

— Informatik I —  
Modul 2: Rechnerarithmetik (1)



Modul 2: Rechnerarithmetik (1)

- Zahlensysteme
- Zahlendarstellung



Rechnerarithmetik

- Rechnerarithmetik soll als Beispiel vorbereitet werden, wie größere Informationseinheiten verarbeitet werden.
- Hierzu werden zunächst die formalen Grundlagen erarbeitet.
- Dann werden Schaltnetze und -werke behandelt (Modul 3, 4).
- Danach werden Verfahren und Schaltungen zur Implementierung der vier Grundrechenarten in einem Rechner vorgestellt (Modul 5).
- Abschließend wird die Funktion einer arithmetisch logischen Einheit (ALU) eines Rechners besprochen (Modul 5).



Formale Grundlagen

- Menschen rechnen gewöhnlich im Dezimalzahlensystem.
  - Rechner rechnen gewöhnlich im Dualzahlensystem.
- Eine Konvertierung ist erforderlich
- Daneben werden weitere Zahlensysteme wie Oktalzahlensystem oder Hexadezimalzahlensystem (eigentlich: Sedezimal) zur kompakteren Darstellung der sehr langen Dualzahlen verwendet.
- Es ist notwendig, die Zusammenhänge und mathematischen Grundlagen dieser Zahlensysteme zu verstehen.



Zahlensysteme (1)

- Gängigste Form: Stellenwertsysteme
- Zahlendarstellung in Form einer Reihe von Ziffern  $z_i$ , wobei das Dezimalkomma (-punkt) rechts von  $z_0$  plaziert sei:

$$z_n z_{n-1} \dots z_1 z_0, z_{-1} z_{-2} \dots z_{-m} \quad \text{z.B. } 1234,567$$

- Jeder Position  $i$  der Ziffernreihe ist ein Stellenwert zugeordnet, der eine Potenz  $b^i$  der Basis  $b$  des Zahlensystems ist.
- Der Wert  $X_b$  der Zahl ergibt sich dann als Summe der Werte aller Einzelstellen  $z_i b^i$ :

$$X_b = z_n b^n + z_{n-1} b^{n-1} + \dots + z_1 b + z_0 + z_{-1} b^{-1} + \dots + z_{-m} b^{-m} = \sum_{i=-m}^n z_i b^i$$



Zahlensysteme (2)

- Interessante Zahlensysteme in der Informatik

b	Zahlensystem	Zahlenbezeichnung
2	Dualsystem	Dualzahl
8	Oktalsystem	Oktalzahl
10	Dezimalsystem	Dezimalzahl
16	Hexadezimalsystem (Sedezimalsystem)	Hexadezimalzahl (Sedezimalzahl)

- Hexadezimalsystem: Die „Ziffern“ 10 bis 15 werden mit den Buchstaben A bis F dargestellt.
- Dualsystem: Wichtigstes Zahlensystem im Rechner
- Oktal- und Hexadezimalsystem: Leicht ins Dualsystem umwandelbar, besser zu verstehen als lange 0-1-Kolonnen.



## Der Euklidische Algorithmus

- Umwandlung vom Dezimalsystem in ein System zur Basis  $b$
- 1. Methode: Euklidischer Algorithmus:**  

$$Z = z_n 10^n + z_{n-1} 10^{n-1} + \dots + z_1 10 + z_0 + z_{-1} 10^{-1} + \dots + z_{-m} 10^{-m}$$

$$= y_p b^p + y_{p-1} b^{p-1} + \dots + y_1 b + y_0 + y_{-1} b^{-1} + \dots + y_{-q} b^{-q}$$
 Die Ziffern werden sukzessive, beginnend mit der höchstwertigen Ziffer, berechnet.
- 1. Schritt:** Berechne  $p$  gemäß der Ungleichung  $b^p \leq Z < b^{p+1}$  (setze  $i = p$ )
- 2. Schritt:** Ermittle  $y_i$  und den Rest  $R_i$  durch Division von  $Z_i$  durch  $b^i$ :  $y_i = Z_i \text{ div } b^i$ ;  $R_i = Z_i \text{ mod } b^i$ ;
- 3. Schritt:** Wiederhole 2. Schritt für  $i = p-1, \dots$  und ersetze dabei nach jedem Schritt  $Z$  durch  $R_i$ , bis  $R_i = 0$  oder bis  $b^i$  (und damit der Umrechnungsfehler) gering genug ist.



## Beispiel

Umwandlung von  $15741,233_{10}$  ins Hexadezimalsystem:

- Schritt:  $16^3 \leq 15741,233 < 16^4 \rightarrow$  höchste Potenz  $16^3$
- Schritt:  $15741,233 : 16^3 = 3$  Rest 3453,233
- Schritt:  $3453,233 : 16^2 = D$  Rest 125,233
- Schritt:  $125,233 : 16 = 7$  Rest 13,233
- Schritt:  $13,233 : 1 = D$  Rest 0,233
- Schritt:  $0,233 : 16^{-1} = 3$  Rest 0,0455
- Schritt:  $0,0455 : 16^{-2} = B$  Rest 0,00253
- Schritt:  $0,00253 : 16^{-3} = A$  Rest 0,000088593
- Schritt:  $0,000088593 : 16^{-4} = 5$  Rest 0,000012299 ( $\rightarrow$  Fehler)

$\rightarrow 15741,233_{10} \approx 3D7D,3BA5_{16}$



## Horner Schema

- Umwandlung vom Dezimalsystem in ein Zahlensystem zur Basis  $b$
- 2. Methode: Abwandlung des Horner Schemas**
- Hierbei müssen der ganzzahlige und der gebrochene Anteil getrennt betrachtet werden:
- Umwandlung des ganzzahligen Anteils:
- Eine ganze Zahl  $X_b = \sum_{i=0}^n z_i b^i$  kann durch fortgesetztes Ausklammern auch in folgender Form geschrieben werden:  

$$X_b = (((\dots((y_n b + y_{n-1}) b + y_{n-2}) b \dots) b + y_1) b + y_0$$



## Horner Schema: Beispiel

- Die gegebene Dezimalzahl wird sukzessive durch die Basis  $b$  dividiert.
  - Die jeweiligen ganzzahligen Reste ergeben die Ziffern der Zahl  $X_b$  in der Reihenfolge von der niedrigstwertigen zur höchstwertigen Stelle.  
 Wandle  $15741_{10}$  ins Hexadezimalsystem um:  

$$15741_{10} : 16 = 983 \text{ Rest } 13 \quad (D_{16})$$

$$983_{10} : 16 = 61 \text{ Rest } 7 \quad (7_{16})$$

$$61_{10} : 16 = 3 \text{ Rest } 13 \quad (D_{16})$$

$$3_{10} : 16 = 0 \text{ Rest } 3 \quad (3_{16})$$
- $\rightarrow 15741_{10} = 3D7D_{16}$



## Umwandlung: Basis $b \rightarrow$ Dezimalsystem

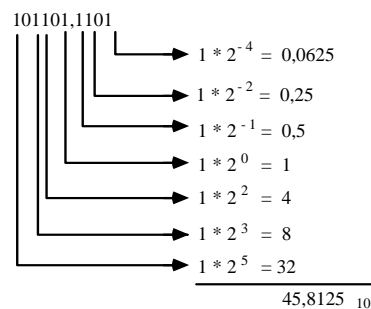
- Die Werte der einzelnen Stellen der umzuwandelnden Zahl werden in dem Zahlensystem, in das umgewandelt werden soll, dargestellt und nach der Stellenwertgleichung aufsummiert.
- Der Wert  $X_b$  der Zahl ergibt sich dann als Summe der Werte aller Einzelstellen  $z_i b^i$ :

$$X_b = z_n b^n + z_{n-1} b^{n-1} + \dots + z_1 b + z_0 + z_{-1} b^{-1} + \dots + z_{-m} b^{-m} = \sum_{i=-m}^n z_i b^i$$



## Beispiel

Konvertiere  $101101,1101_2$  ins Dezimalsystem



## Umwandlung beliebiger Stellenwertsysteme

- Man wandelt die Zahl ins Dezimalsystem um und führt danach mit Methode 1 oder 2 die Wandlung ins Zielsystem durch.
  - Spezialfall:
    - Ist eine Basis eine Potenz der anderen Basis, können einfach mehrere Stellen zu einer Ziffer zusammengefasst werden oder eine Stelle kann durch eine Folge von Ziffern ersetzt werden.
  - Wandlung von  $0110100,110101_2$  ins Hexadezimalsystem
  - $2^4 = 16 \Rightarrow 4$  Dualstellen  $\rightarrow 1$  Hexadezimalstelle
- dual
- 0110100,110101
- ↓
- 00110100,11010100
- Ergänzen von Nullen zur  
Auffüllung auf Vierergruppen
- hexadezimal    3 4 , D 4



## Modul 2: Rechnerarithmetik (1)

- Zahlensysteme
- Zahlendarstellung



## Darstellung negativer Zahlen

- Für die Darstellung negativer Zahlen in Rechnern werden vier verschiedene Formate benutzt:
- Darstellung mit Betrag und Vorzeichen
- Stellenkomplement-Darstellung (Einerkomplement-Darstellung)
- Zweierkomplement-Darstellung
- Offset-Dual-Darstellung / Exzeß-Darstellung



## Darstellung mit Betrag und Vorzeichen

- Eine Stelle wird als Vorzeichenbit benutzt.
- Ist das am weitesten links stehende Bit (MSB, most significant bit):
  - MSB = 0  $\rightarrow$  positive Zahl
  - MSB = 1  $\rightarrow$  negative Zahl

Beispiel:

$$\begin{aligned} 0001\ 0010 &= +18 \\ 1001\ 0010 &= -18 \end{aligned}$$

Nachteile:

- Bei Addition und Subtraktion müssen die Vorzeichen der Operanden gesondert betrachtet werden.
- Es gibt zwei Repräsentationen der Zahl 0 (mit positivem und mit negativem Vorzeichen)



## Stellenkomplement / Einerkomplement

- Stellenkomplement der entsprechenden positiven Zahl.
- Um eine Zahl zu negieren, wird jedes Bit der Zahl komplementiert.
- Dies entspricht dem Einerkomplement:
  - Komplementbildung  $z_{ek} = (2^n - 1) - z$

Bsp:  $4 = 0100_2 \rightarrow -4 = 1011_{ek}$

$$-4 = 2^4 - 1 - 4 = 11_{10} = 1011_2$$

- Negative Zahlen sind wiederum durch ein gesetztes Bit in der ersten Stelle charakterisiert.
- Vorteil gegenüber der Darstellung mit Vorzeichenbit:
  - Erste Stelle bei Addition und Subtraktion muß nicht gesondert betrachtet werden.
  - Aber: Es gibt weiterhin zwei Darstellungen der Null



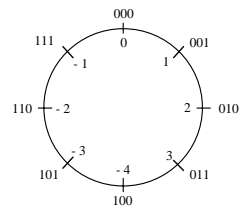
## Zweierkomplement-Darstellung (1)

- Man addiert nach der Stellenkomplementierung noch eine 1
- Man erhält so das Zweierkomplement:  $z_{zk} = 2^n - z$

$$\begin{aligned} 0 \dots 0 &\rightarrow \text{Einerkomplement } 1 \dots 1 \\ &\rightarrow \text{Zweierkomplement } 0 \dots 0 \end{aligned}$$

Nachteil:

- Unsymmetrischer Zahlenbereich. Die kleinste negative Zahl ist betragsmäßig um 1 größer als die größte positive Zahl



- 3-Bit-Zweierkomplementzahlen (Beispiel):



## Zweierkomplement-Darstellung (2)

- Alle anderen negativen Zahlen werden um 1 verschoben, das MSB bleibt aber gleich 1.
- Aus der ersten Stelle kann das Vorzeichen der Zahl abgelesen werden
- Aus dieser Konstruktion ergibt sich der Stellenwert des MSB einer Zweierkomplementzahl mit  $n+1$  Bit zu  $-2^n$ :

$z_n z_{n-1} \dots z_0$  hat den Wert:

$$Z = -z_n \cdot 2^n + z_{n-1} \cdot 2^{n-1} + \dots + z_0$$



## Beispiel

Die Zahl  $-77_{10}$  soll mit 8 Bit dargestellt werden

$$77_{10} = 0100\ 1101_2$$

Mit Vorzeichenbit :  $-77 = 1100\ 1101_2$

Einerkomplement :  $-77 = 1011\ 0010_2$

Zweierkomplement :  $-77 = 1011\ 0011_2$

Bitweise komplementieren

Addition von 1



## Offset-Dual- (Exzeß-)Darstellung

- Wird hauptsächlich bei der Exponenten-Darstellung von Gleitkommazahlen benutzt.
- Die Darstellung einer Zahl erfolgt in Form ihrer **Charakteristik**.
- Der gesamte Zahlenbereich wird durch Addition einer Konstanten (Exzeß, Offset) so nach oben verschoben, daß die kleinste (negative) Zahl die Darstellung  $0\dots 0$  erhält.
- Bei  $n$  Stellen ist der Offset  $2^{n-1}$
- Der Zahlenbereich ist hier auch asymmetrisch.



## Zusammenfassung der Möglichkeiten

Dezimalzahl	Darstellung mit			Charakteristik
	Betrag + Vorzeichen	Einerkomplement	Zweierkomplement	
-4	---	---	100	000
-3	111	100	101	001
-2	110	101	110	010
-1	101	110	111	011
0	100,000	111,000	000	100
1	001	001	001	101
2	010	010	010	110
3	011	011	011	111



## Fest- und Gleitkommazahlen

- Zahlendarstellung auf dem Papier:
  - Ziffern 0 .. 9
  - Vorzeichen + -
  - Komma (Punkt) , .
- Zahlendarstellung im Rechner:
  - Binärziffern 0, 1
- spezielle Vereinbarungen für die Darstellung von Vorzeichen und Komma/Punkt im Rechner sind erforderlich
- Darstellung des Vorzeichens:
  - Wurde im vorigen Abschnitt behandelt
- Darstellung des Kommas mit zwei Möglichkeiten:
  - Festkommadarstellung
  - Gleitkommadarstellung



## Festkomma-Zahlen (1)

- Vereinbarung:
  - Das Komma sitzt innerhalb des Maschinenwortes, das eine Dualzahl enthalten soll, an einer festen Stelle.
- Meist setzt man das Komma hinter die letzte Stelle.
- Andere Zahlen können durch entsprechende Maßstabsfaktoren in die gewählte Darstellungsform überführt werden.
- Negative Zahlen:
  - Meist Zweierkomplement-Darstellung.
- Festkomma-Darstellungen werden heute hardwareseitig nicht mehr verwendet, jedoch bei Ein- oder Ausgabe!



## Festkomma-Zahlen (2)

- Datentyp "integer" (Ganzzahlen) ist ein spezielles Festkommaformat.
- Manche Programmiersprachen erlauben die Definition von Ganzzahlen unterschiedlicher Länge.
- Beispiel "C": "short int", "int", "long int", "unsigned"

Datentyp	DEC-VAX (einer der Urnahmen)		IBM-PC, Apple Macintosh	
	Anzahl der Bits	Zahlenbereich	Anzahl der Bits	Zahlenbereich
short int	16	$-2^{15} .. 2^{15}-1$	16	$-2^{15} .. 2^{15}-1$
int	32	$-2^{31} .. 2^{31}-1$	16	$-2^{15} .. 2^{15}-1$
long int	32	$-2^{31} .. 2^{31}-1$	32	$-2^{31} .. 2^{31}-1$



## Gleitkomma-Darstellung (1)

- Zur Darstellung von Zahlen, die betragsmäßig sehr groß oder sehr klein sind, verwendet man die **Gleitkomma-Darstellung**.
- Sie entspricht einer halblogarithmischen Form

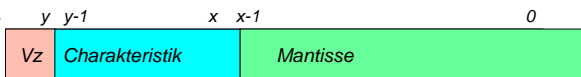
$$X = \pm \text{Mantisse} \cdot b^{\text{Exponent}}$$

- Die Basis  $b$  ist für eine bestimmte Gleitkomma-Darstellung fest (meist 2 oder 16) und braucht damit nicht mehr explizit repräsentiert zu werden.
- Gleitkommazahlen werden meist *nicht* im Zweierkomplement, sondern mit **Betrag** und **Vorzeichen** dargestellt.



## Gleitkomma-Darstellung (2)

- Bei der **Mantisse** ist die Lage des Kommas wieder durch Vereinbarung festgelegt (meist links vom MSB).
- Der **Exponent** ist eine ganze Zahl, die in Form ihrer Charakteristik dargestellt wird.
- Für die Charakteristik und die Mantisse wird im Rechner ein **festes Anzahl von Speicherstellen** festgelegt.
- Die Länge der Charakteristik  $y-x$  bestimmt die Größe des Zahlenbereichs.
- Die Länge der Mantisse  $x$  legt die Genauigkeit der Darstellung fest.



$$\text{Dezimalzahl} = (-1)^{Vz} * (0, \text{Mantisse}) * b^{\text{Exponent}}$$

$$\text{Exponent} = \text{Charakteristik} - b^{(y-1)-x}$$



## Normalisierung

- Legt man für die Zahl 0 ein spezielles Bitmuster fest, ist die erste Stelle der Mantisse in normalisierter Form immer gleich 1.
- Die erste Stelle der Mantisse braucht im Maschinenformat gar nicht erst dargestellt zu werden, d.h. (0,1 . . . .)
- **Man spart ein Bit bei der Speicherung oder gewinnt bei gleichem Speicherbedarf ein Bit an Genauigkeit.**
- Bei arithmetischen Operationen und bei der Konversion in andere Darstellungen darf diese Stelle natürlich nicht vergessen werden.

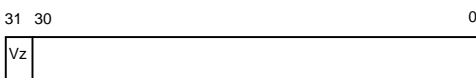


## Beispiel (1)

3 verschiedene Maschinenformate mit je 32 Bit und  $b = 2$ .

Die Zahl  $7135_{10}$  wird in jedem dieser Formate dargestellt.

a) Festkomma-Darstellung mit Zweierkomplement

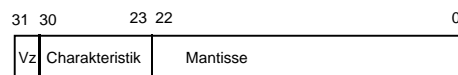


$$0\ 000\ 0000\ 0000\ 0000\ 0001\ 1011\ 1101\ 1111_2 = 0000\ 1BDF_{16}$$



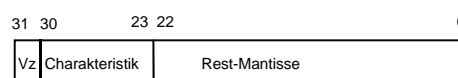
## Beispiel (2)

b) Gleitkomma-Darstellung, normalisiert:



$$0\ 100\ 0110\ 1\ 101\ 1111\ 0111\ 1100\ 0000\ 0000_2 = 46EF\ 7C00_{16}$$

c) Gleitkomma-Darstellung, normalisiert, erste "1" implizit:



$$0\ 100\ 0110\ 1\ 101\ 1110\ 1111\ 1000\ 0000\ 0000_2 = 46DE\ F800_{16}$$





## Ungenauigkeiten

- Die Differenz zwischen zwei aufeinanderfolgenden Zahlen wächst bei Gleitkomma-Zahlen exponentiell mit der Größe der Zahlen, während sie bei Festkomma-Zahlen konstant ist.
- Bei der Darstellung großer Zahlen ergibt sich damit auch eine hohe Ungenauigkeit.
- Die Gesetzmäßigkeiten, die für reelle Zahlen gelten, werden für Maschinendarstellungen verletzt!

Dies gilt insbesondere auch wenn diese Zahlen in einer höheren Programmiersprache oft `real` heißen.



## Beispiel

- Das Assoziativgesetz  $(x + y) + z = x + (y + z)$  gilt selbst dann nicht unbedingt, wenn kein overflow oder underflow auftritt.

z.B.:  $x = 1; y = z = \text{smallreal}/2$

$$\begin{aligned}(x + y) + z &= (1 + \text{smallreal}/2) + \text{smallreal}/2 \\ &= 1 + \text{smallreal}/2 \\ &= 1\end{aligned}$$

$$\begin{aligned}x + (y + z) &= 1 + (\text{smallreal}/2 + \text{smallreal}/2) \\ &= 1 + \text{smallreal} \\ &\neq 1\end{aligned}$$

Hinweis: `smallreal` ist die kleinste Zahl, die man zu 1 addieren kann, um einen von 1 verschiedenen Wert zu erhalten!



## Problematik unterschiedlicher Definitionen

- Es existieren beliebig viele Möglichkeiten, selbst mit einer festen Wortbreite unterschiedliche Gleitkommaformate zu definieren (unterschiedliche Basis  $b$ , Darstellung der Null, Anzahl der Stellen für Charakteristik und Mantisse).
- Es existierten (bis Mitte der 80er Jahre) viele verschiedene, herstellerabhängige Formate
- Man konnte mit dem gleichen Programm auf unterschiedlichen Rechnern sehr unterschiedliche Ergebnisse erhalten!
- Normierung erforderlich



## Normierung (IEEE-Standard)

- IEEE-P 754-Floating-Point-Standard
- In vielen Programmiersprachen lassen sich Gleitkomma-Zahlen mit verschiedener Genauigkeit darstellen
  - z.B. in C: `float`, `double`, `long double`
- Der IEEE Standard definiert mehrere Darstellungsformen
  - IEEE single: 32 Bit
  - IEEE double: 64 Bit
  - IEEE extended: 80 Bit

