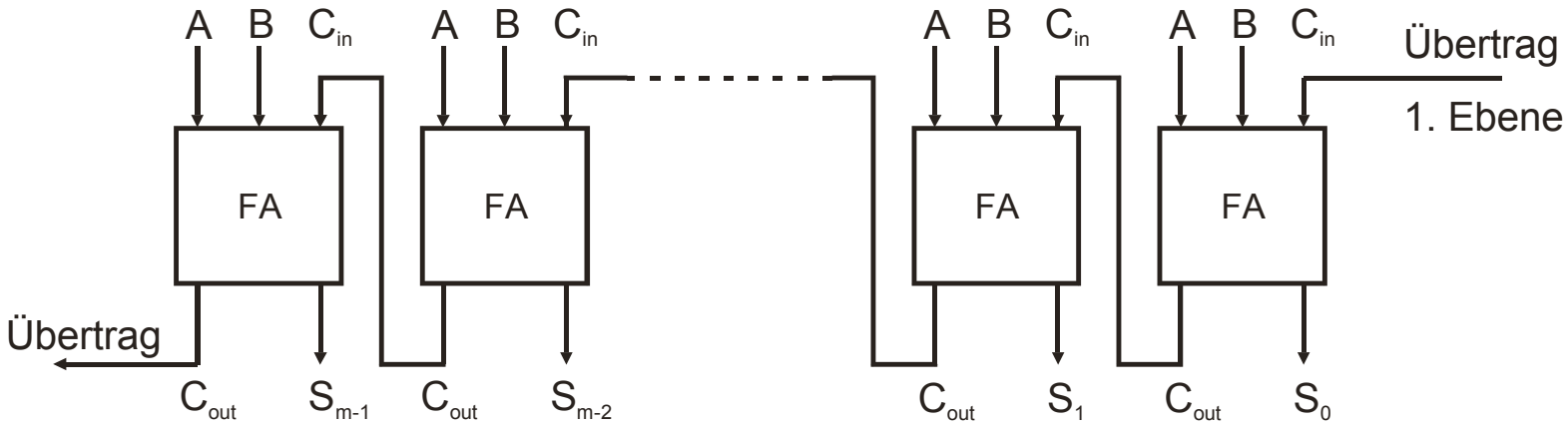


Computerarithmetik (13)

Serieller Addierer:

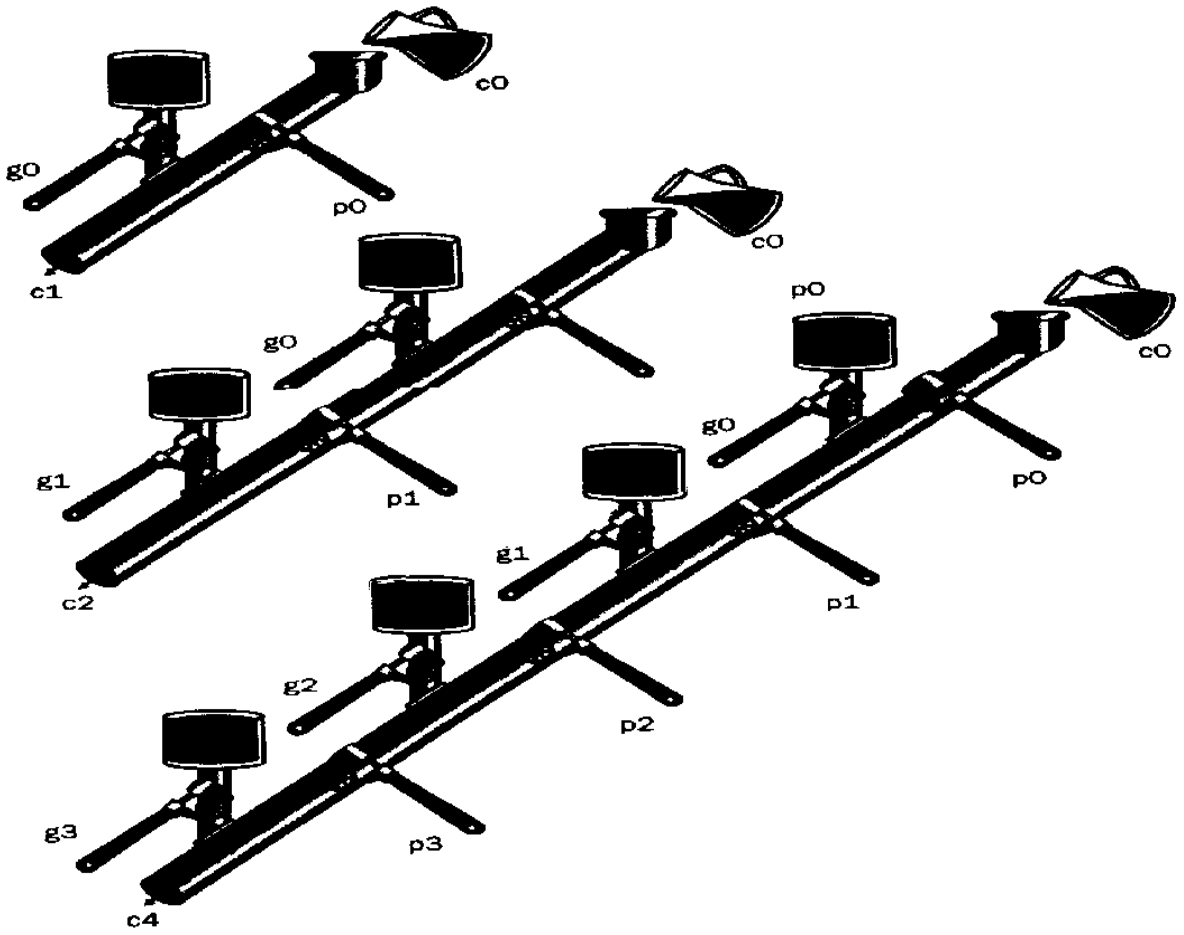


Paralleladdierer:



Computerarithmetik (14)

Rohrleitungsanalogie für Carry-lookahead:



Computerarithmetik (14a)

Rechenzeit zur Addition zweier 32 Bit-Zahlen

A) serieller Addierer

3 GLZ pro Additionsschritt
(Mindestzeit, da Taktung entscheidend)

32 * 3 GLZ = 96 GLZ

B) Carry Ripple Adder (CRA)

31 * 2 GLZ (Ripple Carry)	= 62 GLZ
1 * 3 GLZ (letzte Addition)	= 3 GLZ
	65 GLZ

C) Carry-Lookahead Adder (CLA)

Zusammenschaltung von m 4-Bit CLA (im Beispiel m=8)

1 * 1 GLZ (g_i und p_i)	= 1 GLZ
8 * 2 GLZ (c_1 ... c_4)	= 16 GLZ
1 * 3 GLZ (letzte Addition)	= 3 GLZ
	20 GLZ

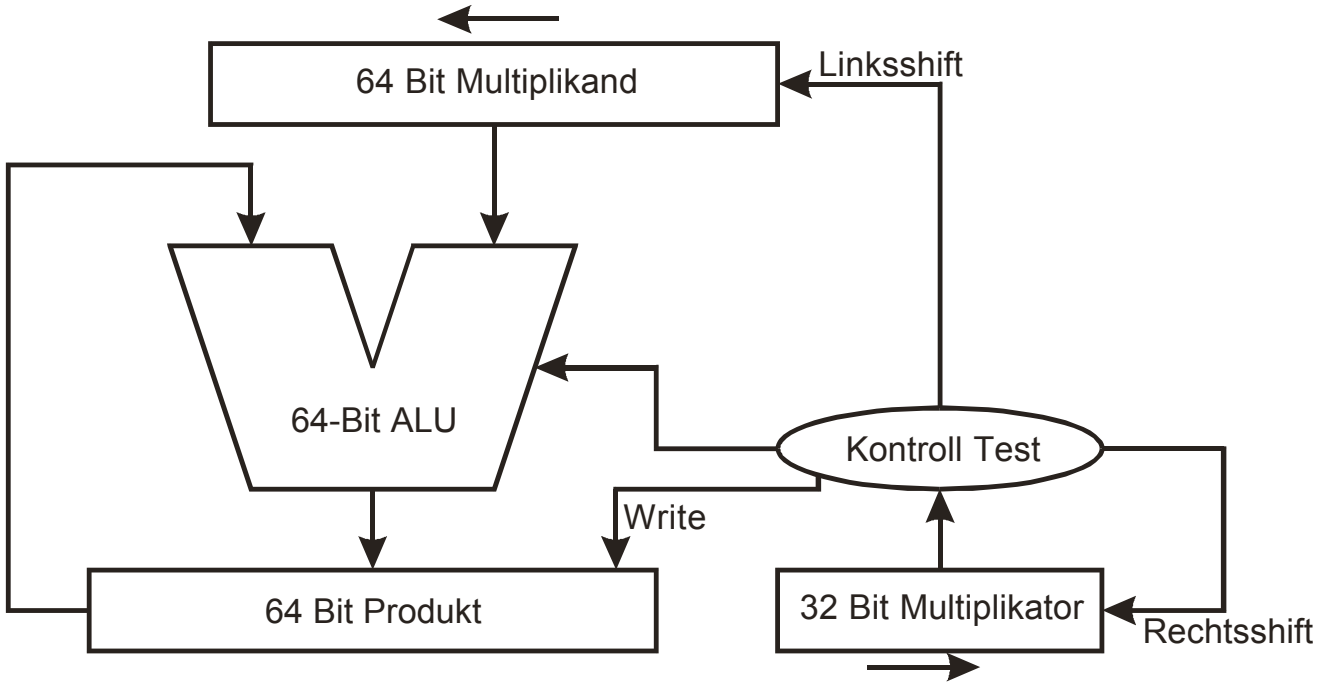
Computerarithmetik (15)

Multiplikation

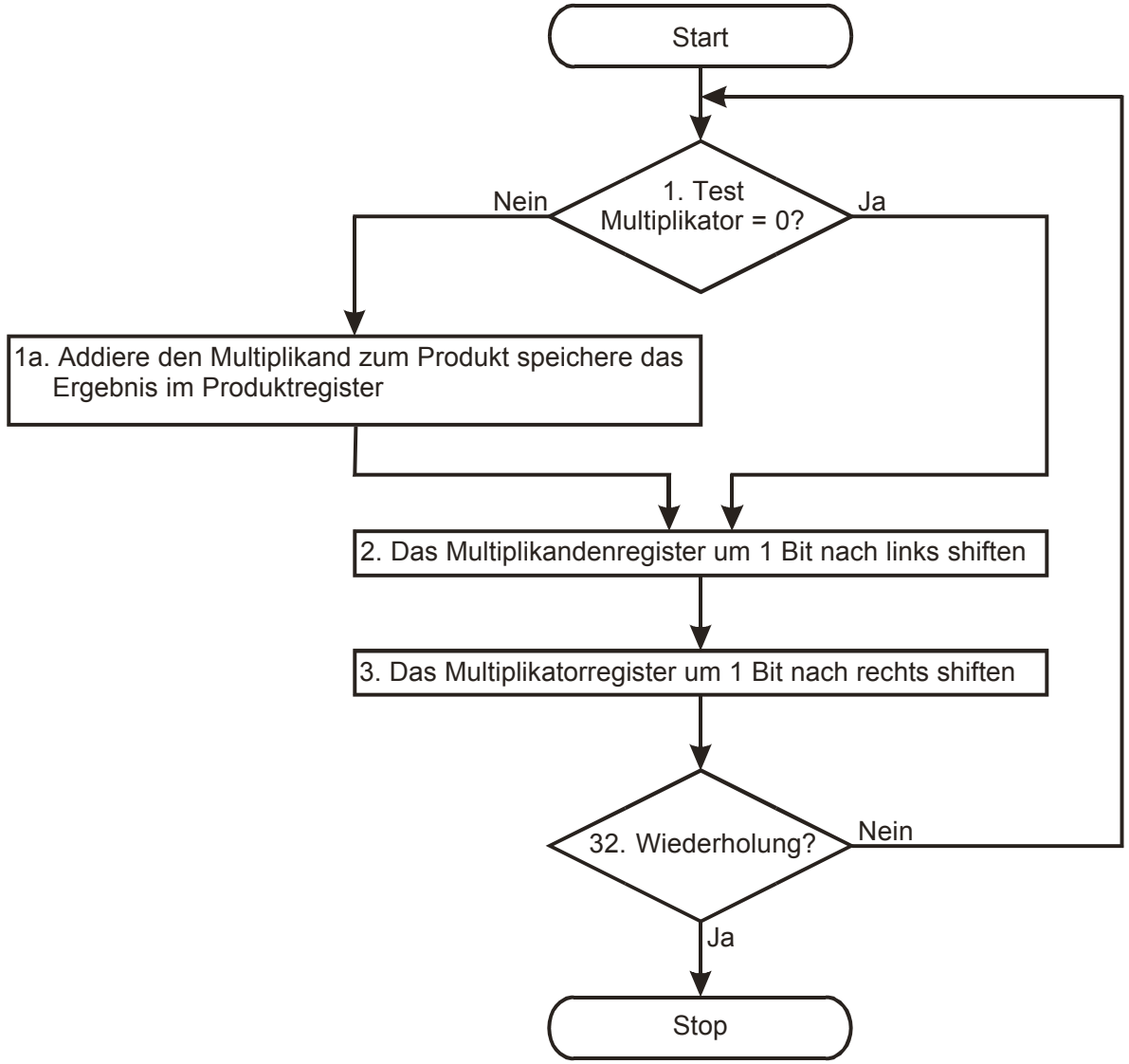
„Multiplication is vexation, Division is as bad.“

Anonymus, Elizabethan manuscript, 1570

1. Version einer Multiplikationshardware:



Dazugehöriger Algorithmus:



Computerarithmetik (15b)

Dazugehöriges Beispiel:

Schleife	Schritt	Multiplikator	Multiplikand	Produkt
0	Anfangswerte	0011	0000 0010	0000 0000
1	1a: 1 -> Prod. = Prod. + Mcand	0011	0000 0010	0000 0010
	2: Shifte Multiplikand nach links	0011	0000 0100	0000 0010
	3: Shifte Multiplikator nach rechts	0001	0000 0100	0000 0010
2	1a: 1 -> Prod. = Prod. + Mcand	0001	0000 0100	0000 0110
	2: Shifte Multiplikand nach links	0001	0000 1000	0000 0110
	3: Shifte Multiplikator nach rechts	0000	0000 1000	0000 0110
3	1: 0 -> Keine Operation nötig	0000	0000 1000	0000 0110
	2: Shifte Multiplikand nach links	0000	0001 0000	0000 0110
	3: Shifte Multiplikator nach rechts	0000	0001 0000	0000 0110
4	1: 0 -> Keine Operation nötig	0000	0001 0000	0000 0110
	2: Shifte Multiplikand nach links	0000	0010 0000	0000 0110
	3: Shifte Multiplikator nach rechts	0000	0010 0000	0000 0110