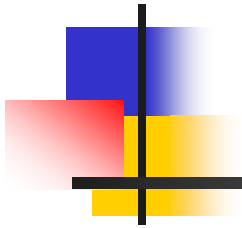


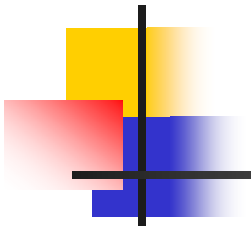
1 Zahlendarstellungen

Technische Grundlagen der Informatik 2
(Rechnertechnologie 2)
SS 2006

Andreas Koch
FG Eingebettete Systeme und ihre Anwendungen

Auf Basis von Material von

R. Hoffmann
FG Rechnerarchitektur
Technische Universität Darmstadt



1. Zahlendarstellungen

1.1 Zahl zur Basis b

1.2 Vorzeichen-Betrag-Darstellung

1.3 Zweikomplementzahl

1.4 Einskomplementzahl

1.5 Binärcodierte Dezimalzahl

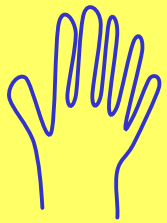
1.6 Gleitkommazahl

- IEEE-32-Bit-Gleitkommmaformat
- IEEE-64-Bit-Gleitkommmaformat

3.4 Konvertierung zwischen Zahlensystemen

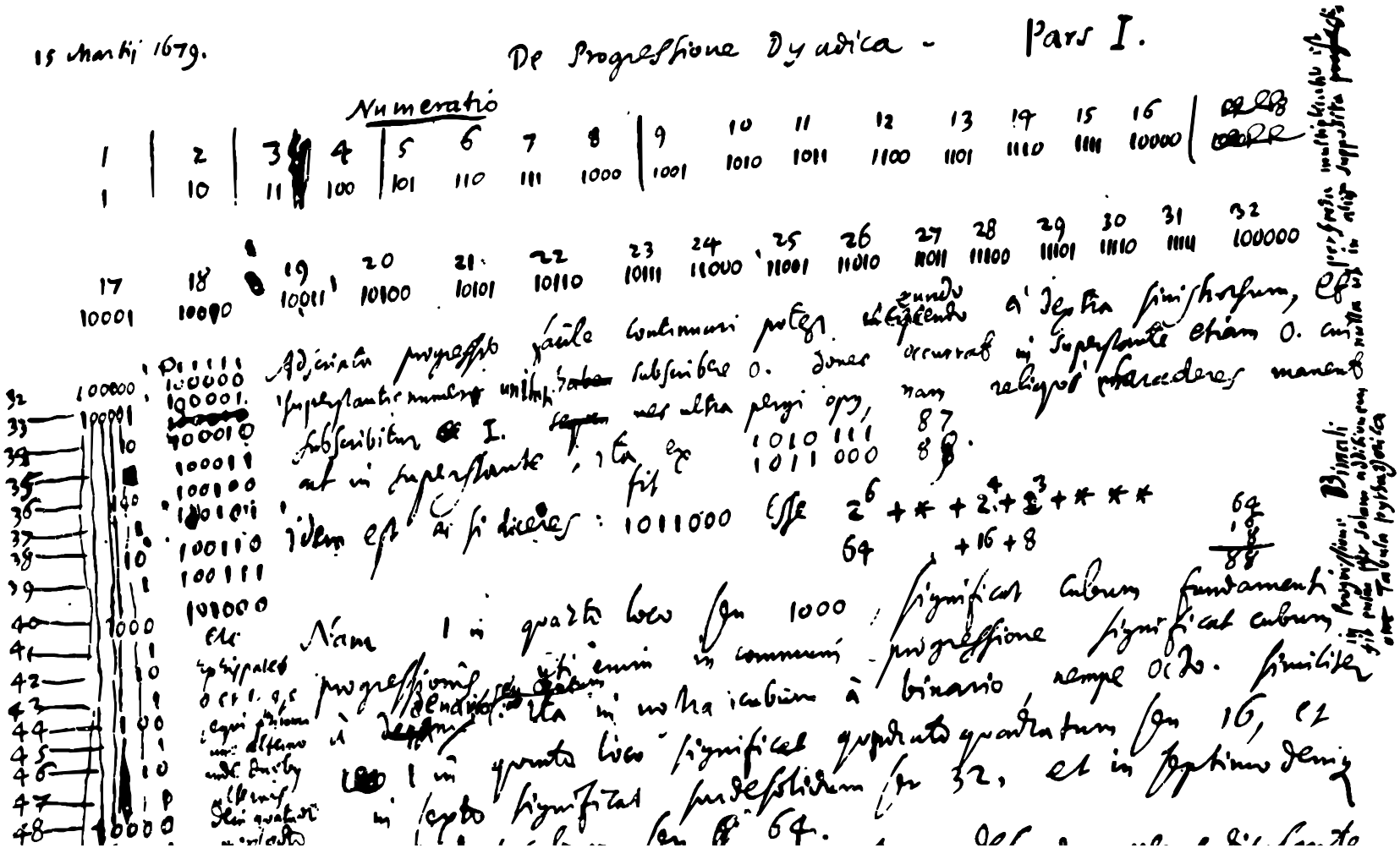
Zahlen

- [**Abstrakte**] Zahl = der **Wert** einer Zahl
- [**Konkrete**] Zahl = (**Darstellung**, **Wert**)

Darstellung				Wert
	/////	101		5
	zwei	iki (türkisch)		2

Acht Cova Zeichen des chinesischen Königs Fo-Hi vor mehr als 4000 Jahren

Leibniz 1679



Leibniz

15. März 1679

Das dyadische Zahlensystem. Teil I

Nebenstehende Folge kann leicht fortgesetzt werden, wenn man, von rechts nach links gehend, unter die Eins der oberen Zahl jeweils 0 schreibt, bis bei der oberen auch 0 vorkommt, worunter man dann 1 schreibt; weiter braucht man nicht zugehen, da die übrigen Ziffern gleich bleiben wie bei der oberen Zahl.

So wird aus: 1010111 87

1011000 88

Das ist dasselbe, wie wenn man

sagte: 1011000 ist 64

2^6 + * + 2^4 + 2^3 + * * * 16

64 + 16 + 8 8

88

1.1 Zahl zur Basis b

- b-näre Zahl, b-adische Zahl, polyadische Zahl, Stellenwertzahl, Positionssystem
- Folge von Ziffern $X_i \in \{0,1,..\ b-1\}$
- Darstellung $(X_n \ldots X_1)_b$ besitzt den Wert $SX_i * b^{i-1}$
  - wird oft als Gleichung geschrieben, obwohl es keine ist

$$(X_n \ldots X_1)_b = \sum_{i=1}^{n} X_i * b^{i-1}$$

- Erweiterung für gebrochene Zahlen

$$(X_n \ldots X_1, X_0 \ldots X_{-k})_b = \sum_{i=-k}^{n} X_i * b^{i-1}$$

- b=2 : Dualzahl
- b=10: Dezimalzahl
- b=16 : Hexadezimalzahl
- b=8 : Oktalzahl
- Spezielle Formen
  - b ist negativ
  - b ist komplexe Zahl

Dualzahl, Oktalzahl, Hexadezimalzahl

Dualzahl	Oktalzahl	Hexadezimalzahl	Wert, dargestellt als Dezimalzahl
oooo	0	0	0
oool	1	1	1
oolo	2	2	2
ooll	3	3	3
oloo	4	4	4
olol	5	5	5
ollo	6	6	6
olll	7	7	7
looo	10	8	8
lool	11	9	9
lolo	12	A	10
loll	13	B	11
lloo	14	C	12
llol	15	D	13
lllo	16	E	14
llll	17	F	15

Notation

- Mit X[n] = X_n X_{n-1} .. X_1 ist zum einen die **Darstellung** einer Zahl als Bitfolge gemeint
- Mit X[n] ist zum anderen die **n-stellige Dualzahl** gemeint, die den folgenden[dualen/binären] Wert besitzt

$$\sum_{i=1}^{n} X_i * 2^{i-1}$$

- Mit kleinem Buchstaben **x** ist der **Wert** der darzustellenden Zahl gemeint.
- Wenn die Stellenzahl n vereinbart ist, dann wird verkürzt nur X anstelle von X[n] benutzt
- Zur Darstellung der Dualziffern wird hier oft
  - o , l benutzt anstelle von
  - 0, 1

$$X[n] = X_n X_{n-1} \ldots X_1 \quad = \quad \sum_{i=1}^{n} X_i * 2^{i-1}$$

Beispiel: Darstellung und Wert

X[4] Darstellung	X dualer oder binärer Wert (als Dezimalzahl dargestellt)	z.B. Wert x einer 2K-Zahl (als Dezimalzahl dargestellt)
oooo	0	0
oool	1	1
oolo	2	2
ooll	3	3
oloo	4	4
olol	5	5
ollo	6	6
olll	7	7
looo	8	-8
lool	9	-7
lolo	10	-6
loll	11	-5
lloo	12	-4
llol	13	-3
lllo	14	-2
llll	15	-1

1.2 Vorzeichen-Betrag-Darstellung

- Vorzeichenbit V_x und Betrag |x|
  - V_x=o für positive Werte, V_x=l für negative Werte
- Eine positive Dualzahl X[n] mit Vorzeichenbit V_x nennen wir Vorzeichenzahl (V_x, X[n])
- Zwei Darstellung für die Null sind möglich
- Darstellung ist
  - gut für Multiplikation, Division
    - getrennte Behandlung von Vorzeichen und Betrag
  - schlecht für Addition und Subtraktion
    - Hardware zu kompliziert

Wertebereich

$$|x| \leq 2^n - 1$$

Abbildung

$$X[n] = |x| \quad \text{und} \quad V_x = \begin{cases} \text{o} & \text{für} \quad x \geq 0 \\ \text{l} & \text{für} \quad x \leq 0 \text{ bzw. } x < 0 \end{cases}$$

Rückabbildung

$$x = (1 - 2V_x) * X[n] \quad = \quad (\overline{V_x} - V_x) * X[n]$$

Offset Binary, Exzess-Darstellung

- Durch Addition der Basis 2^{n-1} wird der Zahlenbereich so nach oben verschoben, daß nur noch Werte = 0 entstehen.
- Definitionsbereich
  $x = -2^{n-1} \ldots (2^{n-1} - 1)$
- Abbildung $X[n] = x + 2^{n-1}$
- Rückkabbildung $x = X[n] - 2^{n-1}$
- MSB gibt Vorzeichen an
  - = o: negative Zahl
  - = l: positive Zahl
- Anwendungen
  - Digital/Analog-Umsetzer
  - zur Darstellung des Exponenten in Gleitkommazahlen, als sogen. Charakteristik

Dezimal x	*Offset Binary X[4]*
+7	1111
+6	1110
+5	1101
+4	1100
+3	1011
+2	1010
+1	1001
0	1000
-1	0111
-2	0110
-3	0101
-4	0100
-5	0011
-6	0010
-7	0001
-8	0000

1.3 Zweikomplementzahl

- auch 2K-Zahl, signed number
- Standard für Festkommarithmetik
- positive Werte werden nicht verändert
  - x=5 → X[4]= 5 = 0101
- auf negative Werte wird 2^n addiert
  - x= -5 → X[4]= -5 + 2^4 = 11 = 1011
  - an dem MSB kann man erkennen, daß der dargestellte Wert negativ ist, dieses Bit wird auch Sign genannt.
- Rückabbildung
  - falls X_n=1 dann muß 2^n abgezogen werden
  - X[4]= 1011 → x = 11 – 2^4 = -5 = 1011 – 10000 = 011 – 1000

Zweikomplementzahl

Abbildung V2K

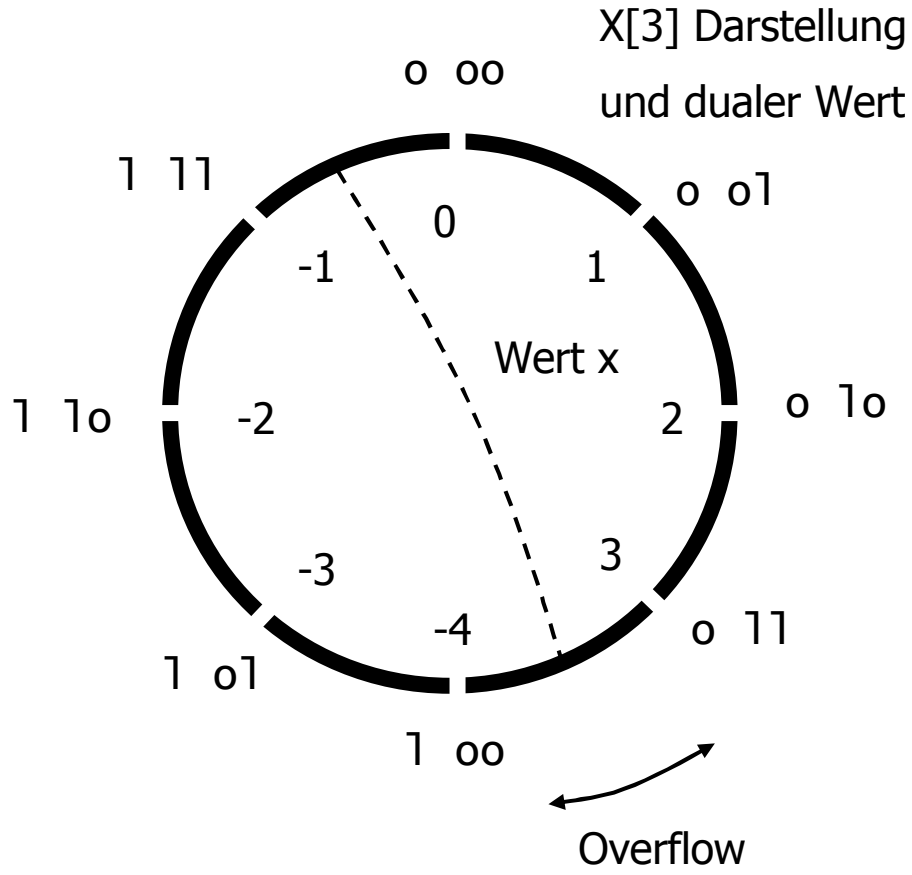
$$V2K: \quad \{-2^{n-1} : 2^{n-1}-1\} \rightarrow \{0 : 2^n - 1\}$$
$$x \mapsto X[n] .$$

$$X[n] = x \bmod 2^n = \begin{cases} x & \text{für} \quad x \geq 0 \\ 2^n + x & \text{für} \quad x < 0 \end{cases}$$

Rückabbildung V2K^{-1}

$$x = X[n] - X_n * 2^n = \begin{cases} X[n] & \text{für} \quad X_n = 0 \\ X[n] - 2^n & \text{für} \quad X_n = 1 \end{cases}$$
$$x = X[n-1] - X_n * 2^{n-1}$$

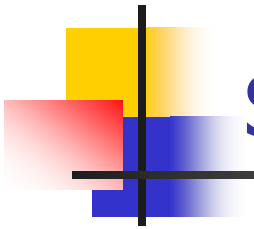
Zahlenring



- Definitionsbereich ist unsymmetrisch zur Null
  - -2^{n-1} darstellbar aber nicht $+2^{n-1}$
  - bei der Addition kann das Ergebnis größer als $2^{n-1}-1$ oder kleiner als -2^{n-1} werden → Überlauf (Overflow)
  - auch Komplementbildung von 2^{n-1} führt zu Overflow!

Wert x der 2K-Zahl X[n]

X[4] Darstellung	X dualer Wert	Wert x der 2K-Zahl
oooo	0	0
oool	1	1
oolo	2	2
ooll	3	3
oloo	4	4
olol	5	5
ollo	6	6
olll	7	7
looo	8	-8
lool	9	-7
lolo	10	-6
loll	11	-5
lloo	12	-4
llol	13	-3
lllo	14	-2
llll	15	-1



Sign-Extension

Arithmetische Erweiterung

- **Aufgabe**: Die Anzahl der Stellen einer 2K-Zahl soll von m auf n>m vergrößert werden, X[m] → X[n]
- **Lösung**: wenn das Sign X_m
  - o ist , dann werden alle weiteren Stellen links davon auf o gesetzt
  - l ist, dann werden alle weiteren Stellen links davon auf l gesetzt
  - allgemein: X[n] = X_m ...X_m X[m]
- **Beispiel**
  - X[4] = olol → X[8] = oooololol
  - X[4] = loll → X[8] = lllllolll
- 2K-Zahlen, die kleine Werte darstellen, beginnen mit oo...o oder ll...l

Arithmetischer Schift von 2K-Zahlen

- **Nach rechts**, Vorzeichenstelle nachziehen
  - Entspricht einer Division durch 2
  - Das Vorzeichen bleibt erhalten
  - Eine Stelle geht verloren (falls sie nicht als Nachkommastelle aufgehoben wird).

x=-7, X2K[4] = 1001

Nach arith-R-Shift: 1100, x= -4

x=7, X2K[4] = 0111

Nach arith-R-Shift: 0011, x=3

Beachte Rundung!

- **Nach links**
  - Entspricht Multiplikation mit 2
  - Das Ergebnis ist u.U. mit n Stellen nicht mehr darstellbar (Überlauf)

x=-2, X2K[4] = 1110

Nach arith-L-Shift: 1100, x= -4

x=3, X2K[4] = 0011

Nach arith-R-Shift: 0110, x=6

Komplementbildung

- **Problem**: Anstelle von x soll –x dargestellt werden
  - X[n] → X´[n]
- **Lösung**: **$X'[n] = 2^n - X[n]$**
- X´[n] nennt man das Komplement von X[n] gegen 2^n
- Wie kann es einfach berechnet werden?
  - Es gilt

$$X[n] + \overline{X[n]} = \sum_{i=1}^{n} (X_i + \overline{X_i}) * 2^{i-1} = 2^n - 1 = 11\ldots1$$

$$X'[n] = 2^n - X[n]$$

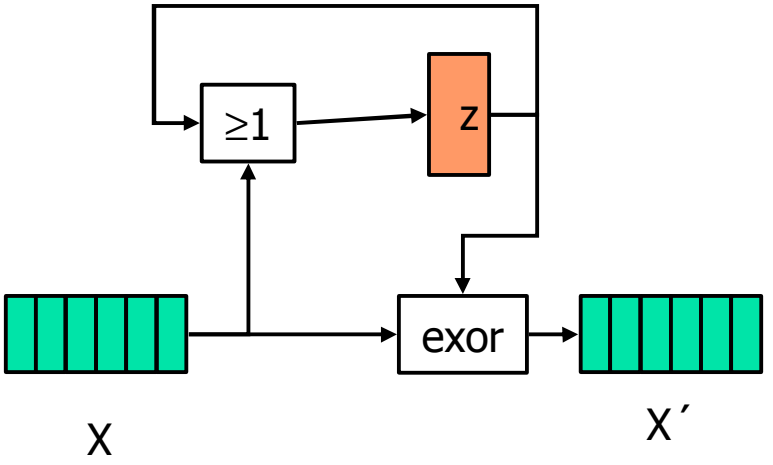
$$\boxed{X'[n] = \overline{X[n]} + 1}$$

  - einsetzen in
  - ergibt
  - Beispiel

olololoo → alle Bits negieren → lololll + l = lolloo

Serielle Komplementbildung

- Betrachte alle Bits X_i von rechts nach links, i=1 ..n
- Solange das Bit X_i =o wird es unverändert übernommen.
- Das erste Bit X_i=l wird auch noch übernommen.
- Alle weiteren Bits werden negiert.
- Beispiel
  - olo**l**oo → lol**l**oo



Schaltwerk zur seriellen Komplementbildung

z ist am Anfang o und wird mit der ersten l auf l gesetzt, wodurch die folgenden Bits über exor negiert werden

1.4 Einskomplementzahl

- Auch 1K-Zahl, inzwischen kaum noch benutzt.
- Unterschied zur 2K-Zahl: Es wird das Komplement gegen 2^n-1 gebildet.
- positive Werte werden nicht verändert
  - x=5 → X[4]= 5 = `olol`
- auf negative Werte wird 2^n -1 addiert
  - x= -5 → X[4]= -5 + 2^4 -1 = 10 = `lolo`
- Zahlenbereich ist symmetrisch zur Null.
- für die Null gibt es zwei Darstellungen
  - `oo...o`
  - `ll...l` (falls nicht verboten)

Einskomplementzahl

Abbildung

$$V1K: \quad \{-2^{n-1}+1 : 2^{n-1}-1\} \rightarrow \{0 : 2^n-2\}$$
$$x \mapsto X[n]\,.$$

Abbildungsvorschrift

$$X[n] = x \bmod (2^n-1) = \begin{cases} x & \text{für } x \geq 0 \\ 2^n-1+x & \text{für } x < 0 \end{cases}$$

Rückabbildungsvorschrift

$$x = X[n] - X_n * (2^n-1) = \begin{cases} X[n] & \text{f. } X_n = \texttt{0} \\ X[n]-2^n+1 & \text{f. } X_n = \texttt{1} \end{cases}$$

Komplementbildung

- Besonders einfach, alle Bits werden negiert:

$$X[n] + X'[n] = 2^n - 1 = \texttt{11...1} = X[n] + \overline{X[n]}$$
$$X'[n] = \overline{X[n]}\,.$$

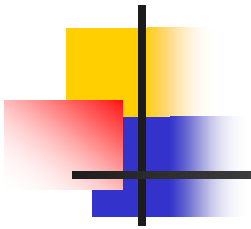
- Warum wird die 1K-Darstellung nicht benutzt?
  - Bei der Addition muß im Falle eines Übertrages eine 1 addiert werden (Hardware- und Zeitaufwand)
  - Für die Addition von Dualzahlen und 2K-Zahlen kann derselbe Addierer benutzt werden.

Vergleich

Wert x dezimal	Vorzeichen zahl $V_x X[2]$	2K-Zahl $X[3]$	1K-Zahl $X[3]$
-4	----	1 00	----
-3	1 11	1 01	1 00
-2	1 10	1 10	1 01
-1	1 01	1 11	1 10
0	0 00 (1 00)	0 00	0 00 (1 11)
1	0 01	0 01	0 01
2	0 10	0 10	0 10
3	0 11	0 11	0 11

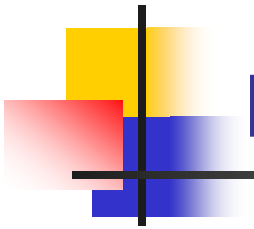
Festkommazahlen in Programmiersprachen

Typ Delphi	C	JAVA	Bereich	Bits	
Byte	unsigned char		0..255	8	un-signed
Word	unsigned int		0..65535	16	
Longword	unsigned long int		0..4294967295	32	
Shortint	char	byte	-128..127	8	signed 2K
Smallint	int	short	-32768..32767	16	
Longint	long	int	-2147483648..2147483647	32	
Int64	long long int	long	-2^63..2^63-1	64	
Currency			-922337203685477.5808.. 922337203685477.5807	64	



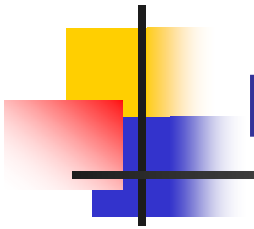
1.5 Binärcodierte Dezimalzahl

- Eine Codierung von Dezimalzahlen im Rechner
- Gründe zur Verwendung
  - Keine Konvertierung für die externe Darstellung erforderlich
    - z.B. in Taschenrechnern
  - Vermeidung von Approximationsfehlern
    - 0,1 = `o,o ooll ooll ooll ....` (nicht endlich!)
    - In der kommerziellen Datenverarbeitung
- Jede BCD-Ziffer wird durch 4 Bits dual codiert.
- In ungepackter Form werden 8 Bits zur Codierung benutzt. Dabei werden die höchstwertigen Bits auf oooo oder auf oolo (ASCII) gesetzt, oder der EBCDIC wird verwendet.



Beispiele

- Gepackte BCD-Zahlen
  - 59_{10}= olol lool $_{BCD}$
  - 81_{10}= looo ool $_{BCD}$
- Ungepackte BCD-Zahlen
  - 59_{10}= oooolol ooolool $_{BCD}$
  - 81_{10}= oooolooo oooooool $_{BCD}$



Negative BCD-Zahlen

- Möglichkeiten
  - Vorzeichen-Betrag-Darstellung
  - 9-Komplement
    - Komplement gegen 10^m-1
    - bei m Dezimalstellen
  - 10-Komplement
    - Komplement gegen 10^m

9-Komplement

- m = Anzahl der Stellen
- |x| < 499…9
- Vorzeichenstelle (4 Bits)
  - 0 bis 4: positive Zahlen
  - 5 bis 9: negative Zahlen
- Komplementbildung
  - (X_{9K},x)→(X'_{9K},-x)
  - Für jede Ziffer wird Differenz gegen 9 gebildet
    - durch Addition von 6 = ollo und anschließender Negation aller Bits

$$X_{9K} = \begin{cases} x & \text{für } x \geq 0 \\ (10^m - 1) + x & \text{für } x < 0 \end{cases}$$

$$X'_{9K} = (10^m - 1) - X_{9K} = 99\ldots9 - X_{9K}$$

$$\begin{aligned} X_{9K}\langle i \rangle &= 15 - (6 + X_{9K}\langle i \rangle) \\ &= \mathtt{1111} - (\mathtt{ollo} + X_{9K}\langle i \rangle) = \overline{\mathtt{ollo} + X_{9K}\langle i \rangle} \end{aligned}$$

10-Komplement

- m = Anzahl der Stellen
- -500…0 ≤ x ≤ 499…9
- Vorzeichenstelle (4 Bits)
  - 0 bis 4: positive Zahlen
  - 5 bis 9: negative Zahlen
- Komplementbildung
  - ($X1_{10K}$,x)→(X'_{10K},-x)
  - Komplement von X_{10K}=500…0 ist verboten (Überlauf)
  - 1.) Negiere alle Bits und addiere 1, merke Überträge zwischen den Tetraden
  - 2.) Addiere lolo (-6) auf alle Tetraden, die keinen Übertrag erzeugt haben

$$X_{10K} = \begin{cases} x & \text{für } x \geq 0 \\ 10^m + x & \text{für } x < 0 \end{cases}$$

$$\begin{aligned} X'_{10K} &= 10^m - X_{10K} = (10^m - 1) - X_{10K} + 1 \\ &= (99\ldots9 - X_{10K}) + 1\,. \end{aligned}$$

$$\begin{aligned} X_{10K} &= 509400 \qquad x = -490600 \\ X'_{10K} &= 490600 \qquad x = +490600 \end{aligned}$$

Beispiele 9er und 10er Komplement

Number	10's complement	9s' compleme
1849	8151	8150
2067	7933	7932
100	9900	9899
7	9993	9992
8151	1849	1848
0	10000 (= 0)	9999

- Beachte: 9er Komplement hat zwei Nullen
  - +0 und -0 = 9999

BCD-Darstellung in der IntelArchitecture-32

- 80 Bits
- Vorzeichen S und 18 Dezimalstellen (Magnitude)
- Kompatibel zum COBOL-Standard

1.6 Gleitkommazahl, Motivation

- Festkommazahlen können nur einen relativ kleinen Zahlenbereich überstreichen
- n Bit Wort
  - positive Dualzahl: 0 .. 2^n-1
  - Darstellung größerer Zahlen durch Verschiebung des fiktiven Kommas nach rechts um m Stellen: Zahlenbereich wird mit 2^m multipliziert
    - Abstand zwischen den Zahlen ist 2^m
  - Verschiebung des Kommas um m Stellen nach links: Zahlenbereich wird durch 2^m dividiert
    - Abstand zwischen den Zahlen ist 2^{-m}
- Problem
  - Der Programmierer muß sich für die Anwendung den Zahlenbereich genau überlegen und das fiktive Komma programmtechnisch mitführen (die Festkommazahl wird auf 2^m normiert.) Ein Überlauf wird durch geeignete Normierung vermieden
- Lösung
  - Codierung der Kommastellung und automatische Normierung
  - Zusätzliche Bits für die Kommastellung
  - Gleitendes Komma: floating point

Eigenschaften der Gleitkommazahl

- **Vorteile**
  - größerer Wertebereich darstellbar
    - seltener: Überlauf (Overflow), Unterlauf (Underflow)
    - Programmierung vereinfacht (keine vorausgeplante Normierung)
  - Relative Darstellungsgenauigkeit in etwa konstant
- **Allgemein**
  - Anzahl der GK-Zahlen ist endlich
  - Es gibt keine
    - beliebig großen/kleine Zahlen
    - beliebig dicht benachbarte GKZ
    - Der Abstand zwischen benachbarten GKZ ist nicht konstant.

- **Nachteile**
  - Hardware-Implementierung deutlich komplexer als für Festkommazahlen
  - Rundungsfehler entstehen, insbesondere bei Addition/Subtraktion von Zahlen deren Kommastellungen sich stark voneinander unterscheiden. Genauigkeit wird vorgetäuscht.
  - Je nach der Auswertungsart von Formeln ergeben sich unterschiedlich große Fehler
    $(a-b) + (c-d) \; ^{1} \; (a+c) - (b+d)$

Normalisierung

- Wert x
- Mantisse mx
- Basis b
- Exponent ex
- Vorzeichen v_x =+1/-1
- k wird festgelegt (meist 0 oder 1)

$$x = mx * b^{ex}$$

$$x = v_x * \underbrace{mx_{norm} * b^k}_{\widehat{mx}_{norm}} * b^{ex}$$

- **Allgemein**: Mantisse wird normalisiert, um die Darstellung eindeutig zu machen und um immer mit der höchsten Genauigkeit zu rechnen,
  - mx_{norm} hat nur r Stellen hinter dem Komma, die erste Stelle hinter dem Komma ist >0

$$\frac{1}{b} \leq mx_{norm} \leq 1 - \frac{1}{b^r} < 1$$

- Speziell beim IEEE 754 - Standard wurde b=2 und k=1 gewählt, d.h. die normalisierte Mantisse (mit $\grave{U}$) liegt zwischen 1 und 2-ε

$$x = v_x * \widehat{mx}_{norm} * b^{ex}$$

$$1 = \widehat{mx}_{norm} = 1,11 \ldots 1$$

IEEE is an abbreviation for Institute of Electrical and Electronic Engineers

Beispiel Normalisierung bei b=10

- Darstellung von 300
  - mx=3, ex=2
  - mx=30, ex=1
  - mx=300, ex=0
- Jetzt mit k=1,r=1 normieren
  - Normierte Mantisse
    - mxnorm Minimal = 0,1
    - mxnorm Maximal = 0,9
    - mx^norm Minimal=1
    - mx^norm Maximal=9
  - Eindeutige Darstellung von 300
    - mxnorm=0,3 ex=2

$$x = v_x * \underbrace{mx_{norm} * b^k}_{\widehat{mx}_{norm}} * b^{ex}$$

$$\frac{1}{b} \leq mx_{norm} \leq 1 - \frac{1}{b^r} < 1$$

Relativer Fehler

- Abstand zwischen zwei Werten (r=Stellen der Mantisse hinter dem Komma) ergibt sich aus dem LSB der Mantisse und dem Exponenten

$$\Delta x = \frac{1}{b^r} * b^{exp}$$

- Relativer Fehler (LSB unsicher)
  - höchstens
    - b^{-r}/b^{k-1}
  - mindestens
    - $b^{-r}/(b^k - b^{-r})$

IEEE, k=1

1,oo ... ox → $2^{-r}/1$

1,11 ... 1x → $2^{-r}/(2-2^{-r}) \approx 2^{-r}/2$

Wie ungenau ist die Darstellung von 8 Millionen € wenn r=23 Nachkommastellen benutzt werden?

Maximaler Relativer Fehler: 2^{-23} → etwa 1 € ungenau

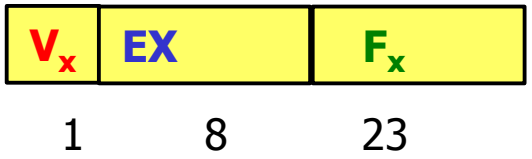
Darstellung des Exponenten

- Charakteristik (biased exponent)
- Auf den Exponenten (true exponent) wird eine Basis (bias) addiert, so daß nur noch positive Werte entstehen.
- 1. Möglichkeit:
  Addition von 2^{n-1}
  - **$EX[n] = ex + 2^{n-1}$** ,
  - verwendet bei DEC-Rechnern, IBM/Siemens-Großrechner mit b=16
- 2. Möglichkeit:
  Addition von $2^{n-1} -1$
  - **$EX[n] = ex + 2^{n-1} -1$**
  - IEEE-Standard

- Durch Verwendung der Charakteristik-Darstellung und der Anordnung (Exponent steht vor der Mantisse) kann der Größenvergleich zweier GK-Zahlen relativ leicht durchgeführt werden:
  - 1. Vorzeichen prüfen
  - 2. Dualzahlen-Vergleich der Charakteristiken
  - 3. Dualzahlen-Vergleich der Mantissen

IEEE-32-Bit-Gleitkommmaformat

- **Vorzeichen V_x** (Sign)
- **Charakteristik EX = ex+127**
  - für ex = -126 ... +127
- Die **Mantisse** MX = 1,**FX** wird auf Werte zwischen 1 und 2 normalisiert
  - 1 =MX = 2 - 2^{-23}
  - Dargestellt werden nur die 23 Binärstellen hinter dem Komma (fraction FX), da die Stelle vor dem Komma immer gleich 1 ist.
- **Wert x**
- Betragsmäßig größter und kleinster Wert (normalisiert):



$$V_x = \begin{cases} 1 & \text{für negative Zahlen} \\ 0 & \text{für positive Zahlen} \end{cases}$$

$$x = (-1)^{V_x} * 2^{EX-127} * (1, FX)$$

$$= \begin{cases} +2^{EX-127} * (1, FX) & \text{für } V_x = 0 \\ -2^{EX-127} * (1, FX) & \text{für } V_x = 1 \end{cases}$$

$$x_{max} = 2^{127} * (2 - 2^{-23}) \approx 1,7 * 10^{38}$$

$$x_{min} = 2^{-126} * (1,0) \approx 1,17 * 10^{-38}$$

Gültige und ungültige Exponenten

ex	EX	Interpretation
128	255 = 1111 1111	kein gültiger Exponent; zur Darstellung von ∞ und Kontrollcodes
127	254 = 1111 111o	gültige Exponenten für normalisierte Gleitkommazahlen
126	253 = 1111 11o1	
⋮		
0	127 = o111 1111	
−1	126 = o111 111o	
⋮		
−126	1 = oooo ooo1	
−127	0 = oooo oooo	kein gültiger Exponent; zur Darstellung von Null bzw. Kleiner Zahl

Beispiele

$x = 3{,}5 = 1{,}75 * 2^{1} = 1{,}75 * 2^{128-127}$

$= 1{,}110\ 0000\ 0000\ 0000\ 0000\ 0000 * 2^{128-127}$

$= (0, 128, 0{,}75)$

$= (0, 1000\ 0000, 110\ 0000\ 0000\ 0000\ 0000\ 0000)$

$x = -11{,}375 = -1011{,}011 * 2^{0} = -1{,}011011 * 2^{3}$

$= (1, 130, 0{,}421875)$

$= (1, 1000\ 0010, 011\ 0110\ 0000\ 0000\ 0000\ 0000)$

Sonderfälle (1)

a) Die Null

Darstellung der positiven Null: $(\circ, 0, 0)$
Darstellung der negativen Null: $(1, 0, 0)$

b) Kleine Zahl (denormalized number)

Darstellung: $(V_\infty, 0, \neq 0)$ Betrag: $0 < |x| < 2^{-126}$
Kleine Zahlen repräsentieren sehr kleine Werte, die kleiner als $x_{\min}$ und größer als Null sind. Der Wert ergibt sich zu
$x = (-1)^{V_\infty} * 2^{EX-126} * (0,FX)$.

c) Große Zahl, ∞

Darstellung: $(V_\infty, 255, 0)$ Betrag: $|x| > x_{\max}$
∞ kann sowohl ein positives als auch ein negatives Vorzeichen besitzen.

d) Ungültige Zahl (Not a Number, *NAN*)

Darstellung: $(V_\infty, 255, \neq 0)$
Ungültige Zahlen werden als Kontrollcodes interpretiert. Entweder entstehen bestimmte Kontrollcodes bei der Ausführung unzulässiger Operationen (z. B. $0*\infty$) oder sie werden dazu benutzt, Statuswerte durch eine Serie von Operationen zu schleusen.

Sonderfälle (2)

e) Überlauf (Overflow)

Überlauf kann bei der Addition, Multiplikation und Division auftreten, wenn nach dem Runden ein Ergebnis $|x| > x_{\max}$ entsteht.

f) Unterlauf (Underflow)

Unterlauf kann bei der Addition, Multiplikation, Division und Reziprokwertbildung $(1/x)$ auftreten, wenn nach dem Runden ein Ergebnis $0 < |x| < x_{\min}$ entsteht. Als Ergebnis wird entweder eine „Kleine Zahl" erzeugt oder $\pm\, 0$.

g) Besondere Operationen

Besondere Operationen können für die Verarbeitung von ∞, kleinen Zahlen und durch die Unterscheidung von positiver und negativer Null definiert werden.

h) Rundung bei einem nicht darstellbaren Rest R

Vier Rundungsarten stehen zur Auswahl:

- Rundung zur nächsten darstellbaren Zahl:

  zur nächsten

$$
\begin{array}{lcll}
|x|, R & \rightarrow & |x| + 1 & \text{für } R \geq 0,5 \quad (MSB \text{ von } R \text{ ist } 1) \\
 & \rightarrow & |x| & \text{für } R < 0,5 \quad (MSB \text{ von } R \text{ ist } 0)
\end{array}
$$

- Rundung in Richtung $+\infty$:

$$
\begin{array}{lcll}
+|x|, R & \rightarrow & +|x| + 1 & \text{für } R \neq 0 \\
-|x|, R & \rightarrow & -|x| &
\end{array}
$$

  Nach oben

- Rundung in Richtung $-\infty$:

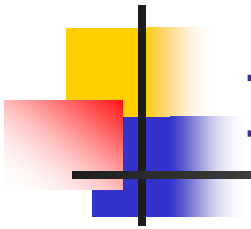
  Nach unten

$$
\begin{array}{lcll}
+|x|, R & \rightarrow & +|x| & \\
-|x|, R & \rightarrow & -|x| - 1 & \text{für } R \neq 0
\end{array}
$$

- Rundung in Richtung 0

$$|x|, R \rightarrow |x|$$

  Nach Null



IEEE-64-Bit-Gleitkommmaformat

- Das IEEE-64-Bit-Gleitkommaformat (long real) entspricht dem IEEE-32-Bit-Gleitkommaformat (short real) in seiner Aufbaustruktur.
- Für das Vorzeichenbit wird 1 Bit, für die Charakteristik werden 11 Bits (bias = 1023) und für Fraction werden 52 Bits benutzt.
- Für noch höhere Ansprüche an die Rechengenauigkeit kann das IEEE-Gleitkommaformat auf (1, = 15, = 63) Bits erweitert werden.

Gleitkommazahlen in Anwendungen

	Bereich +/-	Dezimalstellen-Genauigkeit	Intel-Architecture 32	C	JAVA	Delphi
32-Bit IEEE Short Real	1.2×10^{-38} .. 1.7×10^{38}	7-8	Single Real	float	float	Single
64-Bit IEEE Long Real	2.2×10^{-324} .. 2.0×10^{308}	15-16	Double Real	double	double	Double
80-Bit IEEE			Extended Real			
96-Bit IEEE				long double		

3.4 Konvertierung zwischen Zahlensystemen

- **Gegeben**: m-stellige ganze Zahl A zur Basis a
- **Gesucht**: n-stellige ganze Zahl zur Basis b
- **Lösungsansatz**: stelle A oder B als Potenzreihe oder nach dem Hornerschema dar, setze A=B.

$$A = (A_m A_{m-1} \dots A_1)_a \quad \rightarrow \quad (B_n B_{n-1} \dots B_1)_b = B$$

$$\begin{aligned} A &= A_m * a^{m-1} + \dots + A_2 * a + A_1 \\ B &= B_n * b^{n-1} + \dots + B_2 * b + B_1 \\ A &= (\dots (A_m * a + A_{m-1}) * a + \dots + A_2) * a + A_1 \\ B &= (\dots (B_n * b + B_{n-1}) * b + \dots + B_2) * b + B_1 \end{aligned}$$

Summationsmethode

- $A_a \rightarrow B_b$
- Erstelle eine Tabelle, in der für jede Ziffer A_i und die Stellenwertigkeit a^{i-1} der äquivalente Summand $A_i * a^{i-1}$ (codiert im Zielsystem b) enthalten ist.
- Lies für Ziffer A_i (Zeile der Tabelle) mit der Stellenwertigkeit a^{i-1} (Spalte) den äquivalenten Summanden ☺.
- Addiere alle Summanden im Zielsystem.

$$\sum_{i=1}^{m} A_i * a^{i-1} = B$$

	Äquivalenter Summand, für Stellenwertigkeit a^{i-1} ,Basis b				
Ziffer A_i Basis a	i=0	1	2	...	m-1
0					
1			☺		
...					
a-1					

Hexadezimalzahl → Dezimalzahl

Hexadezimal-ziffer	Stellenwertigkeit					
	$*16^0$	$*16^1$	$*16^2$	$*16^3$	$*16^4$	$*16^5$
0	00	000	0000	00000	000000	00000000
1	01	016	0256	04096	065536	01048576
2	02	032	0512	08192	131072	02097152
3	03	048	0768	12288	196608	03145728
4	04	064	1024	16384	262144	04194304
5	05	080	1280	20480	327680	05242880
6	06	096	1536	24576	393216	06291456
7	07	112	1792	28672	458752	07340032
8	08	128	2048	32768	524288	08388608
9	09	144	2304	36864	589824	09437184
A	10	160	2560	40960	655360	10485760
B	11	176	2816	45056	720896	11534336
C	12	192	3072	49152	786432	12582912
D	13	208	3328	53248	851968	13631488
E	14	224	3584	57344	917504	14680064
F	15	240	3840	61440	983040	15728640
	Äquivalenter dezimaler Summand					

Beispiel

$$
\begin{array}{lrr}
 & F18C_{16} & \\
= & F000 & 61440 \\
+ & 100 & 256 \\
+ & 80 & 128 \\
+ & C & 12 \\
\hline
 & & 61836_{10}
\end{array}
$$

Dezimalzahl → Hexadezimalzahl

Dezimal-ziffer	Stellenwertigkeit						
	$*10^0$	$*10^1$	$*10^2$	$*10^3$	$*10^4$	$*10^5$	$*10^6$
0	0	00	000	0000	00000	00000	000000
1	1	0A	064	03E8	02710	186A0	0F4240
2	2	14	0C8	07D0	04E20	30D40	1E8480
3	3	1E	12C	0BB8	07530	493E0	2DC6C0
4	4	28	190	0FA0	09C40	61A80	3D0900
5	5	32	1F4	1388	0C350	7A120	4C4B40
6	6	3C	258	1770	0EA60	927C0	5B8D80
7	7	46	2BC	1B58	11170	AAE60	6ACFC0
8	8	5D	320	1F40	13880	C3500	7A1200
9	9	5A	384	2328	15F90	DBBA0	895440
	Äquivalenter hexadazimaler Summand						

Beispiel

$$
\begin{array}{rrr}
 & 61836_{10} & \\
= & 60000 & \text{EA60} \\
+ & 1000 & \text{3E8} \\
+ & 800 & \text{320} \\
+ & 30 & \text{1E} \\
+ & 6 & \text{6} \\
\hline
 & \text{F18C}_{16} & = \texttt{1111 0001 1000 1100}
\end{array}
$$

Divisionmethode

- $A_a \rightarrow B_b$
- Bei der sukzessiven ganzzahligen Division durch b stellen die Reste die gesuchten Ziffern B_1, B_2, .. B_n dar.
- Division und Restbildung erfolgt im Quellsystem
- Ziffernumcodierung ist erforderlich wenn b>a ist
- Für den Menschen geeignet; im Rechner sollte die Division vermieden werden.

$$A = (\cdots(B_n b + B_{n-1})b + \cdots + B_2)b + B_1$$

Beispiel: Dezimalzahl → Dualzahl

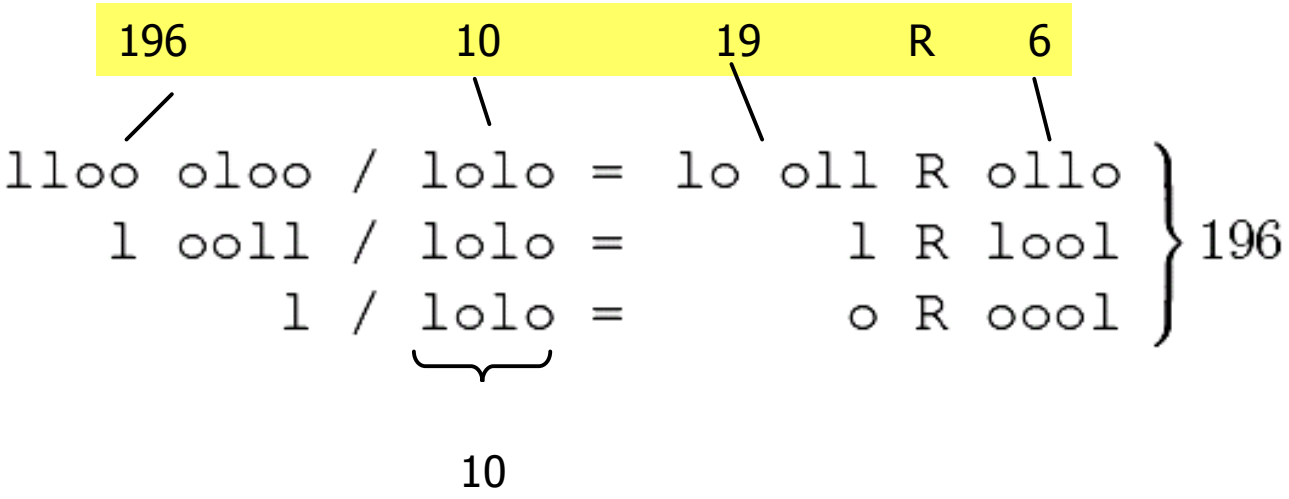
$$\begin{array}{rcrcl} 196/2 & = & 98 & R & 0 \\ 98/2 & = & 49 & R & 0 \\ 49/2 & = & 24 & R & 1 \\ 24/2 & = & 12 & R & 0 \\ 12/2 & = & 6 & R & 0 \\ 6/2 & = & 3 & R & 0 \\ 3/2 & = & 1 & R & 1 \\ 1/2 & = & 0 & R & 1 \end{array}$$

LSB

lloo oloo

Divisionsmethode, Dualzahl → Dezimalzahl

- Dieses Beispiel demonstriert
  - den Aufwand im Rechner wegen der Division von Dualzahlen
  - die Ziffernumcodierung, weil b>a
  - → Wahl einer Methode mit weniger Aufwand



Divisionsmethode für gebrochene Zahlen

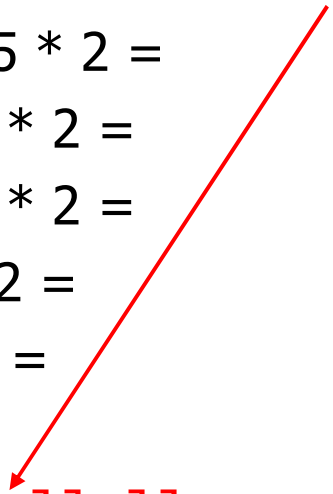
- **Gegeben**: Gebrochene Dezimalzahl $0,A_{-1}\ A_{-2}\ \ldots\ A_{-m}$
- **Gesucht**: Gebrochene Dualzahl $B_b = o,B_{-1}\ B_{-2}\ \ldots\ B_{-n}$
- **Ansatz**: $o,B_{-1}\ B_{-2}\ \ldots\ B_{-n} = A$

- for i:=1 to n do (im Quellsystem)
  - A:=A ∗ b
  - B_{-i}:= Ganzzahliger Teil von A
  - A:=A - B_{-i}

- Die Methode kann zu den Divisionsmethoden gerechnet werden, weil eine Division durch 1/b im Quellsystem durchgeführt wird, die einer Multiplikation mit b entspricht.
- Für gebrochene Zahlen ist die Divisionsmethode im Rechner geeignet.

Beispiel: Dezimal → Dual

- **Beispiel**
- 0,421875 *2 = 0,84375
- 0,84375 * 2 = 1,6875
- 0,6875 * 2 = 1,3750
- 0,3750 * 2 = 0,75
- 0,75 * 2 = 1,5
- 0,5 * 2 = 1,0
- 0 * 2 = 0,0
- B = o,ollollo



- Zum Nachrechnen
  - 0,1 = o,o ooll ooll ooll ... nicht endlich!
- Multiplikation mit 2 kann durch BCD-Addition (x+x=2x) erfolgen.

Beispiel: Dual → Dezimal

Beispiel

o,ollollo * lolo =

oo,llollo um 1 Stelle verschoben

ooll,ollo um 3 Stellen verschoben

oloo,oolllo * lolo

oo,olllo

oool,llo

oolo,oollo * lolo

oo,ollo

oool,lo

oool,lllo

oool,llo

olll,oo

looo,llo * lolo

ol,lo

ollo,

olll,lo * lolo

ol,o

oloo,

olol,o

B = 0,421875

Die Multiplikation mit 10=lolo wird auf die stellenversetzte Addition zurückgeführt

Es ergeben sich die Dezimalziffern im BCD-Code

Multiplikationsmethode

- $A_a \rightarrow B_b$
- 1. Die Ziffern A_i werden umcodiert in die Basis b (Rechnung im Zielsystem)
- 2. for i:=m downto 1 do
  - B:=B * a + A_i

geeignet zur Konvertierung ganzer Zahlen im Rechner

$$(\cdots(A_m * a + A_{m-1}) * a + \cdots + A_2) * a + A_1 \quad = \quad B$$

Beispiel: Dezimalzahl 196 → Dualzahl

$$(\overbrace{\texttt{oool}}^{1} * \texttt{lolo} + \overbrace{\texttt{lool}}^{9}) * \texttt{lolo} + \overbrace{\texttt{ollo}}^{6} = \texttt{lloo oloo}$$

Beispiel: Dualzahl `lloo oloo` → Dezimalzahl

```
      l     l      o      o      o      l      o      o
((((((1*2 + 1)*2 + 0)*2 + 0)*2 + 0)*2 + 1)*2 + 0)*2 + 0 = 196
```