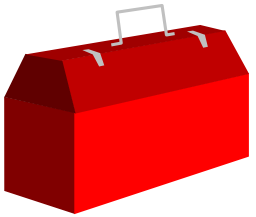
Informatik

Datentypen und Datenstrukturen

Prof. Dr.-Ing. Thomas Wiedemann
Fachgebiet Informatik / Mathematik

Überblick zur 3. Vorlesung

- Begriff der Information
  - Bit und Bytes als Informationseinheiten
  - Rechnen mit Binärzahlen
  - Häufig verwendete Größenangaben (zum Auswendiglernen)
  - Typische Abbildungen auf technische Medien
- Allgemeine Vorgehensweise und Probleme bei der Abbildung realer Daten und Größen auf Computerdaten
  - Die endliche Anzahl verfügbarer Bits und Bytes
  - Regeln zur Auswahl von konkreten Datentypen
- Grundlegende Datentypen
- Überblick und Ausblick auf weitere Datentypen
  - Darstellung von Texten (ausführlicher im 2. Teil des Kurses)
  - Datenstrukturen als Kombination mehrerer primitiver Typen
  - Objektorientierte Datenmodellierung



Begriff der Information

Definition der Information

- **Information** ist Gewinn an Wissen
- Information ist beseitigte Ungewißheit
- Information = Nachricht; Auskunft; Belehrung, Aufklärung.
- Information = räumliche oder zeitliche Folge physikalischer Signale, die mit bestimmten Wahrscheinlichkeiten oder Häufigkeiten auftreten.
- Information = sich zusammensetzende Mitteilung, die beim Empfänger ein bestimmtes Verhalten bewirkt .
- Informationsträger im weiteren Sinn sind Dinge, die nur über sich selber Auskunft geben. Dazu zählen beispielsweise Steine.
- **Informationsträger im engeren Sinne** sind Dinge, die von Lebewesen zur Übertragung oder Speicherung von Information genutzt werden können. So kann man beispielsweise mehrere kleine Steine zum Abzählen benutzen.

Bit und Bytes

Das BIT ist die kleinste Informationseinheit !

- Entspricht dem Informationsgehalt einer JA/NEIN-Antwort !
- Läßt sich technisch mit zwei unterscheidbaren Zuständen abbilden:
- Schalter: EIN/AUS Relais: ZU / OFFEN
- Spannung: HOCH/NIEDRIG Stromfluß: AN/AUS Ladung: HOCH/NIEDRIG
- CD: Langloch/kurzes Loch (Pits) Magnetband: Südpol /Nordpol

Merke: Zwei Zustände lassen sich technisch am effektivsten und sichersten speichern, manipulieren, verarbeiten und übertragen.

Arbeit mit binären Zuständen

Die zwei Zustände werden auf ein logisches Bit abgebildet :

- die logische 1 entspricht meist einem aktiven Zustand,
- die logische 0 entspricht meist dem nicht aktivem Zustand.
- Alle weiteren Darstellungen abstrahieren von der konkreten technischen Lösung und arbeiten nur noch mit der 1 oder 0 !
- Alle Berechnungen und Operationen in heutigen Standardcomputern werden auf Operationen über einzelne Bits zurückgeführt **!**

Abbildung von Daten mit mehr als einem Zustand:

- Zusammenfassung von mehreren Bits zu größeren Datenstrukturen
- der mögliche Wertebereich ergibt aus der Potenz der Anzahl der verwendeten Bits zur Basis 2 (=mögliche Anzahl von Kombinationen)
- derartige Zahlen werden als Binär- oder Dualzahlen bezeichnet

Abbildung von Daten auf Bitstrukturen

Weitere, wichtige Definitionen und Regeln

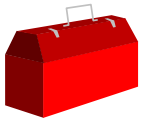
- Das zweitwertige BIT (JA/NEIN) ist die kleinste Informationseinheit !
- Alle größeren Daten werden durch die Zusammenfassung von mehreren Bits dargestellt.
- Die endlichen Ressourcen aktueller Computer begrenzen in jedem Fall auch die Anzahl der Bits.
- Die Umwandlung zwischen textueller Darstellung (für Menschen verständlich) und das jeweilige Bitformat wird in der Regel durch den Rechner selbst durchgeführt !

Abbildung realer Zahlen und Größen auf Computerdaten

Bestimmung des Zahlentyps aus mathematischer Sicht

- Natürliche, ganze, gebrochene, rationale, irrationale, komplexe Zahlen
- Bestimmung des Wertebereichs (minimaler / maximaler Wert : sowohl für Mantisse wie Exponent !! Warum ist auch der Bereich um 0 kritisch ?)

Allgemeine Regeln:



- Es ist insbesondere der Worst Case, also der schlechteste bzw. ungünstigste Fall zur berücksichtigen. Bei Nichtbefolgung drohen sehr fatale Wertebereichsverletzungen (siehe Ariane 5 - Absturz)
- Wertebereichs- oder Performanceprobleme können die mathematische Sichtweise überstimmen (sehr große ganze Zahl als 10E19 darstellen)
- Die durch die endliche Bitzahl vorhandenen oberen Grenzen sind explizit zu dokumentieren und deren Verletzung vollständig zu unterbinden bzw. definiert abzufangen (z.B. bei $1/\infty$ weiter mit 0 rechnen).

Wertebereich von binären Zahlen

Wertekombinationen
(am Beispiel von 3 bits)

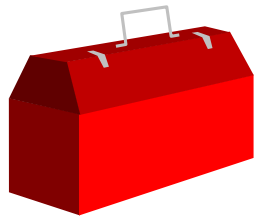
BITS	Zahl
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

MERKE:

Der Bereich umfaßt das Intervall

0 bis 2^N-1

Bits	Werteanzahl (von 0 bis Anzahl-1)	
1	2	
2	4	
3	8	
4	16	
5	32	
6	64	
7	128	
8	256	
10	1024	= 1 K
16	65.536	= 64 K
20	1.048.576	= 1 Mega (ca. 1 Mio)
30	1.073.741.824	= 1 Giga (ca. 1 Mrd)
32	4.294.967.296	= 4 Giga (ca. 4 Mrd.)

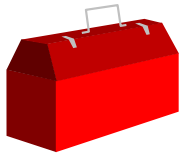


Die unterstrichenden Werteangaben sind auswendig zu lernen !

Darstellung ganzer Zahlen (INTEGER)

Ganze und natürliche Zahlen werden in der Regel durch einfache Zusammenfassung von 8, 16 oder 32 Bit abgebildet:

Bits	Werteanzahl (von 0 bis Anzahl-1)	
8	256 (von 0 bis 255)	
16	65.536	= 64 K
32	4.294.967.296	= 4 Giga (ca. 4 Mrd.)



MERKE:
Der Bereich umfaßt das Intervall
0 bis 2^N-1

Wie kann das Vorzeichen gespeichert werden ?

- einfacher Trick: Zweckentfremdung des höchsten Bits (8./16. Oder 32.) als Vorzeichen - Bit=0 -> Positiv Bit=1 -> Negativ
- Folge: Wertebereich bleibt zwar in der Anzahl gleich groß, reduziert sich im positiven wie negativen Bereich um eine Zweierpotenz

Bits	Bereich
8	+127 bis –127 (-128)
16	zwischen +/-32000

MERKE: Der Bereich umfaßt das Intervall
$-(2^{N-1}-1)$ bis $+(2^{N-1}-1)$

Konvertierung von binären Zahlen

Menschen sind die Arbeit mit Dezimalsystem gewöhnt !

- (Zahlensystem mit 10 Zuständen -> Anzahl Wert = 10Stellenzahl)
- zur Kommunikation mit dem Computer müssen alle Zahlen in das binäre System und danach wieder zurück umgerechnet (konvertiert) werden
- Konvertierungen in das Hexadezimalsystem (Basis 16) sind analog durchzuführen

Algorithmus zur Konvertierung von Dezimalzahl zu Dualzahl

(heute in jedem besseren Compilerpaket drin):

0. Setze Dualzahl auf 0, Setze Summand auf 1
1. Ganzzahldivision der Dezimalzahl durch 2, Schreibe Quotient in Dezimalzahl
2. Falls Divisionsrest =1 dann addiere Summand zur Dualzahl dazu
3. Multipliziere Summand mit 2
4. Falls Dezimalzahl größer als 0 dann weiter bei Schritt 1

HA: Entwickeln Sie den Algorithmus für die Rückkonvertierung !

Darstellung gebrochener und rationaler Zahlen (FLOAT)

Für numerische Werte, welche nicht mit ganzen Zahlen dargestellt werden können, werden in der Regel Gleitkommazahlen verwendet:

- Zahlen mit gebrochenem Anteil 3,1415
- Zahlen größer oder kleiner als $\pm 2^{32}$, auch $3*10^{12}$ (widerspricht an sich der reinen Mathematik und kann zu Problemen bzgl. Genauigkeit führen)

Technische Realisierung:

Zerlegung der Zahl in Exponentialschreibweise (Gleitkommazahl)

$$z = \text{Mantisse} * 10^{\text{Exponent}} = m\ 10^{exp}$$

Trick: Es werden nur die Mantisse und der Exponent gespeichert. Die Basis der Potenz (10 oder 2) wird nicht gespeichert sondern gilt als fest vereinbart.

Die beiden Bestandteile werden in eine Menge von Bytes gepackt.

Wertebereich von Gleitkommazahlen

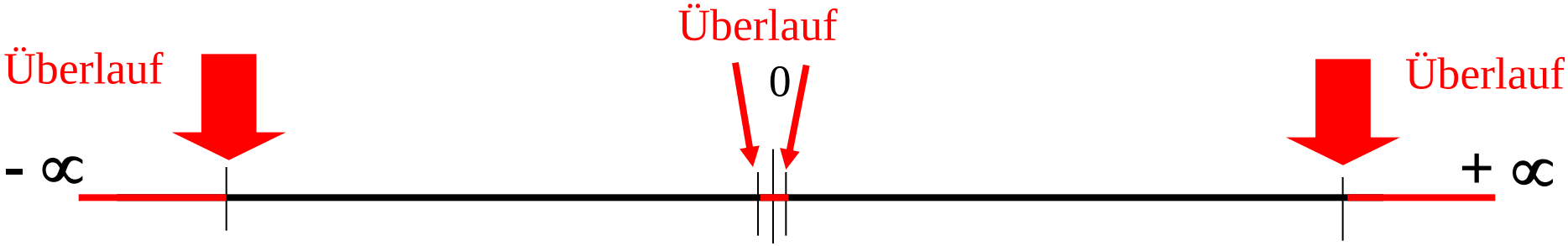
Die Ablage der Gleitkommazahl als BITMUSTER:

- Es gibt keine absolut eindeutige Festlegung der genauen Packreihenfolge.
- Zwischen verschiedenen Computerherstellern können Abweichungen auftreten (z.B. speichert der Prozessor der Apple-Computer die Zahlen teilweise anders !!!!)

Bytes	Bits	[Bits exp]	[Bits m]	Wertebereich	Genauigkeit [Stellen]
4	32	6	(32-6)=26	$+/-(1{,}5*10^{-45}$ bis $3.8*10^{38})$	7-8
8	64	12	(64-12)=52	$+/-(1.7*10^{-308}$ bis $1.7*10^{308})$	15-16
10	80				

Zum einfacheren Verständnis: Größenbereich etwa $+/- 10^{x}$ mit $x = 2^{exp}$

MERKE: Aufgrund des endlichen Bitumfanges des Exponenten besteht sowohl bei sehr großen wie auch sehr kleinen Zahlen die **Gefahr des Überlaufes** bzw. großer Ungenauigkeiten :



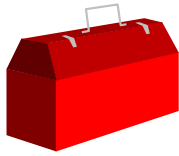
Weitere primitive Datentypen (Boolean / Character)

Wahrheitswerte (Boolean)

- haben nur zwei Zustände WAHR (TRUE) oder FALSCH (FALSE)
- theoretisch ist ein Bit zur Speicherung ausreichend (wird auch bei sehr vielen Wahrheitswerten so auf einzelne Bits abgebildet)
- bei nur einem einzelnen Wahrheitswert erfolgt aus Gründen der Geschwindigkeit eine Speicherung mit 1 oder 2 Byte (da der Prozessor sowieso ein ganzen Zugriff von 8 oder 16 Bit machen muß)

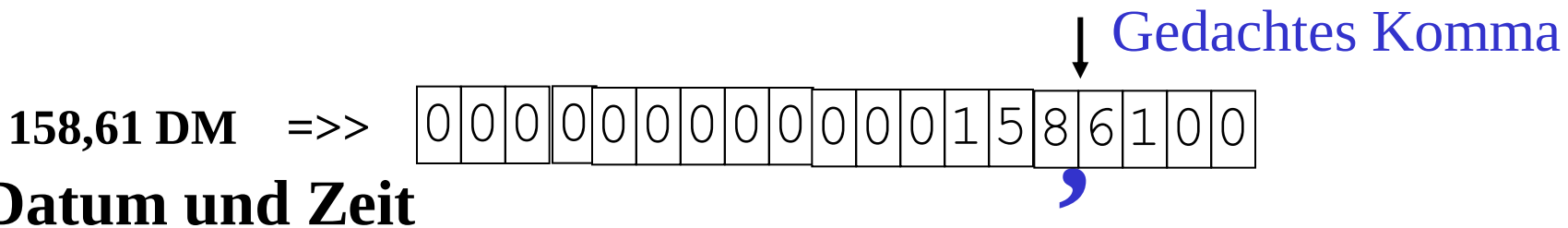
Zeichencodes (Character)

- zur Speicherung eines Textzeichens
- früher mit einem Byte -> 256 verschiedene Zeichen möglich
- Probleme mit anderen Sprachen (Kyrillisch, Arabisch, Asiatisch)
- heute zunehmend mit 2 Byte pro Zeichen (sogenannter Unicode)



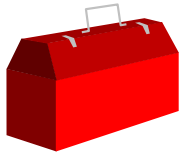
Währungsbeträge

- sehr groß (> 4Mrd.), aber nur begrenzte Zahl von Nachkommastellen
- Lösung: als 8 Byte Ganzzahl mit Verschiebung um 4 Nachkommastellen
- 8 Byte -> 8 * 8 Bit = 64 Bit = 2^{64} Zustände = 2^{10*6*2^4} = ca. $16*10^{18}$
- ergibt theoretisch 19 Stellen, davon 4 weg für Nachkommastellen



Datum und Zeit

- Anforderungen: schnell, effizient, leichte Berechnung von Intervallen
- Speicherung der Sekunden seit einem Stichtag (z.B. 1.1.1980)
- Speicherung als gebrochene Zahl (Microsoft -> alle Windowsprogramme):
  - Ganzer Teil = Datum seit Stichtag
  - gebrochener Teil = Uhrzeit /24h
- echtes Problem: Berechnung von Wochentagen (Schaltjahre) ?



Arbeiten mit verschiedenen Zahlensystemen

Unsere heutigen Zahlensysteme (und nicht das römische!)

sind sogenannte Positionssysteme, d.h.

Der Wert einer Zahl wird eindeutig durch
Form und Position der einzelnen Zeichen
bestimmt.

- Dies kann mathematisch ausgedrückt werden mit

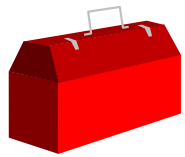
$$n = \sum_{i=0}^{N-1} b_i \cdot B^i$$

oder bei gebrochenen Zahlen mit

$$g = \sum_{i=-M}^{N-1} b_i \cdot B^i$$

Mit

- n : Zahlenwert als Summe der Positionswerte
- B : Basis des Zahlensystems (mit B >= 2)
- b_i : Ziffern (mit b_i = 0, … B-1)
- N : Anzahl der Stellen (wegen Start bei i=0 bis N-1)



Arbeiten mit verschiedenen Zahlensystemen

Als Basis B werden verschiedene Werte verwendet

und es ergeben sich folgende Zahlensysteme:

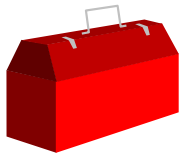
Namen für Zahlensysteme		**gleiche Beispielzahl**:
• *B*= 2	: **Dualsystem** (Binärsystem)	10011
• *B*= 8	: Oktalsystem (selten)	23
• *B*=10	: **Dezimalsystem**	**19**
• *B*=12	: Zwölfersystem	17
	(in der Informatik nicht gebräuchlich)	
• *B*=16	: **Hexadezimalsystem (häufig!!)**	13

$$n = \sum_{i=0}^{N-1} b_i \cdot B^i$$

Problem: für B=16 reichen unsere Ziffern 0-9 nicht aus ➔ zusätzlich die Buchstaben A-F

(A=10 B=11 C =12 D=13 E =14 F=15)

damit wird 31 (Dezimal) zu 1 *16 +15 = 1F



Konvertierung in das Dezimalsystem

Zur eindeutigen Kennzeichnung werden die Zahlen ggf. mit einem tiefgestellten Index $(\text{Zahl})_B$ mit B=Basis gekennzeichnet (vgl. Beispiel unten rechts)

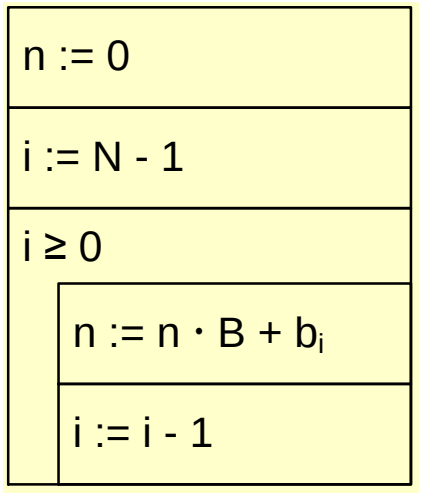
Eine im Positionssystem mit der Basis B dargestellte natürliche Zahl n

$$n = \sum_{i=0}^{N-1} b_i \cdot B^i \qquad (N \geq 1)$$

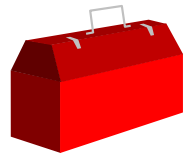
läßt sich mit Hilfe des Hornerschemas (=> Math.) wie folgt darstellen:

$$n = \left(\ldots\left(\left(b_{N-1} \cdot B + b_{N-2}\right) \cdot B + b_{n-3}\right) \cdot B + \ldots + b_1\right) \cdot B + b_0$$

Mit Hilfe dieser Darstellung können Konvertierungen **in das Dezimalsystem** einfach durchgeführt werden:



$$(1578)_{10} = ((1 \cdot 10 + 5) \cdot 10 + 7) \cdot 10 + 8$$
$$(754)_8 = (7 \cdot 8 + 5) \cdot 8 + 4 = (492)_{10}$$
$$(375)_8 = (3 \cdot 8 + 7) \cdot 8 + 5 = (253)_{10}$$
$$(1210)_8 = ((1 \cdot 8 + 2) \cdot 8 + 1) \cdot 8 + 0 = (648)_{10}$$
$$(888)_9 = (8 \cdot 9 + 8) \cdot 9 + 8 = (728)_{10}$$
$$(ADA)_{16} = (10 \cdot 16 + 13) \cdot 16 + 10 = (2778)_{10}$$



Konvertierung aus dem Dezimalsystem

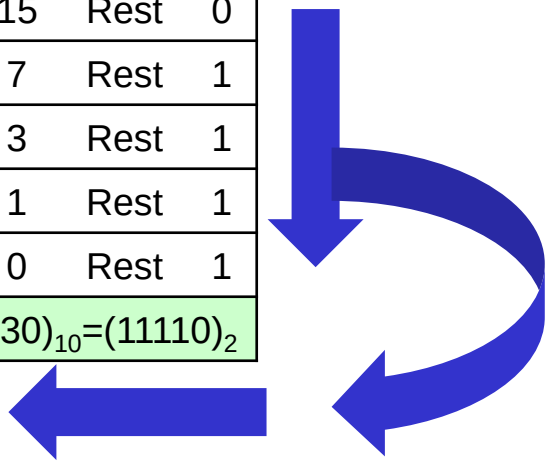
Ebenfalls auf Basis des Horner-Schemas:

Umwandlung einer Dezimalzahl *x* in ein Positionssystem mit der Basis *B*:

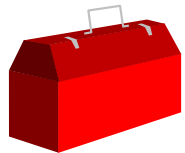
1. x : B = **y Rest z** **(Rest engl. Modulo)**
2. x := y
   a) falls x≠0 → Schritt 1
   b) falls x=0 → Schritt 3
3. Ergebniszahl: Reste z von unten nach oben nebeneinander geschrieben

z := x mod B
x := ⌊x/B⌋
x > 0

$(30)_{10}=?_2$						
x				y		z
30	:	2	=	15	Rest	0
15	:	2	=	7	Rest	1
7	:	2	=	3	Rest	1
3	:	2	=	1	Rest	1
1	:	2	=	0	Rest	1
→ $(30)_{10}=(11110)_2$						



$(43)_{10}=?_2$						
x				y		z
43	:	2	=	21	Rest	1
21	:	2	=	10	Rest	1
10	:	2	=	5	Rest	0
5	:	2	=	2	Rest	1
2	:	2	=	1	Rest	0
1	:	2	=	0	Rest	1
→ $(43)_{10}=(101011)_2$						



Konvertierung echt gebrochener Zahlen aus dem Dezimalsystem

Algorithmus zur Konvertierung einer echt gebrochenen Dezimalzahl x in ein Positionssystem mit der Basis B:

1. x · B = y Überlauf z (z = ganzzahliger Anteil) **(Multiplikation beachten !!!)**
2. x := y – z
   a) x≠0 und noch nicht genügend Nachkommastellen ermittelt → Schritt 1
   b) sonst → Schritt 3
3. Ergebniszahl: Schreibe hinter 0. (bzw. 0,) die ermittelten Überläufe von oben nach unten nebeneinander,

$(0.34375)_{10} = ?_2$

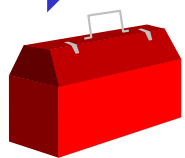
x				y		z
0.34375	·	2	=	0.6875	Überlauf	0
0.6875	·	2	=	1.375	Überlauf	1
0.375	·	2	=	0.75	Überlauf	0
0.75	·	2	=	1.5	Überlauf	1
0.5	·	2	=	1.0	Überlauf	1
0.0	·	2	=	0.0	Überlauf	0

→ $(0.34375)_{10} = (0.01011)_2$

$(0.408203125)_{10} = ?_8$

x				y		z
0.408203125	·	8	=	3.265625	Überlauf	3
0.265625	·	8	=	2.125	Überlauf	2
0.125	·	8	=	1.0	Überlauf	1
0.0	·	8	=	0.0	Überlauf	0

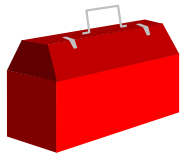
→ $(0.408203125)_{10} = (0.321)_8$



Konvertierung unecht gebrochener Zahlen aus dem Dezimalsystem

1. Aufteilung in ganzzahligen und echt gebrochenen Teil
2. getrennte Konvertierung beider Teile *(auch in den realen Konvertierungsprogrammen (vgl. VB Texte) wird dies getrennt ausgeführt ...)*
3. danach Zusammenfassung als Zahl

$(12.25)_{10}=(?)_2$														
ganzzahliger Teil: $(12)_{10}=(?)_2$							echt gebrochener Teil: $(0.25)_{10}=(?)_2$							
x			y			z	x			y			z	
12	:	2 =	6	Rest		0	0.25	·	2 =	0.5	Überlauf		0	
6	:	2 =	3	Rest		0	0.5	·	2 =	1.0	Überlauf		1	
3	:	2 =	1	Rest		1	0.0	·	2 =	0.0	Überlauf		0	
1	:	2 =	0	Rest		1								
→ $(12)_{10}=(1100)_2$							→ $(0.25)_{10}=(0.01)_2$							
→$(12.25)_{10}=(1100.01)_2$														



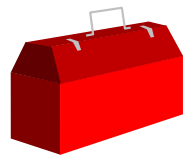
Probleme bei der Konvertierung in andere Zahlensysteme

Viele gebrochene Zahlen, die sich ganz genau im Dezimalsystem darstellen lassen („abbrechende Kommazahlen“), lassen sich nicht ganz genau als Dualzahl darstellen, da sie dort entweder nicht abbrechende, periodische Zahlen sind, von denen jedoch nur die ersten *p* Ziffern gespeichert werden, oder zwar abbrechende Zahlen sind, jedoch mit mehr als *p* Ziffern, wodurch Ungenauigkeit entsteht.

Beispiel: $0.1_{(10)}=0.0001100110011\ldots_{(2)}$.
Das Bitmuster 0011 wiederholt sich ständig.

x				y		z
0,1	·	2	=	0,2	Überlauf	0
0,2	·	2	=	0.4	Überlauf	0
0,4	·	2	=	0,8	Überlauf	0
0,8	·	2	=	1,6	Überlauf	1
0,6	·	2	=	1,2	Überlauf	1
0,2	·	2	=	0,4	Überlauf	0
0,4	·	2	=	0,8	Überlauf	0
0,8	·	2	=	1,6	Überlauf	1
0,6	·	2	=	1,2	Überlauf	1
…						

Genau diese Kenntnis zu möglichen Problemen bei der Konvertierung ist besonders praxisrelevant wegen möglicher Fehler bei der Berechnung (und damit auch Klausur-relevant)



Rechnen mit Binärzahlen : Addition

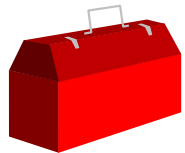
Für die duale Addition gilt allgemein:

0	+	0					=	0		
0	+	1					=	1		
1	+	0					=	1		
1	+	1					=	0	Übertrag	1
1	+	1	+	1	(vom Übertrag)		=	1	Übertrag	1

Beispiel: 45 + 54

0	1	0	1	1	0	1	=	45
0	1	1	0	1	1	0	=	54
1	1	1	1				=	Übertrag
1	1	0	0	0	1	1	=	99

Hinweis: Diese Vorgehensweise kann relativ einfach mit logischen Schaltungen realisiert werden (vgl. VL Hardware)



Rechnen mit Binärzahlen: Bitverschiebung und Multiplikation

Die **Linksverschiebung** eines vorzeichenlosen ganzzahligen Wertes um **n** Bits entspricht einer **Multiplikation mit 2^n**.

Bitdarstellung für **i** = 0000000000000101

```
i = i << 1;    // einmal bitweise nach links
```

Nach dieser Operation enthält **i** folgende Bitdarstellung:

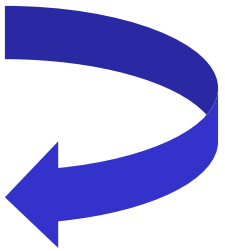
i = 0000000000001010 → Wert von **i** ist 10 (2*5)

```
i = i << 2;    // erneut 2mal bitweise nach links
```

i = 0000000000101000 → Wert von **i** ist 40 (2*4*5)

Diese Eigenschaft von Binärzahlen wird ausgenutzt zur Multiplikation von Binärzahlen: Ein Faktor wird immer weiter nach links geschoben und immer auf das Endergebnis aufaddiert, wenn das i.-Bit des anderen Faktors =1 ist ! (Realisierung mit Schiebregister + Addierer oder als N-facher Addierer bei N-stelligen Binärzahlen - mehr Hardware, aber schneller)

Beispiel: 45 * 5 = 0101101 * 101



0101101	1 => Addieren
	0 => NICHT addieren
0101101	1 => Addieren
111100001	**Ergebnis (= 225 (Dezimal))**