

VO 050041:

Technische Grundlagen der Informatik

Begleitende Folien zur Vorlesung

Wintersemester 2016/17

Vortragende: Peter Reichl, Andreas Janecek

Zuletzt aktualisiert: 9. Dezember 2016

Teil 2:

Zahlensysteme und -darstellungen

Überblick

- 1 Zahlensysteme
- 2 Konvertierung zwischen Zahlensystemen
- 3 Arithmetische Operationen im Dualsystem
- 4 Zahlendarstellungen am Computer
- 5 Numerik

Literatur

- Mikroprozessortechnik (Wüst, Vieweg+Teubner): Kapitel 2
- Informatik (Blieberger, Springer-Verlag): Kapitel 7+8
- *(Computerarchitektur (Tanenbaum): Anhang A und B)*

Überblick

- 1 Zahlensysteme
 - Additionssysteme
 - Stellenwertsysteme
- 2 Konvertierung zwischen Zahlensystemen
- 3 Arithmetische Operationen im Dualsystem
- 4 Zahlendarstellungen am Computer
- 5 Numerik

Additionssysteme: Beispiel

Römisches Zahlensystem

- **Wert einer Zahl durch Form und *Anzahl* der Zeichen (Symbole) bestimmt**
- $I=1$, $V=5$, $X=10$, $L=50$, $C=100$, $D=500$, $M=1000$
- $XLIX = 49 = -10+50-1+10$
- $CDXCV = 495 = -100+500-10+100+5$
- $MMXIV = 2014 = 1000+1000+10+1-1+5$

Additionssysteme: Überblick

Additionssysteme

- Addition einfach, kein Übertrag
- Restliche Operationen mühsam
- Darstellung großer Zahlen mühsam
- Keine Null!
- Maschinelle Darstellung?!

Stellenwertsysteme

Stellenwertsysteme: Überblick

- **Wert einer Zahl durch Form und *Position* der Zeichen (Symbole) bestimmt**
- “Positionssystem”, “Polyades Zahlensystem”
- **Basis B :** $B \in \mathbb{N}$; $B \geq 2$
- Zahl x wird in Potenzen von B zerlegt
- B verschiedene Symbole, Ziffern 0 bis $B - 1$
- n ... Anzahl der Ziffern, b_i ... Werte der einzelnen Ziffern

$$x = \sum_{i=0}^{n-1} b_i \cdot B^i = b_0 \cdot B^0 + b_1 \cdot B^1 + b_2 \cdot B^2 + \cdots + b_{n-1} \cdot B^{n-1}$$

Dezimalsystem

Basis 10

$$x = 2017_{10} = 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 7 \cdot 10^0$$

Binärsystem (“Dualsystem”)

Basis 2

$$x = 11001_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 25_{10}$$

Nur zwei verschiedene Zustände!

- = Minimalvariante
- **technisch am einfachsten realisierbar**

Oktalsystem

Basis 8

$$x = 315_8 = 3 \cdot 8^2 + 1 \cdot 8^1 + 5 \cdot 8^0 = 205_{10}$$

Hexadezimalsystem

Hexadezimalsystem

Hex:	0	1	2	...	9	A	B	C	D	E	F	10
Dez:	0	1	2	...	9	10	11	12	13	14	15	16

Basis 16

$$x = F1_{16} = 0xF1 = 15 \cdot 16^1 + 1 \cdot 16^0 = 241_{10}$$

Gebrochene Zahlen: Überblick

- Bei **gebrochenen Zahlen** trennt das Komma den ganzzahligen vom gebrochenen Teil
- **Basis** B ; $B \geq 2$; Ziffern 0 bis $B - 1$
- N : Anzahl der **signifikanten** Stellen **vor** dem Komma
- M : Anzahl der **signifikanten** Stellen **nach** dem Komma

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

$$\begin{aligned} & b_{-M} \cdot B^{-M} + b_{-M+1} \cdot B^{-M+1} + b_{-M+2} \cdot B^{-M+2} + \dots + \\ & \dots + b_0 + b_1 \cdot B + b_2 \cdot B^2 + \dots + \\ & \dots + b_{N-2} \cdot B^{N-2} + b_{N-1} \cdot B^{N-1} \end{aligned}$$

Gebrochene Zahlen: Beispiel

Gebrochene Dezimalzahl

$$x = 23,42_{10} = 2 \cdot 10^{-2} + 4 \cdot 10^{-1} + 3 \cdot 10^0 + 2 \cdot 10^1 = 23,42_{10}$$

Gebrochene Binärzahl

$$x = 10,011_2 = 1 \cdot 2^{-3} + 1 \cdot 2^{-2} + 0 \cdot 2^{-1} + 0 \cdot 2^0 + 1 \cdot 2^1 = 2,375_{10}$$

Überblick

1 Zahlensysteme

2 Konvertierung zwischen Zahlensystemen

Basis 10 \Rightarrow Basis X \Rightarrow 10

Zwischen Basen 2, 8, 16

3 Arithmetische Operationen im Dualsystem

4 Zahlendarstellungen am Computer

5 Numerik

Methode für Ganzzahlen

Basis 10 \Rightarrow 2

- (Ganz)Zahl so lange durch 2 dividieren, bis 0 erreicht
- Jeweilige Reste (rückwärts gelesen) ergeben Binärzahl

Basis 2 \Rightarrow 10

- Potenzen von 2 summieren, entsprechen den gesetzten Bits der Binärzahl
- Siehe Folien zu Stellenwertsysteme

Beispiel: 373_{10}

Quotient	Rest
373	
186	1
93	0
46	1
23	0
11	1
5	1
2	1
1	0
0	1

$373_{10} = 101110101_2$ (Tabelle von unten nach oben lesen!)

Methode für Ganzzahlen

Basis 10 \Rightarrow 16

- Zahl so lange durch 16 dividieren, bis 0 erreicht
- Jeweilige Reste ergeben Hexadezimalzahl
- Achtung: falls Rest $> 9 \rightarrow$ durch entspr. Buchstaben ersetzen!

Basis 16 \Rightarrow 10

- Potenzen von 16 summieren
- Siehe Folien zu Stellenwertsysteme

Beispiel: 373_{10}

Quotient	Rest
$373 : 16 = 23$	5
$23 : 16 = 1$	7
$1 : 16 = 0$	1

$373_{10} = 175_{16}$ (Tabelle von unten nach oben lesen!)

Beispiel: 2702_{10}

Quotient	Rest
$2702 : 16 = 168$	14 (E)
$168 : 16 = 10$	8
$10 : 16 = 0$	10 (A)

$2702_{10} = A8E_{16}$ (Tabelle von unten nach oben lesen!)

Methode für Gebrochene Zahlen

Basis 10 \Rightarrow 2

- Ganzzahlen (Zahlen links vom Komma) trennen

\Rightarrow “normale” Umrechnung

- Ziffernfolge rechts vom Komma kann nicht immer exakt berechnet werden

\Rightarrow Häufig nur Näherungswert möglich

Einfache Beispiele für exakte Umrechnung

Dezimalsystem	Dualsystem
0,5	$0,1_2$
0,25	$0,01_2$
0,125	$0,001_2$

Basen mit GGT 2

Abkürzung

- Basis 2 \Rightarrow Basis 8: Dreiergruppen bilden ($2^3 = 8$)

0 111 101 110 100 011
 └─┘ └─┘ └─┘ └─┘ └─┘
 7 5 6 4 3₈

- Basis 2 \Rightarrow Basis 16: Vierergruppen bilden ($2^4 = 16$)

0111 1011 1010 0011
 └─┘ └─┘ └─┘ └─┘
 7 B A 3₁₆

Geht natürlich auch in die andere Richtung!

Überblick

1 Zahlensysteme

2 Konvertierung zwischen Zahlensystemen

3 Arithmetische Operationen im Dualsystem

Exkurs: Bits und Bytes

Addition

Subtraktion

Einfache Multiplikation/Division

4 Zahlendarstellungen am Computer

5 Numerik

Bit

- **binary digit**
- Zwei Zustände: “0” und “1”
- 1 bit \Rightarrow 2 Zustände, 2 bit \Rightarrow 4 Zustände,
3 bit \Rightarrow 8 Zustände, ... , n bit $\Rightarrow 2^n$ Zustände

Vielfache von Bit

Dezimalpräfix			Binärpräfix		
Name	Symbol	Wert	Name	Symbol	Wert
Kilobit	kbit	10^3	Kibibit	Kibit	2^{10}
Megabit	Mbit	10^6	Mebibit	Mibit	2^{20}
Gigabit	Gbit	10^9	Gibibit	Gibit	2^{30}
...

Byte

- Folge von (üblicherweise) 8 Bit \Rightarrow "Oktett"



1

¹http://www.teach-ict.com/gcse_computing/ocr/214_representing_data/units/miniweb/images/bitbyte.jpg

Byte

Vielfache von Byte. VORSICHT: Oft ist unklar welcher Präfix tatsächlich verwendet wird!

Dezimalpräfix			Binärpräfix		
Name	Sym.	Wert	Name	Sym.	Wert
Kilobyte	kB	10^3 Byte	Kibibyte	KiB	2^{10} Byte
Megabyte	MB	10^6 Byte	Mebibyte	MiB	2^{20} Byte
Gigabyte	GB	10^9 Byte	Gibibyte	GiB	2^{30} Byte
...

Achtung! Erhebliche Unterschiede

Mit ansteigender Größe der Dezimal- und Binärpräfixe wird die Unterscheidung bedeutender

Addition im Dualsystem

Additionstabelle

A	+	B	=	Übertrag	Σ
0	+	0	=	0	0
0	+	1	=	0	1
1	+	0	=	0	1
1	+	1	=	1	0

Übertrag (engl. carry) bei nächsthöherer Stelle berücksichtigen!

Addition im Dualsystem

Beispiel: Zwei 8 Bit Dualzahlen addieren

$$\begin{array}{r} \\ + \\ \hline \textcolor{red}{1} \end{array}$$

Carry-Bit

- **Ergebnis kann 9 Bit lang sein!**
- Wird in einem **Spezialregister** gespeichert
- Muss nach Operation ausgelesen werden
- Ansonsten würden wir most significant bit verlieren und einen Überlauf haben!

Most/Least Significant Bit

msb und lsb

- msb = most significant **bit** = höchstwertiges bit
- lsb = least significant **bit** = niedrigstwertiges bit

Dualsystem

$$10011_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Vergleich: Dezimalsystem

$$17564_{10} = 1 \cdot 10^4 + 7 \cdot 10^3 + 5 \cdot 10^2 + 6 \cdot 10^1 + 4 \cdot 10^0$$

Most/Least Significant Byte

Byte

1 byte = 8 bit, auch Oktett genannt

MSB und LSB

- MSB = Most Significant **Byte** = höchstwertiges Byte
- LSB = Least Significant **Byte** = niedrigstwertiges Byte

Achtung auf die Schreibweise

msb/lb vs. MSB/LSB

Subtraktion im Dualsystem

Subtraktion

- Prinzipiell ist eine Subtraktion im Dualsystem relativ einfach durchführbar (negatives Ergebnis möglich!)
- Rechner können keine direkte Subtraktion durchführen (würde zusätzliche Schaltungen erfordern)
- **Subtraktion auf Addition zurückführbar**
- **Komplement des Subtrahenden wird zum Minuenden addiert**
- **Komplementbildung** im Dualsystem besonders einfach (Invertierung bzw. **NOT**)
- **Einerkomplement vs. Zweierkomplement**

Stellenkomplement (“Einerkomplement”)

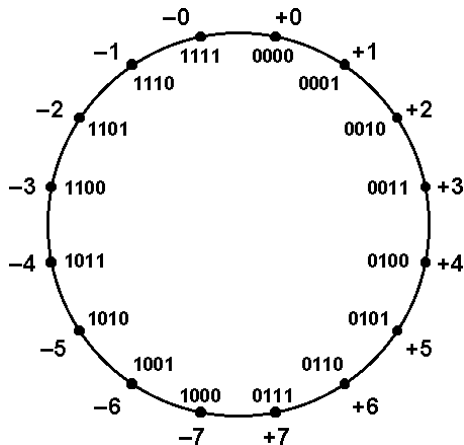
Stellenkomplement $\overline{B - 1}$

- Nullen und Einser werden vertauscht
- Stellenkomplement im Binärsystem auch “Einerkomplement” genannt
- **Addition des Komplements $\overline{B - 1}$ zur urspr. Zahl B ergibt immer höchstzulässige Zahl** - ergibt sich aus der Definition!

Beispiel

$$\begin{array}{rcl}
 B & = & 1 \ 0 \ 1 \ 0_2 \\
 + \ \overline{B - 1} & = & 0 \ 1 \ 0 \ 1_2 \\
 B + \overline{B - 1} & = & 1 \ 1 \ 1 \ 1_2
 \end{array}$$

Stellenkomplement (“Einerkomplement”)



Stellenkomplement (“Einerkomplement”)

Eigenschaften

- Zwei Darstellungen für Zahl 0
 - Einmal mit positivem und einmal mit negativem Vorzeichen
- ⇒ Redundanz (begrenzte Anzahl an Bits)
- ⇒ Bringt Probleme, wenn bei einer Operation die Null durchschritten wird

Verbesserung: Zweierkomplement

- Diese Probleme werden bei der Kodierung von Zahlen in der Zweierkomplementdarstellung vermieden

Zweierkomplement

Zweierkomplement \bar{B}

- **Vollständiges Komplement** zur jeweiligen Basis

⇒ Dezimalsystem: 10, Binärsystem: 2, ...

- Basis 2: “Zweierkomplement”
- Berechnung: Stellenkomplement + 1
- **Addition von \bar{B} zur urspr. Zahl B ergibt immer Null**

Beispiel

$$B = 1010_2$$

$$\bar{B} = 0110_2$$

$$B + (\bar{B}) = 10000_2$$

Zweierkomplement

Trick zur schnelleren Umwandlung (Wikipedia)

- ... einer negativen in eine positive Binärzahl oder umgekehrt von Hand:

⇒ Von rechts anfangen, alle Nullen und die erste Eins abschreiben und alle nachfolgenden Stellen invertieren

Beispiel

$$B = 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0_2$$

$$\overline{B} = 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0_2$$

Zweierkomplement

Interpretation des “Vorzeichenbits” (Wikipedia)

- Alle Bits haben die gleiche Wertigkeit wie bei positiver Darstellung
- ABER: Das msb (most significant bit = höchstwertige bit) erhält die negative Wertigkeit

⇒ msb wird abgezogen (falls es 1 ist)

Beispiel

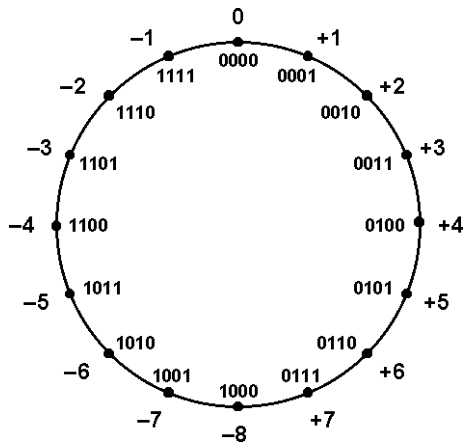
$$B = 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0_2 = +40_{10}$$

$$\overline{B} = 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0_2 = -40_{10}$$

$$B = 32 + 8 = +40_{10}$$

$$\overline{B} = -128 + 64 + 16 + 8 = -40_{10}$$

Zweierkomplement



Komplemente

Es gilt:

$$B + \overline{B - 1} + 1 = 0$$

$$B + \overline{B} = 0$$

$$\overline{B} = -B$$

Zweierkomplement auch interpretierbar als negative Größe der Zahl B !

Komplement im Dezimalsystem

Differenz zweier zweistelliger Zahlen im Dezimalsystem:

$$b - a = b - a + 100 - 100 \quad (1)$$

$$b - a = b + (100 - a) - 100 \quad (2)$$

$$b - a = b + \bar{a} - 100 \quad (3)$$

- Differenz $(100 - a)$, d.h. die Differenz zur **nächsthöheren** Zehnerpotenz = Komplement von a , Symbol: \bar{a}
- Anstatt $b - a$ rechnen wir $b + \bar{a}$ und subtrahieren anschließend 100.
- Subtraktion von 100 → Übertrag in dritter Stelle streichen

Komplement im Dezimalsystem

Beispiel: $17 - 14$

$$\begin{array}{r} 17_{10} \\ - 14_{10} \\ \hline 3_{10} \end{array}$$

Komplement:

$$\begin{array}{r} 100_{10} \\ - 14_{10} \\ \hline 86_{10} \end{array}$$

$$\begin{array}{r} 17_{10} \\ + 86_{10} \\ \hline 103_{10} \end{array}$$

- Komplementbildung erfordert immer noch Subtraktion, im Binärsystem allerdings nicht (kommt gleich)
- Anderes Problem: Was passiert wenn Ergebnis negativ, d.h. $b - a$ bei $a > b$?

Komplement im Dezimalsystem

Beispiel: $45 - 81$

$$\begin{array}{r} 45_{10} \\ - 81_{10} \\ \hline - 36_{10} \end{array}$$

Komplement:

$$\begin{array}{r} 100_{10} \\ - 81_{10} \\ \hline 19_{10} \end{array}$$

$$\begin{array}{r} 45_{10} \\ + 19_{10} \\ \hline 64_{10} \end{array}$$

- **ERGEBNIS IST FALSCH!**
- Kein Übertrag aufgetreten, daher auch nicht streichbar
- Subtraktion fehlt! Zwischenergebnis rückkomplementieren!

$$45 - 81 = 45 + (100 - 81) - 100 \quad (4)$$

$$= 45 + 19 - 100 \quad (5)$$

$$= 64 - 100 = -36 \quad (6)$$

Komplement im Dezimalsystem

Rückkomplementieren

$$b - a = b + (100 - a) - 100 \quad (7)$$

$$b - a = [b + (100 - a)] - 100 \quad (8)$$

$$b - a = c - 100 \quad (9)$$

$$b - a = -(100 - c) \quad (10)$$

- $c < 0$; $-(100 - c)$ = negatives Komplement von c !
- **Falls kein Übertrag vorhanden: Rückkomplementieren**

⇒ Ergebnis ist negativ

- **Falls Übertrag vorhanden, diesen streichen**

⇒ Ergebnis ist positiv

Komplement

Komplement im Binärsystem

- Wir bilden zuerst das **Stellen**komplement
- Für jede Ziffer: Differenz zu größtmöglichem Wert ($= 1$)
- Ist die Ziffer 1, ist die Differenz 0
- Ist die Ziffer 0, ist die Differenz 1
- Entspricht Boole'scher (logischer) Operation **NOT**
- Für vollständiges (Zweier-)Komplement: $+1$

Im Dezimalsystem (zweistellige Zahl)

- $(99 - a)$: **Stellen**komplement, ziffernweise berechenbar!
- Für jede Ziffer: Differenz zu größtmöglichem Wert ($= 9$)
- Vollständiges Komplement durch Addition von 1

Subtraktion im Dualsystem

Beispiel: Zwei 8 Bit Dualzahlen subtrahieren

55_{10}		0	0	1	1	0	1	1	1_2
$- 26_{10}$		0	0	0	1	1	0	1	0_2
<hr/>									
29_{10}		?	?	?	?	?	?	?	$?_2$

Subtraktion im Dualsystem

1.) Komplementbildung des Subtrahenden

<i>NOT</i>	0	0	0	1	1	0	1	0_2
=	1	1	1	0	0	1	0	1_2
+								1_2
	1	1	1	0	0	1	1	0_2

2.) Addition

Übertrag vorhanden \Rightarrow Ergebnis positiv!

	0	0	1	1	0	1	1	1_2
+	1	1	1	0	0	1	1	0_2
1	0	0	0	1	1	1	0	$1_2 = 29_{10}$

Subtraktion im Dualsystem

Beispiel: Zwei 8 Bit Dualzahlen subtrahieren

	26_{10}		0	0	0	1	1	0	1	0_2
-	55_{10}		0	0	1	1	0	1	1	1_2
	-29_{10}		?	?	?	?	?	?	?	$?_2$

1.) Komplementbildung des Subtrahenden

<i>NOT</i>		0	0	1	1	0	1	1	1_2
=		1	1	0	0	1	0	0	0_2
+									1_2
		1	1	0	0	1	0	0	1_2

Subtraktion im Dualsystem

2.) Addition

$$\begin{array}{r}
 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0_2 \\
 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1_2 \\
 \hline
 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1_2
 \end{array}$$

3.) Rückkomplementbildung

KEIN Übertrag \Rightarrow **Ergebnis negativ!**

Zwischenergebnis:	1	1	1	0	0	0	1	1_2
Stellenkompl.:	0	0	0	1	1	1	0	0_2
+1	0	0	0	0	0	0	0	1_2
=	0	0	0	1	1	1	0	1_2

- $11101_2 = 29_{10} \Rightarrow$ Minus davorschreiben: -29_{10}

Verschiebeoperationen (Shifting)

Verschieben nach links (bei Big Endian)

Verschieben einer Zahl um S Stellen nach **links** entspricht **Multiplikation** mit B^S

$$x \cdot B^S = \left(\sum_{i=0}^{N-1} b_i \cdot B^i \right) \cdot B^S = \sum_{i=0}^{N-1} b_i \cdot B^i \cdot B^S = \sum_{i=0}^{N-1} b_i \cdot B^{i+S}$$

Beispiel: $01100111_2 = 8\text{bit unsigned integer}$

Stelle	7	6	5	4	3	2	1	0
Vorher	0	1	1	0	0	1	1	1
Nachher	1	1	0	0	1	1	1	0

Vorsicht vor Überläufen (Overflows)!

Verschiebeoperationen (Shifting)

Verschieben nach rechts (bei Big Endian)

Verschieben einer Zahl um S Stellen nach **rechts** entspricht **Division** durch B^S

$$x \cdot B^{-S} = \left(\sum_{i=0}^{N-1} b_i \cdot B^i \right) \cdot B^{-S} = \sum_{i=0}^{N-1} b_i \cdot B^i \cdot B^{-S} = \sum_{i=0}^{N-1} b_i \cdot B^{i-S}$$

Beispiel: $1010_2 = 4\text{bit unsigned integer}$

Stelle	3	2	1	0
Vorher	1	0	1	0
Nachher	0	1	0	1

Vorsicht vor Unterläufen (Underflows)!

Überblick

- 1 Zahlensysteme
- 2 Konvertierung zwischen Zahlensystemen
- 3 Arithmetische Operationen im Dualsystem
- 4 Zahlendarstellungen am Computer**
 - Endianess
 - Negative Binärzahlen
- 5 Numerik

Big/Little Endian

(Byte)ordnung

Steht am **Anfang** einer Folge aus mehreren Teilen^a der **höchstwertige Teil (Big-Endian)** oder der **niedrigstwertige Teil (Little-Endian)**?

^aBits, Bytes, Words, ...

4660₁₀ im 16-Bit Format

- 4660₁₀ \Rightarrow 1234₁₆
- Big-endian: 12 | 34
 Adresse | Adresse+1
- Little-endian: 34 | 12
 Adresse | Adresse+1

Big/Little Endian - Beispiele

Im Alltag

- Datum: 16. Oktober 2014
- Uhrzeit: 18 Uhr 45 Minuten 17 Sekunden
- Adresse: Boltzmannngasse 3, 1090 Wien, Österreich, ...

Big/Little Endian - Beispiele

Dualsystem: Big Endian

$$10011_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 19_{10}$$

Dualsystem: Little Endian

$$10011_2 = 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 = 25_{10}$$

Konvention

Zahlen: Bei Verwendung von Stellenwertsystemen durch Menschen ist Big Endian Konvention!

Bytereihenfolge

Szenario

- Es müssen mehr Daten transferiert oder gespeichert werden, als in der kleinsten adressierbaren Einheit unterzubringen sind.
- Bsp: 439041101_{10} als 32 bit Integer
- **Big Endian:** 00011010 00101011 00111100 01001101

	Big Endian			Little Endian		
Adresse	Hex	Dez	Binär	Hex	Dez	Binär
10000	1A	26	00011010	4D	77	01001101
10001	2B	43	00101011	3C	60	00111100
10002	3C	60	00111100	2B	43	00101011
10003	4D	77	01001101	1A	26	00011010

Endianess

Relevanz

- Wurde v.a. bei beginnender Vernetzung extrem wichtig
- **AMD & Intel: Little Endian**
 - \Rightarrow niederwertigstes Byte hat niedrigste Adresse
 - **ABER:** Bit-Reihenfolge ist Big Endian!
- **RISC (SPARC, MIPS, PPC, ARM, ...): Big Endian**
 - \Rightarrow höchstwertiges Byte hat niedrigste Adresse
 - Bit- und Byte-Reihenfolge ident
- Auch exotische Zwischendinge möglich, Bi-Endian:
Umschaltbar
 - Byte order: big endian
 - Bit order: little endian

Negative Binärzahlen

Verschiedene Ansätze

- 1 Vorzeichen und Betrag: Vorzeichenbit, dann absolute Größe der Zahl
- 2 Einerkomplement: überholt
- 3 **Zweierkomplement: Standard**
- 4 Exzessdarstellung: Wertebereichsverschiebung

Vorzeichen und Betrag

Vorzeichen und Betrag

n	$n - 1$...	0
VZ	Betrag		

- **Führendes Bit codiert Vorzeichen:** (0/1 \Rightarrow +/-)
- Grundsätzlich:
 - $z = +0$ bis $+2^{\textcolor{red}{n-1}-\textcolor{green}{1}}$ \Rightarrow 000...00 bis 011...11
 - $z = -0$ bis $-(2^{\textcolor{red}{n-1}-\textcolor{green}{1}})$ \Rightarrow 100...00 bis 111...11
- **$n-1$:** Aufteilung in positiven/negativen Bereich benötigt 1 Bit
- **-1 :** Null wird als -0 und $+0$ kodiert

$$-(2^{n-1} - 1) \leq z \leq 2^{n-1} - 1$$

Vorzeichen und Betrag

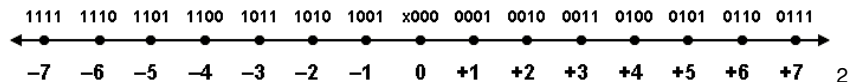
Beispiel

- Wortlänge 4 Bits, $\Rightarrow 2^4 = 16$ Kombinationen
- Bei positiven Zahlen: 0 bis $2^4 - 1 = 15$ darstellbar
- Ausweitung auf negative Zahlen: Darstellungsbereich auf negative und positive Hälfte aufteilen
- **Zahlen von -7 bis 7 darstellbar**

8-Bit-Prozessoren

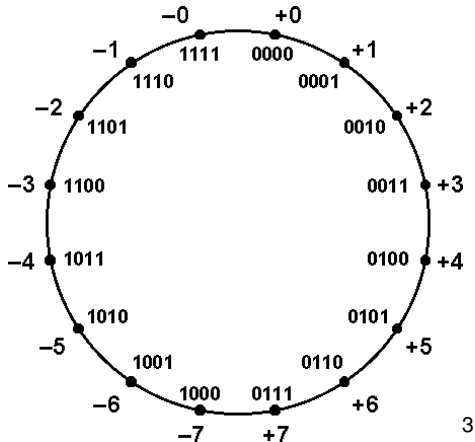
$$x = 8 \Rightarrow -127 \leq z \leq 127$$

Vorzeichen und Betrag



²<http://www.iris.uni-stuttgart.de/lehre/eggenberger/gdi/1/Codierung/Zahlendarstellung/VorzeichenBetrag.gif>

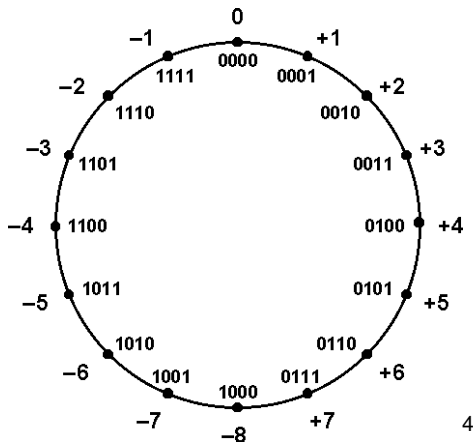
Stellenkomplement (“Einerkomplement”)



3

³<http://www.iris.uni-stuttgart.de/lehre/eggenberger/gdi/1/Codierung/Zahlendarstellung/B-1-Komplement.gif>

Zweierkomplement



⁴<http://www.iris.uni-stuttgart.de/lehre/eggenberger/gdi/1/Codierung/Zahlendarstellung/B-Komplement.gif>

Exzessdarstellung

Exzessdarstellung

- Zur Zahl z wird eine Exzess q addiert, damit das Ergebnis w (= die Darstellung) nicht negativ ist.
- Exzess q muss daher gleich dem Betrag der kleinsten negativen Zahl gewählt werden

Exzessdarstellung

Beispiel: $n = 2^5 \Rightarrow q = 2^4 = 16$

- $z = -2^4$ bis $-1 \Rightarrow 00000$ bis 01111
- $z = 0 \Rightarrow 10000 (= q)$
- $z = 1$ bis $+(2^4 - 1) \Rightarrow 10001$ bis 11111
- Führendes Bit codiert Vorzeichen: $0/1 \Rightarrow -/+$
- Null hat nur eine Codierung!
- **Ordnungsrelation bleibt erhalten!**

\Rightarrow **Vergleiche zwischen Zahlen!**

Überblick

- 1 Zahlensysteme
- 2 Konvertierung zwischen Zahlensystemen
- 3 Arithmetische Operationen im Dualsystem
- 4 Zahlendarstellungen am Computer
- 5 Numerik
 - Festpunkt-Darstellung
 - Gleitpunkt-Darstellung
 - Fehlerfortpflanzung

Numerische Berechnungen und Numerik

- Berechnungen unter Verwendung reeller Zahlen (oder deren *Näherung!*) nennt man *numerische Berechnungen*
- Die dazugehörige mathematische Disziplin: *Numerik*

Näherung

- Speicher und Rechenzeit begrenzt
 - Nur signifikante Stellen speichern
- ⇒ Oft kann eine Zahl nicht exakt im Computer gespeichert werden, sondern nur eine Näherung (*Approximation*) davon
- Beispiel: $0.1_{10} \approx 0.00011001100110011\dots$
- ⇒ Dadurch ergibt sich auch ein entsprechender *Rundungsfehler*

Dezimaltrennzeichen

Notation des Dezimaltrennzeichens: “,” vs. “.”

- Festkomma-Darstellung vs. Fixed-point arithmetic
- Gleitkomma-Darstellung vs. Floating-point arithmetic

Achtung auf die Notation

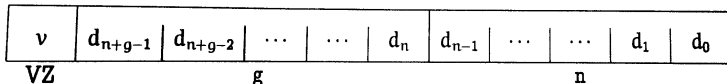
Auf den nächsten Folien wird ein **Punkt** (und kein Komma) als Dezimaltrennzeichen verwendet

Festpunkt-(Festkomma)-Darstellung

Beispiel zweier Dezimalzahlen in Festkomma-Darstellung mit 12 Vorkomma und 22 Nachkommastellen

- 000000000000.00000000000000000001602
- 149700000000.000000000000000000000000

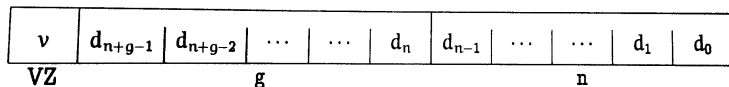
Mögliche Darstellung im Computer



Der Betrag einer $N = n + g + 1$ Bit breiten ganzen Zahl Z wird in g Vorkomma- und n Nachkommastellen unterteilt

Festpunkt-(Festkomma)-Darstellung

Mögliche Darstellung im Computer



$$vd_{N-2}d_{N-3}\cdots d_1d_0 = (-1)^v \cdot 2^{-n} \sum_{j=0}^{N-2} d_j \cdot 2^j = (-1)^v \cdot d_{N-2}\cdots d_1d_0$$

Beispiel: $N = 16$ Bit breite und $n = 3$ Nachkommastellen

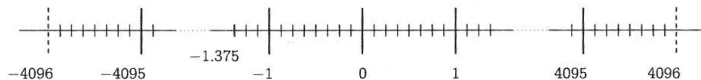
$$vd_{14}d_{13}\cdots d_1d_0 = (-1)^v \cdot 2^{-3} \sum_{j=0}^{14} d_j \cdot 2^j$$

$$1000\ 0000\ 0000\ 1011 = -(1.011)_2 = (-1)^1 \cdot 2^{-3} \cdot (2^3 + 2^1 + 1^0)$$

Festpunkt-(Festkomma)-Darstellung

Bsp: Kleinste und größte darstellbare Zahl mit $N = 16$, $n = 3$

- $1111\ 1111\ 1111\ 1111 = -4095.875_{10}$
- $0111\ 1111\ 1111\ 1111 = +4095.875_{10}$
- Zwei aufeinanderfolgende Zahlen unterscheiden sich jeweils um den Betrag $0000\ 0000\ 0000\ 0001 = 0.125_{10}$
- Damit überdeckt dieses Festpunkt-System auf der reellen Zahlengeraden das Intervall $[-4095.875_{10}, +4095.875_{10}]$ **gleichmäßig** mit konstantem Abstand $2^{-n} = 0.125_{10}$



Festpunkt-Zahlensystem mit $n = 3$ Nachkommastellen

Festpunkt-(Festkomma)-Darstellung

Probleme der Festpunkt-Darstellung

- Intervall zwischen größter und kleinster darstellbarer Zahl sehr klein
 - Größere Zahlen sind nur über eine Reduktion der Nachkommastellen darstellbar
- ⇒ Verlust an Genauigkeit ist für große Zahlen oft vernachlässigbar, da die Bedeutung der Nachkommastellen mit steigenden Absolutbeträgen sinkt
- ⇒ Die Verwendung von sehr kleinen von Null verschiedenen Zahlen ist jedoch sehr wichtig, z.B. für wissenschaftliche Anwendungen
- Festpunkt-Darstellung kann nicht beiden Forderungen (Darstellung sehr großer UND sehr kleiner Zahlen) genügen

Gleitpunkt-(Gleitkomma/Fließkomma)-Darstellung

Fließkommadarstellung

$$x = (-1)^V \cdot M \cdot B^{\pm E}$$

- V ... Vorzeichen ($V=1$: negative Zahl, $V=0$: positive Zahl)
- M ... Mantisse: Für Genauigkeit entscheidend
- B ... Basis
- E ... Exponent: Für Bereich entscheidend

Englische Bezeichnung

“Floating-point” (Dezimalpunkt statt Komma)

Normalisierte Gleitpunkt-Darstellung

Beispiele

- $-0.0000123_{10} = -123 \cdot 10^{-7} = -12.3 \cdot 10^{-6}$
- $2016_{10} = 20.16 \cdot 10^2 = 0.2016 \cdot 10^4$
- ...

Mehrdeutigkeiten möglich

- Mögliche **Normalisierung** um Mehrdeutigkeiten zu vermeiden:

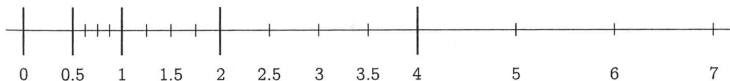
⇒ Mantisse hat genau eine Vorkommastelle, die ungleich 0 ist

- $-1.23 \cdot 10^5$
- $2.016 \cdot 10^3$

Normalisierte Gleitpunkt-Darstellung

Bsp: Gleitpunkt-Zahlensystem mit $B = 2$, $E_{min} = -1$, $E_{max} = 2$ und einer Mantissenlänge von $p = 3$

- Achtung: hier fehlen die negativen Zahlen!



Gleitpunkt-Zahlensystem mit $b = 2$, $p = 3$, $e_{min} = -1$, $e_{max} = 2$

- Lange vertikale Markierungen entsprechen Mantisse von 1.00
- $\Rightarrow +1.00 \cdot 2^{-1} = 0.5_{10} \quad +1.00 \cdot 2^0 = 1_{10} \quad +1.00 \cdot 2^1 = 2_{10} \dots$
- Bsp.: $+1.01 \cdot 2^{-1} = 0.101_2 = 0.5_{10} + 0.125_{10} = +0.625_{10}$
 - Bsp.: $+1.11 \cdot 2^1 = 11.1_2 = 2_{10} + 1_{10} + 0.5_{10} = +3.5_{10}$

Denormalisierte Gleitpunkt-Darstellung

Problem im vorherigen Beispiel: Lücke zwischen 0 und kleinsten positiven darstellbaren Zahl 0.5_{10}

- Grund: Normalisierungsbedingung ($m_0 \neq 0$)
- U.a. gilt nun die Eigenschaft $x = y \Leftrightarrow x - y = 0$ NICHT mehr

⇒ Bsp.: $y = 1.11 \cdot 2^{-1} = 0.875_{10}$; $x = 1.00 \cdot 2^0 = 1.00_{10}$

⇒ $x - y = 0.01 \cdot 2^{-1} = 0.125_{10}$ kann nicht als normalisierte Gleitpunktzahl dargestellt werden

⇒ Nächstliegende Zahl wäre Null. Durch eine Rundung auf Null kann ein folgenschwerer Laufzeit-Fehler passieren

⇒ Beispiel: **if** ($x \neq y$) **then** $z = 1 / (x / y)$;

Denormalisierte Gleitpunkt-Darstellung

Um Eigenschaft $x = y \Leftrightarrow x - y = 0$ zu garantieren, ...

- ...erweitert man die normalisierten Zahlen um genau jene Zahlen, die betragsmäßig zu klein sind, um normalisiert dargestellt werden zu können
- Diese durch Normalisierungsbedingung ($m_0 \neq 0$) weggefallenen Zahlen, werden zurückgewonnen in dem man $m_0 = 0$ für $E = E_{min}$ zulässt

⇒ Diese Zahlen nennt man **denormalisierte** Zahlen

⇒ Sie liegen sämtlich im Bereich $[-B^{E_{min}}, +B^{E_{min}}]$



Gleitpunkt-Zahlensystem inklusive denormalisierter Gleitpunktzahlen

Maschinengenauigkeit

Maß für den Rundungsfehler, der bei der Rechnung mit Gleitkommazahlen auftritt

- Bsp.: Zwei aufeinanderfolgende binäre Gleitkommazahlen sind
- $x = 1.000...00$
- $y = 1.000...01$
- Die Differenz beschreibt die relative Genauigkeit des Zahlensystems

⇒ “Maschinengenauigkeit” bzw. “Maschinen-Epsilon”

- $\epsilon = 2^{-p}$

Genauigkeit \Rightarrow nur endliche Genauigkeit möglich

Single-Precision: 32 bit

- VZ: 1 bit, Exponent: 8 bit, Mantisse: 23 Bit

\Rightarrow Maschinenepsilon $\epsilon = 2^{-23} \Rightarrow$ dezimal $\approx 1.2 \cdot 10^{-7}$

\Rightarrow Anzahl Dezimalstellen: ≈ 7

Double-Precision: 64 bit

- VZ: 1 bit, Exponent: 11 bit, Mantisse: 52 Bit

\Rightarrow Maschinenepsilon $\epsilon = 2^{-52} \Rightarrow$ dezimal $\approx 2.2 \cdot 10^{-16}$

\Rightarrow Anzahl Dezimalstellen: ≈ 16

Darstellung und Codierung im IEEE 754 Standard

Exponent in Exzessdarstellung

- Biased exponent e , $e = E + q (\Rightarrow E = e - q)$
- E ... rechnerisch wirkende Exponent, q ... Exzess/bias

$\Rightarrow q = 127$ bei 32 Bit (single-precision)

$\Rightarrow q = 1023$ bei 64 Bit (double-precision)

Spezielle Codierung für

- Null
- Unendlich
- Ungültige Zahl (NaN, not a number)

Beispiel: -5.375_{10} in Single Precision

Umwandlung in Dualsystem

- $-5.375_{10} = -101.011_2$

Normierung der Mantisse $\pm 1. \dots \cdot 2^E$

- $-1.01011_2 \cdot 2^2$
- Nur Bitfolge nach dem Komma wird gespeichert: 01011
- Restlichen Stellen werden mit 0-ern aufgefüllt

\Rightarrow 010110000000000000000000 (23 bit)

Exponent in Exzessdarstellung

- $e = E + q \Rightarrow 129 = 2 + 127$
- $e = 10000001$

Beispiel: -5.375_{10} in Single Precision

Ergebnis

- $-5.375_{10} = -101.011_2 = -1.01011_2 \cdot 2^2(\text{entspricht } E) = -1.01011_2 \cdot 2^{(129-127)}(\text{entspricht } e-q)$
- Vorzeichenbit = 1
- Exponent $e = 10000001$
- Mantisse $M = 01011000000000000000000$ (23 bit)

Rundungsfehleranalyse

Wie pflanzen sich Rundungsfehler fort?

- Typisches Beispiel *Taschenrechner*:
Ziehe k -mal die Wurzel aus 2 und quadriere anschließend das Ergebnis k -mal
- Erwartetes Ergebnis: 2
- Erzieltes Ergebnis für große k : 1

Addition dreier Maschinenzahlen

- Aufgabe: addiere $x = a + b + c$
- Maschinengenauigkeit ϵ
- Zerlegung der Gesamtrechnung:
 $e = (a +_M b)$ und $f = (e +_M c)$

Addition von Maschinenzahlen

Rundungsfehleranalyse

$$\begin{aligned} f &= e +_M c \\ &= (e + c)(1 + \epsilon_2) \\ &= ((a +_M b) + c)(1 + \epsilon_2) \\ &= ((a + b)(1 + \epsilon_1) + c)(1 + \epsilon_2) \\ &= a + b + c + (a + b)\epsilon_1 + (a + b + c)\epsilon_2 + (a + b)\epsilon_1\epsilon_2 \end{aligned}$$

Erste Näherung

In erster Näherung (d.h. unter Vernachlässigung des quadratischen Terms $(a + b)\epsilon_1\epsilon_2$) ergibt sich also:

$$f = a + b + c + (a + b)\epsilon_1 + (a + b + c)\epsilon_2$$

mit $|\epsilon_1|, |\epsilon_2| \leq \epsilon$

Addition von Maschinenzahlen

Für den relativen Fehler in erster Näherung ergibt sich damit

$$f_{rel}(x) = \frac{x - f}{x} \doteq \frac{(a + b + c) - (a + b + c + (a + b)\epsilon_1 + (a + b + c)\epsilon_2)}{a + b + c}$$

und daraus

$$= -\frac{a + b}{a + b + c}\epsilon_1 - \epsilon_2$$

und wegen $|\epsilon_1|, |\epsilon_2| \leq \epsilon$ die Abschätzung

$$|f_{rel}(x)| \doteq \left| \frac{a + b}{a + b + c}\epsilon_1 + \epsilon_2 \right| \leq \left(1 + \left| \frac{a + b}{a + b + c} \right| \right) \epsilon$$

Addition von Maschinenzahlen

1. Beobachtungen für den relativen Fehler

Der relative Fehler wird groß wenn:

- $|a + b| \gg |a + b + c|$, oder
- $a + b + c \approx 0$

2. Beobachtungen für den relativen Fehler

Andere Berechnungsreihenfolge liefert andere Faktoren:

- $x = (b + c) + a \rightarrow |f_{rel}| \leq \left(1 + \left|\frac{b+c}{a+b+c}\right|\right) \epsilon$
- $x = (a + c) + b \rightarrow |f_{rel}| \leq \left(1 + \left|\frac{a+c}{a+b+c}\right|\right) \epsilon$

⇒ Es wird also jeweils der bei der ersten Addition auftretende Fehler verstärkt!

Addition von Maschinenzahlen

Beispiel: $a = 1.11_2 \cdot 2^{-1}$, $b = -1.10_2 \cdot 2^{-1}$, $c = 1.10_2 \cdot 2^{-3}$

- Addition dieser Maschinenzahlen (dreistellige Mantisse inkl. führender 1) in der Reihenfolge: $(a + b) + c$

$$\begin{aligned} x &= (1.11_2 \cdot 2^{-1} +_M (-1.10)_2 \cdot 2^{-1}) +_M 1.10_2 \cdot 2^{-3} \\ &= 1.00_2 \cdot 2^{-3} +_M 1.10_2 \cdot 2^{-3} \\ &= 1.01_2 \cdot 2^{-2} \text{ (korrektes Ergebnis)} \end{aligned}$$

- Andere Reihenfolge: $x = a + (b + c)$

$$\begin{aligned} x^* &= 1.11_2 \cdot 2^{-1} +_M (-1.10_2 \cdot 2^{-1} +_M 1.10_2 \cdot 2^{-3}) \\ &= 1.11_2 \cdot 2^{-1} +_M (-1.00_2 \cdot 2^{-1}) \text{ (*)} \\ &= 1.10_2 \cdot 2^{-2} \end{aligned}$$

$$\text{mit relativem Fehler } \left| \frac{1.01_2 \cdot 2^{-2} - 1.10_2 \cdot 2^{-2}}{1.01_2 \cdot 2^{-2}} \right| = 20\%$$

⇒ **Also: Reihenfolge ist wichtig!!**

(*) Hinweis für alle Interessierten: In diesem Bsp. kann man davon ausgehen, dass im Rechenwerk für die Mantisse mehr als 3 Bit zur Verfügung stehen ("Guard Bit, Round Bit, Sticky Bit"). Das erlaubt eine genaue Berechnung der Differenz, danach wird nach der dritten signifikanten Ziffer abgeschnitten.

Addition von Maschinenzahlen

Besonders kritisch: Endergebnis nahe bei Null (“Auslöschung”)

- Beispiel: Differenz zwischen $a = 3/5$ und $b = 4/7$ bei fünfstelliger Mantisse (hier 5 Stellen inkl. führender 1)
- Exaktes Ergebnis: $a - b = 1/35 \approx 0.11101_2 \cdot 2^{-5}$
- Rundung: $a = (1.0011001\dots)_2 \cdot 2^{-1} \approx 1.0011_2 \cdot 2^{-1}$ und $b = (1.001001\dots)_2 \cdot 2^{-1} \approx 1.0010_2 \cdot 2^{-1}$
- Also ergibt Rechnung: $1.0011_2 \cdot 2^{-1} - 1.0010_2 \cdot 2^{-1} = 0.0001_2 \cdot 2^{-1} = 1.0000_2 \cdot 2^{-5} = 1/32$
- Relativer Fehler: $|\frac{x-f}{x}| = |(1/35 - 1/32)/(1/35)| = 9.4\%$
- Vergleich: die Maschinengenauigkeit bei fünfstelliger Mantisse (inkl. führender 1) liegt bei ca. 3.1%

Gerundete Eingangszahlen

Differenz $y = a - b$ für Eingangszahlen mit Rundungsfehlern

- $a \rightarrow a(1 + \epsilon_a)$, $b \rightarrow b(1 + \epsilon_b)$, Maschinengenauigkeit ϵ
- Relativer Fehler bei gerundeten Eingangszahlen:

$$\begin{aligned} f_{rel}(y) &= \frac{x - f}{x} = \frac{a - b - (a(1 + \epsilon_a) - b(1 + \epsilon_b)) \cdot \epsilon}{a - b} \\ &= -\frac{a}{a - b} \cdot \epsilon_a + \frac{b}{a - b} \cdot \epsilon_b - \epsilon \end{aligned}$$

- Eingabefehler werden also extrem verstärkt, falls sich a und b fast auslöschen!
- Das gilt aber nur für Eingangswerte mit Rundungsfehlern: Differenz mit exakten Zahlen ist ok!

Addition von Maschinenzahlen

Beispiel: Patriot-Scud Bug

- (Tragisches) Beispiel: sog. Patriot-Scud Software-Bug
- 25. Februar 1991 (Golf-Krieg): amerikanisches Raketenabwehrsystem Patriot verpasst Entdeckung einer irakischen Scud-Rakete, was zum Tod von mindestens 28 Menschen führt
- Grund: unpräzise Zeitkalkulation aufgrund von arithmetischen Rundungsfehlern

Addition von Maschinenzahlen

Beispiel: Patriot-Scud Bug

- Was war passiert?
- Interne Systemzeit misst in Zehntelsekunden. Diese Zeit wurde jeweils mit 10 malgenommen (Sekunden), und zwar über ein 24-Bit Festkommazahlenregister.
- Problem: $1/10$ lässt sich im Binärsystem darin nicht exakt darstellen: Rundungsfehler nach der 24. Nachkommastelle.
- Patriot-System lief bereits über 100 Stunden, d.h. dieser kleine Rundungsfehler entsprach bereits einer Zeitdifferenz von ca. 0.34 Sekunden. In dieser Zeit fliegt eine Scud-Rakete ca. einen halben Kilometer.

Addition von Maschinenzahlen

Beispiel: Patriot-Scud Bug

- Weiteres Problem: Bugfix war eigentlich bereits eingebaut, aber nicht überall konsistent
- Konsequenz: keine gegenseitige Auslöschung der Rundungsfehler
- Ergebnis: das Patriot-System vermutete die Scud-Rakete an einer falschen Stelle und konnte sie daher nicht entdecken.