3 Zahlensysteme und ASCII-Zeichen

In diesem Kapitel wollen wir uns damit befassen, welche Möglichkeiten es gibt, Zahlen in einem Rechner darzustellen. Dabei unterscheidet man hauptsächlich zwischen ganzen und reellen Zahlen, die man beim Programmieren als integer bzw. float bezeichnet. Wir werden die in Bild 3-1 aufgeführten Darstellungsarten nacheinander kennen lernen und uns anschließend die Darstellung von Zeichen ansehen.

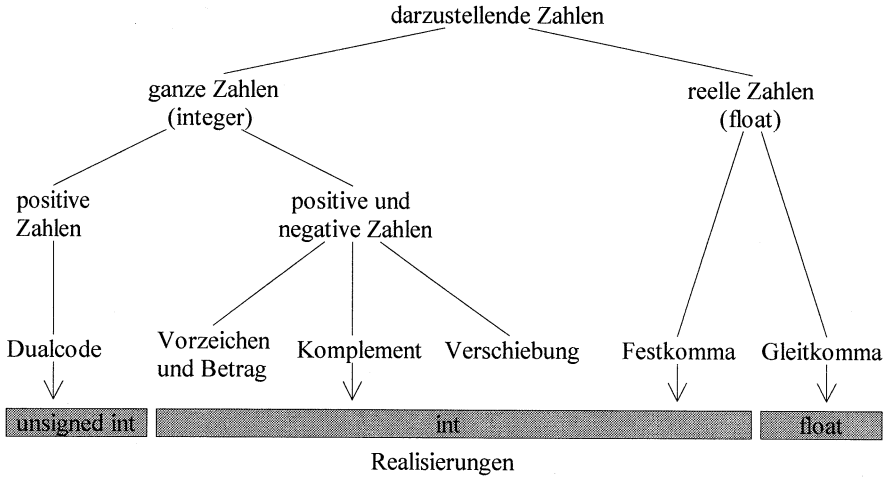

Bild 3-1: Überblick über die Zahlendarstellungsarten

Zur einfacheren Einführung des Dualcodes betrachten wir zunächst das uns vertraute Dezimalsystem.

3.1 Dezimalsystem

Eine wichtige Grundvoraussetzung für die Entwicklung der Datentechnik bildete die Festlegung auf ein sinnvolles Zahlensystem. Von den verschiedenen Zahlensystemen, wie zum Beispiel das Fünfer-System der Chinesen, das Zwanziger-System der Mayas und das Sechziger-System der Sumerer, hat sich nur das *Dezimalsystem* durchgesetzt, weil man beim Zählen und Rechnen die zehn Finger zu Hilfe nehmen kann. Das war der maßgebende Vorteil für die allgemeine Verbreitung.

Das Dezimalsystem gehört zu den so genannten *Stellenwertsystemen*, d. h. jeder Stelle einer Zahl ist eine Zehnerpotenz als Vervielfachungsfaktor zugeordnet.

Tausender	Hunderter	Zehner	Einer	Zehntel	Hundertstel
10^3	10^2	10^1	10^0	10^{-1}	10^{-2}
1	**2**	**3**	**4**	**5**	**6**

Tabelle 3-1: Aufbau des Dezimalsystems am Beispiel der Zahl $(1234,56)_{10}$

Mathematisch wird das Dezimalsystem durch die Formel

$$z = \sum_{i=-m}^{n} z_i \, b^i = \sum_{i=-m}^{n} z_i \, 10^i \qquad \text{mit } z_i \in \{0,1,2,...9\};\ m, n \text{ ganze Zahlen}$$

definiert. Auf das obige Beispiel angewendet bedeutet das:

$$(1234,56)_{10} = 1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 + 5 \cdot 10^{-1} + 6 \cdot 10^{-2}$$

Dabei bezeichnet man den Zeichenvorrat, den man in diesem System benutzen darf, als *Alphabet*. Beim Dezimalsystem sind es die Ziffern 0 bis 9.

3.2 Dualsystem

Im Gegensatz zum Dezimalsystem mit den zehn Ziffern 0 bis 9 erwies es sich in der Datentechnik als sinnvoll, ein Zahlensystem mit nur zwei verschiedenen Zuständen festzulegen, z. B. mit den beiden Ziffern 0 und 1. Technisch kann man dieses binäre Zahlensystem, meist *Binärcode* genannt, einfach und sehr sicher realisieren.

Unter einem *Code* versteht man allgemein die eindeutige Zuordnung (Codierung) von einem Zeichenvorrat zu einem anderen. So ordnet zum Beispiel der ASCII-Code dem „A" den binären Wert 0100 0001 zu.

Die Anzahl der Binärstellen werden in *bit* (bit = binary digit, binäre Stelle) angegeben. Eine 4 stellige Binärzahl (z. B. 0110) hat also 4 bit.

8 Binärstellen fasst man oft als 1 *Byte* zusammen: also 8 bit = 1 Byte oder kurz 1 B.

Beim *Dualsystem* oder *Dualcode* hat man definiert, dass jeder Stelle als Vervielfachungsfaktor eine Zweierpotenz zugeordnet wird (analog zum Dezimalsystem). Oder mathematisch ausgedrückt:

$$z = \sum_{i=-m}^{n} z_i 2^i \qquad \text{mit } z_i \in \{0,1\};\ m, n \text{ ganze Zahlen}$$

Beispiel: Die Dualzahl 1011,01 bedeutet also:

2^3	2^2	2^1	2^0	2^{-1}	2^{-2}
1	**0**	**1**	**1**	**0**	**1**

$$(1011{,}01)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = (11{,}25)_{10}$$

In der folgenden Tabelle sind die wichtigsten *Zweierpotenzen* zusammengestellt.

2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512

2^{10}	1024
2^{11}	2048
2^{12}	4096
2^{13}	8192
2^{14}	16384
2^{15}	32768
2^{16}	65536
2^{17}	131072
2^{18}	262144
2^{19}	524288

2^{-1}	0,5
2^{-2}	0,25
2^{-3}	0,125
2^{-4}	0,0625
2^{-5}	0,03125
2^{-6}	0,015625
2^{-7}	0,0078125
2^{-8}	0,00390625
2^{-9}	0,001953125
2^{-10}	0,0009765625

Tabelle 3-2: Zweierpotenzen mit ihren Dezimalwerten

Dezimalzahl		Dualzahl			
10^1	10^0	2^3	2^2	2^1	2^0
	0				0
	1				1
	2			1	0
	3			1	1
	4		1	0	0
	5		1	0	1
	6		1	1	0
	7		1	1	1

Dezimalzahl		Dualzahl			
10^1	10^0	2^3	2^2	2^1	2^0
	8	1	0	0	0
	9	1	0	0	1
1	0	1	0	1	0
1	1	1	0	1	1
1	2	1	1	0	0
1	3	1	1	0	1
1	4	1	1	1	0
1	5	1	1	1	1

Tabelle 3-3: Einige Dezimalzahlen mit den dazugehörigen Dualzahlen

Die letzte Stelle der Dualzahlen ändert sich jeweils von einer Dezimalzahl zur nächsten. Sie gibt also an, ob die Zahl gerade oder ungerade ist. Dagegen ändert sich die vorletzte Stelle der Dualzahlen nur alle zwei Dezimalzahlen, die drittletzte Stelle nur alle vier Dezimalzahlen, die viertletzte Stelle nur alle acht Dezimalzahlen usw.

Betrachten wir Dualzahlen mit einer festen Länge n. Die Stelle, die am weitesten links steht, nennt man die *höchstwertige Stelle* (engl.: most significant bit, abgekürzt

MSB). Entsprechend heißt die Stelle, die ganz rechts steht, die *niederwertigste Stelle* (engl.: least significant bit, abgekürzt: *LSB*).

Die höchstwertige Stelle n gibt an, ob von dem Zahlenbereich, der sich mit dieser Stellenanzahl darstellen lässt, die obere oder untere Hälfte bezeichnet wird:

- 0 an Stelle n (MSB) kennzeichnet die Hälfte mit den kleineren Werten,
- 1 an Stelle n (MSB) kennzeichnet die Hälfte mit den höheren Werten.

Entsprechend teilt die Stelle n-1 jede Hälfte wiederum in zwei Hälften auf. Das setzt sich so weiter fort bis zur Stelle 1:

- 0 an Stelle 1 (LSB) kennzeichnet eine gerade Zahl,
- 1 an Stelle 1 (LSB) kennzeichnet eine ungerade Zahl.

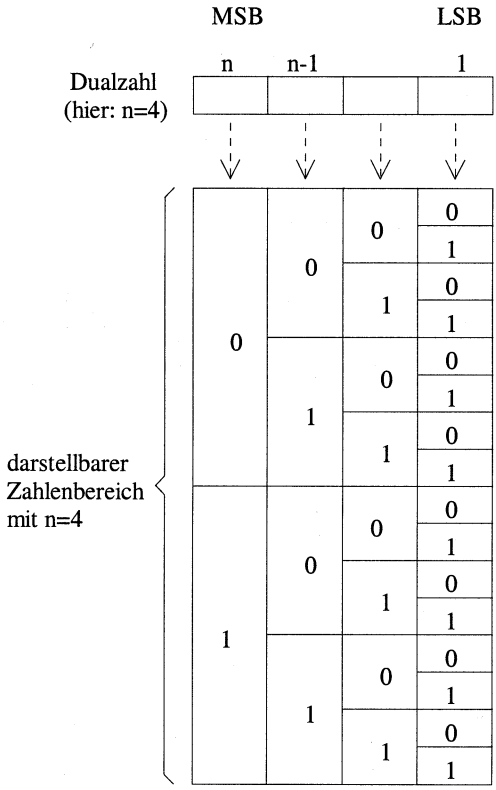

Bild 3-2: Bedeutung der Stellen einer Dualzahl

Für das Arbeiten mit dem Dualsystem brauchen wir noch folgende Regeln:

Bei einer Dualzahl mit einer Länge von n Stellen gilt:
- Es können 2^n verschiedene Zahlen dargestellt werden!
- Die größte darstellbare Zahl beträgt $2^n - 1$!

Beispiel: Im Bild 3-2 ist n = 4. Also gilt:
$2^4 = 16$ verschiedene Zahlen sind möglich.
Die größte Zahl ist $2^4 - 1 = 16 - 1 = 15$.

Eine Addition erfolgt im Dualsystem prinzipiell wie im Dezimalsystem. Bei zwei Summanden sind folgende Kombinationen möglich:

0 + 0 = 0,
0 + 1 = 1,
1 + 0 = 1,
1 + 1 = 0 und ein Übertrag in die nächst höhere Stelle.

3.2.1 Umwandlung von Dual- in Dezimalzahlen

Zur Umwandlung (auch *Konvertierung* genannt) von Dualzahlen in Dezimalzahlen muss man die Zweierpotenzen kennen, oder man verwendet die Tabelle 3-2.

Beispiel: $(100111011{,}01)_2 \rightarrow (315{,}25)_{10}$

Man trägt die Dualzahl in die Tabelle ein und addiert dann alle Zweierpotenzen, in deren Spalte eine 1 steht:

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
1024	512	256	128	64	32	16	8	4	2	1	0,5	0,25	0,125
		1	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

$$2^8 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 + 2^{-2} = 256 + 32 + 16 + 8 + 2 + 1 + 0{,}25 = 315{,}25$$

3.2.2 Umwandlung von Dezimal- in Dualzahlen

Will man eine Dezimalzahl in eine Dualzahl umwandeln, dann muss man die Zweierpotenzen kennen. Wenn man z. B. $(315{,}25)_{10}$ in eine Dualzahl konvertieren will, dann muss man die höchste Zweierpotenz suchen, die in 315,25 enthalten ist, also $256 = 2^8$. Für den Rest bestimmt man wieder die höchste Zweierpotenz usw.

```
315,25-256=59,25
          59,25-32=27,25
                   27,25-16=11,25
                            11,25-8=3,25
                                    3,25-2=1,25
                                            1,25-1=0,25
                                                    0,25-0,25=0
1·2^8  +  1·2^5  +  1·2^4  +  1·2^3 + 1·2^1  + 1·2^0 + 1·2^-2
= (100111011,01)_2
```

Bei ganzen Dezimalzahlen kann man das Verfahren vereinfachen. Man teilt die Dezimalzahl fortlaufend durch 2 und vermerkt jeweils, ob ein Rest auftritt oder nicht. Die Reste von unten nach oben gelesen, ergeben die Dualzahl.

Beispiel:

315 : 2 = 157 Rest 1
157 : 2 = 78 Rest 1
78 : 2 = 39 Rest 0
39 : 2 = 19 Rest 1
19 : 2 = 9 Rest 1
9 : 2 = 4 Rest 1
4 : 2 = 2 Rest 0
2 : 2 = 1 Rest 0
1 : 2 = 0 Rest 1 → Dualzahl: 100111011

Erklärung: Ein Teilen durch 2 entspricht im Dualcode ein Verschieben der Zahl um eine Stelle nach rechts: aus $(100)_2 = (4)_{10}$ wird $(10)_2 = (2)_{10}$.
Entsteht beim Dividieren durch 2 ein Rest, dann war die Zahl ungerade und das bedeutet im Dualcode, dass die letzte Stelle 1 sein muss.
Die letzte Division „1 : 2" ergibt immer einen Rest, der dann die höchste Stelle des Dualcodes bildet.

3.3 Darstellung von negativen Zahlen

Mit dem Dualcode kann man alle positiven ganzen Zahlen einschließlich der Null darstellen. Wie kann man aber mit dem Dualcode negative Zahlen erzeugen?

Im Dezimalsystem werden neben den zehn Ziffern 0 bis 9 auch noch die beiden Vorzeichen + und - sowie das Komma verwendet. Bei der Darstellung von Zahlen in Rechnern wollen wir aber an dem Binärcode festhalten: Es stehen uns also nur die beiden Zeichen 0 und 1 zur Verfügung. Die Vorzeichen und das Komma dürfen nicht als neue Zeichen eingeführt werden.

3.3.1 Zahlendarstellung mit Vorzeichen und Betrag

Analog zu dem Dezimalsystem bietet sich zunächst das Verfahren *Vorzeichen und Betrag* an. Dabei gibt das Bit, das am weitesten links steht, das Vorzeichen an. Da führende Nullen oder ein Pluszeichen eine Dezimalzahl nicht verändern, ergibt sich fast zwangsläufig folgende Definition für das Vorzeichenbit:

- 0 → positive Zahl
- 1 → negative Zahl.

Beispiel: $(+\,91)_{10} = (\mathbf{0}\ 101\ 1011)_2$
$(-\,91)_{10} = (\mathbf{1}\ 101\ 1011)_2$

Das linke Bit erhält also die Bedeutung eines Vorzeichens, ohne dass wir ein zusätzliches Zeichen im Binärsystem aufnehmen müssen.

	Vorzeichen	Betrag	
+5	0	000 0101	positive Zahlen
+4	0	000 0100	
+3	0	000 0011	
+2	0	000 0010	
+1	0	000 0001	
+0	0	000 0000	
-0	1	000 0000	negative Zahlen
-1	1	000 0001	
-2	1	000 0010	
-3	1	000 0011	
-4	1	000 0100	
-5	1	000 0101	

Tabelle 3-4: Negative Zahlen gebildet durch Vorzeichen und Betrag

Bei Programmbeginn kann man eine Zahl als unsigned bzw. signed, also ohne bzw. mit Vorzeichen, deklarieren. Dadurch ändert sich die Bedeutung des höchsten Bits, was wieder Auswirkungen auf das Rechenwerk hat:

- Bei der Addition von Zahlen ohne Vorzeichen (unsigned) werden alle Stellen addiert unabhängig davon, ob das höchste Bit jeweils 0 oder 1 ist.
- Bei Zahlen mit Vorzeichen (signed) bedeutet eine 1 als höchstes Bit eine Subtraktion, also eine von der Addition abweichende Operation.
- Bei der Subtraktion muss die betragsmäßig kleinere Zahl von der größeren abgezogen werden. Unter Umständen muss man dazu die Operanden vertauschen und das beim Vorzeichen des Ergebnisses berücksichtigen.
  Beispiel: 392 – 528 = - (528 – 392) = -136

Diese Fallunterscheidungen müssen bei jeder Addition bzw. Subtraktion durchgeführt werden und verzögern deshalb diese Operationen. Besser wäre ein Verfahren, bei dem das Rechenwerk die Zahlen sofort verabeiten kann, egal ob sie ein Vorzeichen haben oder nicht und ob sie addiert oder subtrahiert werden sollen.

Zusammenfassung: Vorzeichen und Betrag

Zahlenbereich (für n = 8): von 0111 1111 also + $(2^7 - 1)$ = +127
bis 1111 1111 also - $(2^7 - 1)$ = -127.

Vorteile: Negative Zahlen kann man direkt darstellen und muss sie nicht erst errechnen (wie bei der Komplement-Darstellung).

Nachteile: 1) Für die Null gibt es zwei verschiedene Darstellungen: 0000 0000 und 1000 0000. Es bedeutet einen gewissen Zusatzaufwand, dies als Gleichheit zu erkennen.

2) Bei der Subtraktion müssen die Operanden vertauscht werden, wenn der Betrag des Subtrahenden größer ist als der des Minuenden. Auch dafür ist eine zusätzliche Logik erforderlich.
3) Das Rechenwerk muss Zahlen mit und ohne Vorzeichen unterschiedlich verarbeiten.
4) Das Rechenwerk muss addieren und subtrahieren können.

3.3.2 Negative Zahlen durch Komplement-Darstellung

Eine andere Methode, um negative Zahlen zu bilden, basiert auf der Komplement-Bildung. Dazu sind einige Vorbetrachtungen notwendig.
In der Rechnertechnik wird die Subtraktion überwiegend auf die Addition zurückgeführt, indem man, anstatt den Subtrahenden abzuziehen, eine geeignete Zahl addiert. Diese Zahl bezeichnet man als *Komplement.* Dieses Verfahren ist aber nur möglich, wenn der Zahlenbereich begrenzt ist. Bei Rechnern ist durch die Wortlänge, z. B. 16 bit oder 32 bit, der Zahlenbereich fest vorgegeben.

Machen wir uns das Prinzip an einem Beispiel aus dem Dezimalsystem klar.

Beispiel: Voraussetzung: Wir wollen nur *einstellige* Dezimalzahlen zulassen!
Berechnet werden soll: 8 - 3 = ?
Wir wollen diese Subtraktion durch eine Addition ersetzen. Welche Zahl muss man dann zu 8 addieren und wie erhält man das richtige Ergebnis?

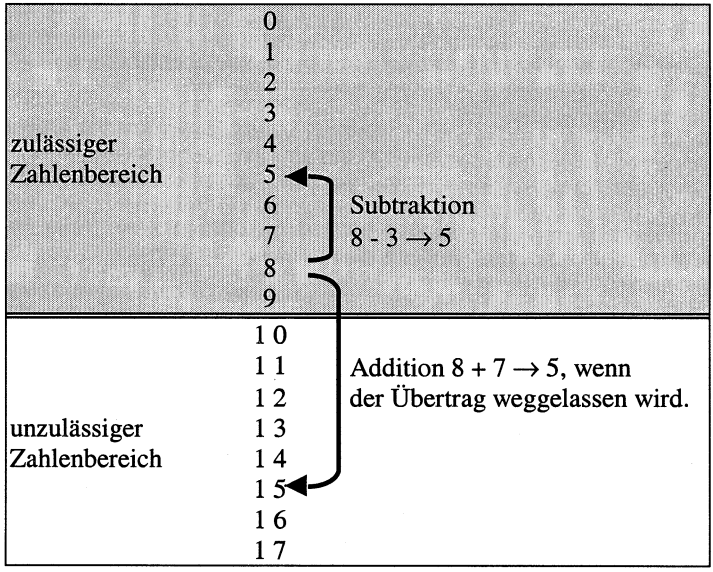

Bild 3-3: Prinzip der Komplement-Rechnung: Anstatt von 8 den Wert 3 zu subtrahieren, addiert man zu 8 das Komplement von 3 und lässt den Übertrag weg.

Wenn man zu 8 die Zahl 7 addiert, erhält man als Ergebnis 15. Da diese Zahl außerhalb des zulässigen Zahlenbereiches liegt, lässt man den Übertrag auf die nächste Stelle weg und hat das richtige Ergebnis 5. Dieses Ergebnis ist wieder eine Zahl aus dem zugelassenen Zahlenbereich.

Man bezeichnet die Zahl 7 als das (Zehner-)Komplement zu 3. Die Zahl (hier 3) plus ihr Komplement (hier 7) ergeben hier 10, also die größte zugelassene Zahl plus 1.

Als allgemeine Regel kann man aufstellen:

Eine Zahl x plus dem Komplement von x ergibt die größte zugelassene Zahl + 1.

Oder die Regel anders ausgedrückt:

Das Komplement zu einer Zahl x erhält man, indem man die größte zugelassene Zahl um 1 erhöht und davon x subtrahiert.

Bisher haben wir das Problem der Subtraktion anscheinend noch nicht gelöst, da man zur Komplement-Bildung doch noch subtrahieren muss. Im Dualsystem ist das zum Glück aber nicht notwendig, wie wir im Folgenden feststellen werden.

Die obige allgemeine Regel wird dabei konkretisiert auf Zweier- und Einer-Komplement.

3.3.2.1 Zweier-Komplement

Definition: Im Dualsystem erhält man zu einer n-stelligen Zahl x das *Zweier-Komplement* $K_2(x)$ durch Invertieren aller n Bits und anschließendem Addieren einer Eins.

Beispiel: $x = (91)_{10} = (0101\ 1011)_2$ mit n = 8, also als achtstellige Dualzahl.

Dualzahl	0101 1011
Invertierte Zahl	1010 0100
+ 1	0000 0001
Komplement $K_2(91)$	1010 0101

Probe: Addiert man die Dualzahl und ihr Zweier-Komplement, dann muss die größte zugelassene Zahl (bei n = 8: 1111 1111) plus 1 (also 1 0000 0000) herauskommen.

Dualzahl (91)	0101 1011
+ Komplement $K_2(91)$	1010 0101
	1 0000 0000 Stimmt also!

Die Zweier-Komplement-Bildung im Dualsystem ist damit zwar erklärt, aber das Ziel dieses Abschnittes ist es, negative Zahlen festzulegen. Wenn man eine größere Zahl von einer kleineren subtrahiert, erhält man zwangsläufig eine negative Zahl. Berechnen wir z. B. 3 – 4 bzw. 3 + $K_2(4)$:

Für n = 8 gilt: $(3)_{10} = (0000\ 0011)_2$ $(4)_{10} = (0000\ 0100)_2$

Dualzahl 4		0000 0100
$K_2(4)$:	invertiert	1111 1011
	+1	0000 0001
		1111 1100
Dualzahl 3		0000 0011
+ $K_2(4)$		1111 1100
- 1		1111 1111

Die Zahl -1 wird also bei Anwendung des Zweier-Komplements als 1111 1111 dargestellt. Mit ähnlichen Rechnungen kann man weitere negative Zahlen bestimmen. Dabei ist die linke Stelle immer eine 1. Wie bei der Darstellung „Betrag und Vorzeichen" bildet das höchstwertige Bit auch hier eine Vorzeichenstelle.

+8	0	1	0	0	0	
+7	0	0	1	1	1	n = 4 → 16 Zahlen
+6	0	0	1	1	0	
+5	0	0	1	0	1	
+4	0	0	1	0	0	
+3	0	0	0	1	1	n = 3 → 8 Zahlen
+2	0	0	0	1	0	
+1	0	0	0	0	1	n = 2 → 4 Zahlen
0	0	0	0	0	0	
-1	1	1	1	1	1	
-2	1	1	1	1	0	
-3	1	1	1	0	1	
-4	1	1	1	0	0	
-5	1	1	0	1	1	
-6	1	1	0	1	0	
-7	1	1	0	0	1	Positive Zahlen beginnen mit 0,
-8	1	1	0	0	0	negative Zahlen beginnen mit 1.
-9	1	0	1	1	1	

Bild 3-4: Dualzahlen im Zweierkomplement mit der benötigten Stellenanzahl

Bei der Zweier-Komplement-Darstellung gelten folgende Regeln:

1) Zahl auf die richtige Stellenanzahl bringen.
2) Bei positiven Zahlen steht als MSB eine 0 und bei negativen Zahlen eine 1.
3) Um den Betrag einer negativen Zahl zu erhalten, muss man wieder das Zweier-Komplement bilden.

Sehr wichtig bei der Komplement-Bildung ist die Festlegung und Einhaltung des zulässigen Zahlenbereiches. Deshalb sind im Bild 3-4 und in der Tabelle 3-5 die darstellbaren Zahlenbereiche angegeben.

Wortlänge	nur positiver Bereich	Zweierkomplement	
		positiver Bereich	negativer Bereich
n	0 ... $2^n - 1$	bis $2^{n-1} - 1$	bis -2^{n-1}
2	0 ... 3	1	-2
3	0 ... 7	3	-4
4	0 ... 15	7	-8
5	0 ... 31	15	-16
6	0 ... 63	31	-32
7	0 ... 127	63	-64
8	0 ... 255	127	-128
9	0 ... 511	255	-256
10	0 ... 1023	511	-512
11	0 ... 2047	1023	-1024
12	0 ... 4095	2047	-2048
16	0 ... 65.535	32.767	-32.768
24	0 ... 16.777.215	8.388.607	-8.388.608
32	0 ... 4.294.967.295	2.147.483.647	-2.147.483.648

Tabelle 3-5: Darstellbare Zahlenbereiche in Abhängigkeit von der Wortlänge

3.3.2.2 Einer-Komplement

Bei der Bildung des Zweier-Komplements muss man nach dem Invertieren noch eine 1 addieren. Diese Addition vermeidet man bei dem Einer-Komplement.

Definition: Im Dualsystem erhält man zu einer n-stelligen Zahl x das *Einer-Komplement* $K_1(x)$ durch Invertieren aller n Bits.

Beispiel: $x = (91)_{10} = (0101\ 1011)_2$ mit n = 8, also als achtstellige Dualzahl.

Dualzahl	0101 1011
Komplement $K_1(91)$	1010 0100

Probe: Addiert man die Dualzahl und ihr Einer-Komplement, dann muss die größte zugelassene Zahl (bei n = 8: 1111 1111) herauskommen.

Dualzahl (91)	0101 1011	
+ Komplement $K_1(91)$	1010 0100	
	1111 1111	Stimmt also!

Wie sind die negativen Zahlen beim Einer-Komplement festgelegt? Führen wir also wieder die Subtraktion 3 - 4 durch:

Für n = 8 gilt: $(3)_{10} = (0000\ 0011)_2$ $(4)_{10} = (0000\ 0100)_2$

Dualzahl 4		0000 0100
$K_1(4)$:	invertiert	1111 1011
Dualzahl 3		0000 0011
+ $K_1(4)$		1111 1011
- 1		1111 1110

Die Zahl -1 wird bei Anwendung des Einer-Komplements als 1111 1110 dargestellt. Auch bei den anderen negativen Zahlen steht immer eine 1 an der linken Stelle .

Es gibt beim Einer-Komplement eine „positive" Null und eine „negative" Null:

$K_1(x) = 1111\ 1111 \rightarrow x = 0000\ 0000$

+8	0	1	0	0	0	
+7	0	0	1	1	1	n = 4 → 16 Zahlen
+6	0	0	1	1	0	
+5	0	0	1	0	1	
+4	0	0	1	0	0	
+3	0	0	0	1	1	n = 3 → 8 Zahlen
+2	0	0	0	1	0	
+1	0	0	0	0	1	n = 2 → 4 Zahlen
+0	0	0	0	0	0	
-0	1	1	1	1	1	
-1	1	1	1	1	0	
-2	1	1	1	0	1	
-3	1	1	1	0	0	
-4	1	1	0	1	1	
-5	1	1	0	1	0	
-6	1	1	0	0	1	Positive Zahlen beginnen mit 0,
-7	1	1	0	0	0	negative Zahlen beginnen mit 1.
-8	1	0	1	1	1	

Bild 3-5: Dualzahlen im Einerkomplement mit der benötigten Stellenanzahl

Beim Einer-Komplement gelten ähnliche Regeln wie beim Zweier-Komplement:

1) Zahl auf die richtige Stellenanzahl bringen.
2) Bei positiven Zahlen steht als MSB 0 und bei negativen Zahlen eine 1.
3) Um den Betrag einer negativen Zahl zu erhalten, muss man wieder das Einer-Komplement bilden.

Da die Null beim Einer-Komplement doppelt vorkommt, ist der Zahlenbereich bei gleicher Stellenanzahl jeweils um eine Zahl kleiner als beim Zweier-Komplement (→ Bild 3-5).

3.3.2.3 Rechenregeln beim Einer- und Zweier-Komplement

Beim Rechnen mit Komplementen muss man folgende Regeln beachten:

1) Anzahl der Stellen festlegen bzw. zulässigen Zahlenbereich beachten.
2) Zahlen in voller Stellenzahl darstellen.
3) Einer- bzw. Zweier-Komplement bilden.

Zusätzlich:

1) Beim Zweier-Komplement wird ein Überlauf nicht berücksichtigt. (Voraussetzung ist natürlich, dass der Zahlenbereich eingehalten wird).
2) Gibt es beim Einer-Komplement einen Überlauf, dann muss man eine 1 addieren.

Beispiele: mit n = 8 Stellen; $(35)_{10} \rightarrow (0010\ 0011)_2$; $(91)_{10} \rightarrow (0101\ 1011)_2$

	Dezimal	**Einer-Komplement**	**Zweier-Komplement**
	35	$K_1(35) = 1101\ 1100$	$K_2(35) = 1101\ 1101$
	91	$K_1(91) = 1010\ 0100$	$K_2(91) = 1010\ 0101$
1)	35	0010 0011	0010 0011
	+ 91	0101 1011	0101 1011
	+126	0111 1110	0111 1110
2)	35	0010 0011	0010 0011
	- 91	1010 0100	1010 0101
	- 56	1100 0111	1100 1000
		$K_1(1100\ 0111) = 0011\ 1000$	$K_2(1100\ 1000) = 0011\ 1000$
3)	- 35	1101 1100	1101 1101
	+ 91	0101 1011	0101 1011
		1 0011 0111	1 0011 1000
		Überlauf → +1 0000 0001	Überlauf weglassen
	+ 56	0011 1000	0011 1000
4)	- 35	1101 1100	1101 1101
	- 91	1010 0100	1010 0101
		1 1000 0000	1 1000 0010
		Überlauf → +1 0000 0001	Überlauf weglassen
	- 126	1000 0001	1000 0010
		$K_1(1000\ 0001) = 0111\ 1110$	$K_2(1000\ 0010) = 0111\ 1110$

Fazit aus diesen Beispielen:

1) Bei positiven Zahlen rechnet man mit den „normalen" Dualzahlen. Da keine negativen Zahlen auftreten, gibt es keinen Unterschied, ob man für deren Darstellung das Einer- oder Zweier-Komplement vereinbart hat.
2) Ist ein Ergebnis negativ, kann man das entsprechende Komplement bilden, falls man den Betrag wissen möchte. Wenn es aber in einer nachfolgenden Rechnung weiter verwendet wird, kann man das Ergebnis direkt übernehmen.

Entscheidend bei der Komplement-Bildung ist die Einhaltung des zulässigen Zahlenbereiches (→ z. B. Tabelle 3-5). Ein Rechenwerk erkennt zwei Arten von möglichen Überschreitungen des Zahlenbereichs:

1) Tritt bei einer n-stelligen Zahl ein *Übertrag* in die nächst höhere Stelle n+1 auf, dann wird das *Carry*bit C gesetzt. Beim Rechnen mit positiven Zahlen (unsigned integer) bedeutet das eine Überschreitung des Zahlenbereichs.

   Beispiel: Addition von 255 und 1 als achtstellige Zahlen:

   $255|_{10} \rightarrow$ 1111 1111
   $1|_{10} \rightarrow$ 0000 0001
   1 0000 0000 → Das Ergebnis ist bei n = 8: $0|_{10}$.
   Deshalb als Markierung „Carry".

2) Gibt es bei einer n-stelligen Zahl einen *Überlauf* entweder in die Stelle n+1 oder in die (Vorzeichen-)Stelle n, dann wird das *Overflow*bit O ($O = C_{n+1}$ EXOR C_n) gesetzt. Beim Rechnen in der Komplement-Darstellung (signed integer) liegt dann eine Zahlenbereichsüberschreitung vor. Sie kann bei der Addition zweier positiver oder zweier negativer Zahlen auftreten.

   Beispiele: Addition von 127 und 1 als achtstellige Zahlen (Zweier-Komplement):

   $127|_{10} \rightarrow$ **0**111 1111
   $1|_{10} \rightarrow$ 0000 0001
   1000 0000 → Das Ergebnis lautet: $-128|_{10}$.
   Deshalb als Markierung „Overflow".

   Addition von -128 und -1 als achtstellige Zahlen mit Zweier-Komplement-Darstellung

   $-128|_{10} \rightarrow$ **1**000 0000
   $-1|_{10} \rightarrow$ 1111 1111
   1 **0**111 1111 → Das Ergebnis lautet: $+127|_{10}$.
   Deshalb als Markierung „Overflow".

Das Bild 3-6 zeigt, dass bei diesen drei Beispielen jeweils die Bereichsgrenze überschritten wird. Es muss also eine Markierung des Ergebnisses mit „Carry" bzw. „Overflow" erfolgen, da das fehlerhafte Ergebnis sonst als korrekt gewertet wird.

Das Rechenwerk selbst kann zwischen Zahlen ohne oder mit Vorzeichenstelle nicht unterscheiden. Deshalb muss das Programm vorgeben, ob entweder das C- oder O-Bit abzuprüfen ist.

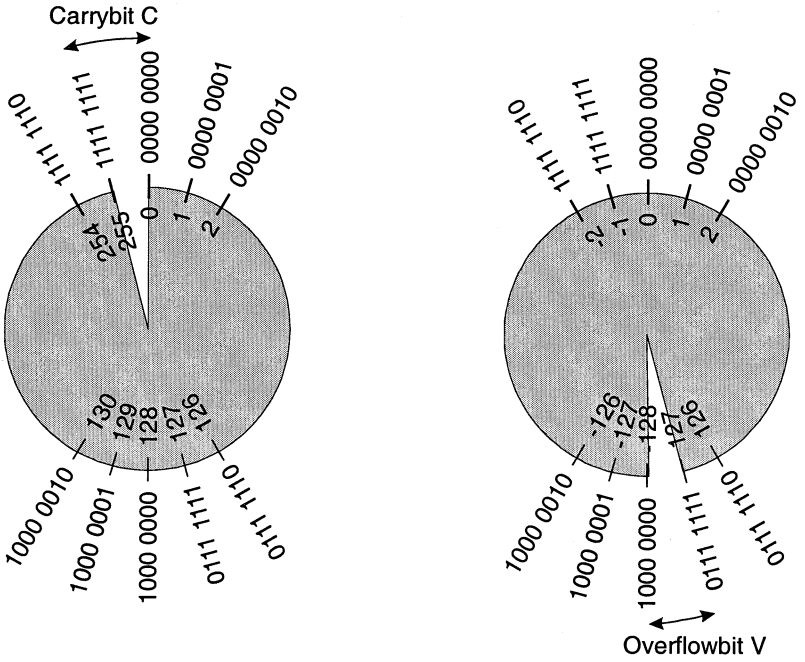

Bild 3-6: *Zahlenring* für achtstellige Zahlen: positive Zahlen (links) und Zweier-Komplement (rechts). Bei Bereichsüberschreitung wird das Carrybit C (links) bzw. das Overflowbit V (rechts) gesetzt.

Zusammenfassung: Komplement-Darstellung

Zahlenbereich (für n = 8):

von	0111 1111	also + $(2^7 - 1) = +127$
bis	1000 0000	also - $2^7 = -128$ beim Zweier-Komplement
bzw.	1000 0000	also - $(2^7 - 1) = -127$ beim Einer-Komplement.

Vorteile:
1) Anstelle einer Subtraktion addiert man das entsprechende Komplement des Subtrahenden: a - b = a + K(b) .
2) Beim Zweier-Komplement: Die Null wird eineindeutig dargestellt.

Nachteile:
1) Man muss das Komplement bilden.
2) Die Bildung des Zweier-Komplements ist etwas aufwendiger als beim Einer-Komplement.

3.3.3 Verschiebung um einen Basiswert

Wenn man einen gemischten Zahlenbereich aus positiven und negativen Zahlen nur mit positiven Werten darstellen will, dann kann man das erreichen, indem man den gemischten Zahlenbereich um einen *Basis-* oder *Biaswert* (bias, engl.: verschieben) ins Positive verschiebt. Der Bias-Wert entspricht etwa dem mittleren Wert des Zahlenbereichs.

Beispiel für n = 8: Gemischter Zahlenbereich -127 ... +128
→ Bias: 127
→ positiver Zahlenbereich 0 ... 255.

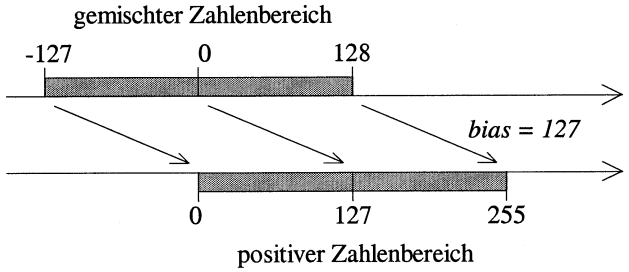

Bild 3-7: Darstellung von gemischten Zahlen durch Verschieben um einen Biaswert

Durch Addition des Basiswertes erhält man aus der gemischten Zahlendarstellung die positive, verschobene Zahl.

Beispiele: „45" im gemischten Zahlenbereich ergibt 45 + 127 = 172 im positiven Zahlenbereich.
„-28" im gemischten Zahlenbereich ergibt -28 + 127 = 99 im positiven Zahlenbereich.

Zusammenfassung: Ganze Zahlen

Zur Darstellung von negativen Zahlen benutzt man das Zweier- und Einer-Komplement, da das Rechenwerk positive und negative Zahlen einheitlich bearbeiten kann. Die Vor- und Nachteile zwischen den beiden Darstellungsformen wiegen sich zwar gegenseitig auf, aber die sehr weit verbreiteten Mikroprozessor-Familien Intel 80x86 und Motorola 680x0 arbeiten in ihren internen Rechenwerken mit dem Zweier-Komplement.

Dagegen beschränkt sich die Darstellung negativer Zahlen mit Vorzeichen und Betrag oder durch Verschiebung nur auf bestimmte Sonderfälle (→ Kap. 3.4.2):

- Vorzeichen und Betrag: Darstellung der Mantisse bei Gleitkommazahlen,
- Verschiebung: Darstellung des Exponenten bei Gleitkommazahlen.

3.4 Reelle Zahlen

Bisher haben wir nur ganze Zahlen betrachtet. Nun wollen wir auch gebrochene Zahlen zulassen. Wie kann man aber das Komma angeben, da die beiden binären Zeichen schon belegt sind? Dazu gibt es zwei verschiedene Strategien:

- Festkomma-Darstellung und
- Gleitkomma-Darstellung.

3.4.1 Festkomma-Darstellung

Bei der *Festkomma*-Darstellung deklariert man die Variable intern als Integer-Wert und verwaltet stattdessen die Kommastelle im Programm. Bei jeder Rechenoperation überprüft das Programm, ob und um wie viele Stellen sich das Komma verändert.

Die Eingabe der Zahlen kann man u. U. vereinfachen, indem man nur die Ziffernfolge ohne Komma anzugeben braucht. So kann man sich bei häufigen Eingaben etwas Tipparbeit ersparen, besonders bei folgenden Fällen:

Beispiele:			
	kleine gebrochene Zahlen (Kapazität eines Kondensators)	(, 000) 47	Farad
	gemischte Zahlen	1 234, 56	€
	große ganze Zahlen (mittlere Entfernung Erde – Mond)	384 (000,)	km

Da z. B. bei einer Festkomma-Addition das Komma aller Operanden an der gleichen Stelle stehen muss, sind gegebenenfalls die Operanden vorher zu transformieren. Dabei entstehen durch Auf- bzw. Abrunden Ungenauigkeiten.

Beispiel: 113,46 € + 16 % MwSt.

16 % von 113,46 = 18,1536 →

$$\begin{array}{r} 113{,}46\ € \\ \underline{18{,}15\ €} \\ 131{,}61 € \end{array}$$

3.4.2 Gleitkomma-Darstellung

Bei der *Gleitkomma*-Darstellung (auch halblogarithmische Darstellung genannt) ist das Komma ein fester Bestandteil der Zahl. Durch eine standardisierte Schreibweise braucht man es nicht explizit anzugeben.

Wir betrachten im Folgenden die *Floatingpoint*-Darstellung nach dem Standard ANSI/IEEE 754 (US-American National Standard Institute / Institute of Electrical and Electronics Engineers). Dieser Standard hat sich inzwischen durchgesetzt und bildet die Grundlage sowohl für einen weltweiten Datenaustausch wie auch für die Rechenwerke in den Mikroprozessoren.

In der Gleitkomma-Darstellung wird jede Zahl z in der Form

$$z = \pm m \cdot b^{\pm e}$$

dargestellt. Dabei ist m die *Mantisse*, e der Exponent und b die Basis des Exponenten. Sowohl die Mantisse wie auch der Exponent haben ein Vorzeichen.

Zur Darstellung einer Gleitkomma-Zahl braucht man also folgende Angaben:

1) Vorzeichen der Mantisse,
2) Betrag der Mantisse mit Komma,
3) Basis (ganze Zahl),
4) Vorzeichen des Exponenten und
5) Betrag des Exponenten (ganze Zahl).

Fünf verschiedene Angaben für eine Zahl beanspruchen viel Speicherplatz. Deshalb will man diese Anzahl verkleinern. Dazu gibt es verschiedene Möglichkeiten. Im *IEEE-Standard 754* ist definiert:

- Das Vorzeichen s wird folgendermaßen festgelegt:
  s = 0 : positiv ; s = 1 : negativ .
- Die Angabe der Basis erübrigt sich, da die Basis durch die Hardware des Rechners festgelegt ist. Der IEEE-Standard bezieht sich auf die Basis 2.
- Um bei den Gleitkomma-Zahlen das Komma der Mantisse nicht darstellen zu müssen, werden die Zahlen *normalisiert.* Das heißt, bei der Mantisse steht das Komma rechts neben der höchstwertigen Stelle, die ungleich Null ist.

  Definition: Eine Gleitkomma-Zahl der Form $\pm\, m \cdot b^{\pm e}$ heißt normalisiert, falls gilt:

  $$b > |m| \geq 1.$$

  Beispiele:

Dezimalsystem	654,321	$\rightarrow 6{,}54321 \cdot 10^{2}$
	0,06543	$\rightarrow 6{,}543 \cdot 10^{-2}$
Dualsystem	1001,011	$\rightarrow 1{,}001011 \cdot 2^{3}$
	0,011011	$\rightarrow 1{,}1011 \cdot 2^{-2}$

  Da im Binärsystem $2 > |m| \geq 1$ gilt, muss die Stelle links vom Komma stets 1 sein. Wenn aber jede normalisierte Zahl im Dualsystem mit „1," beginnt, dann kann man diesen Teil auch weglassen. Deshalb gibt man mit *fraction f* (engl.: echte gebrochene Zahl) nur die Stellen rechts vom Komma an. Beim Fraction *f* fehlt gegenüber der Mantisse die führende Eins und das Komma.

- Das Vorzeichen des Exponenten e vermeidet man, indem man den Zahlenbereich um einen *Basiswert* (bias) verschiebt (siehe Abschnitt 3.3.3). Den verschobenen (positiven) Exponenten bezeichnet man auch als *Charakteristik c* oder *biased Exponent.* Diese Art der Darstellung hat den Vorteil, dass man zwei Exponenten schnell vergleichen kann. Für die Umwandlung gilt:

  Charakteristik = biased Exponent = Exponent + bias

  Bei der Charakteristik benutzt man den größten und kleinsten Wert nur für spezielle Ersatzdarstellungen, die wir später noch betrachten werden.

Beispiel für einen Exponent mit n = 8:
→ Charakteristik: 1 bis 254 (0 und 255 haben spezielle Bedeutungen)
→ Exponent: -126 bis +127
→ Bias: 127

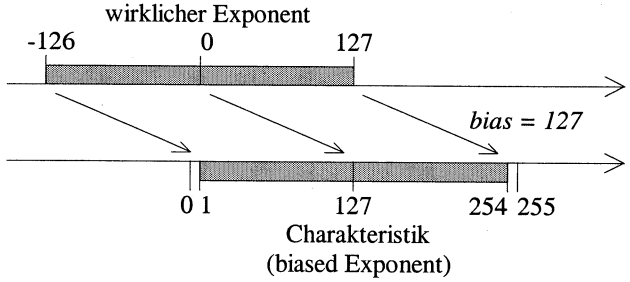

Bild 3-8: Umwandlung des Exponenten in die Charakteristik

Beispiele: Exponent = 45 → Charakteristik = 45 + 127 = 172
Exponent = -28 → Charakteristik = -28 + 127 = 99

Jetzt benötigt man nur noch 3 Angaben zur Darstellung einer Gleitkommazahl:

- das Vorzeichen der Mantisse,
- die Mantisse in normalisierter Form und
- den Exponenten als biased-Wert (Charakteristik).

In der IEEE 754 sind single, double und extended precision als Standardformate definiert. In den Coprozessoren der früheren Mikroprozessor-Generationen wurde das extended precision Format verwendet. Aus dieser Zeit ist auch die Angabe der Mantisse (also mit „1,“) anstatt des Fractions zu erklären. Heute arbeitet z. B. der Pentium intern in diesem Format. Seine acht Gleitkommaregister sind je 80 bit lang.

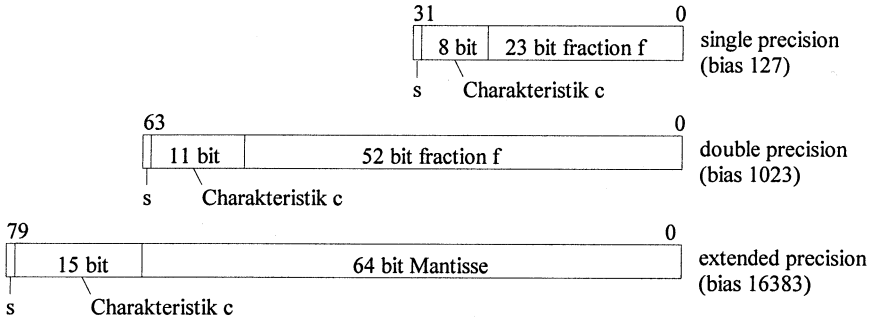

Bild 3-9: 32, 64 und 80 bit Standardformate für Gleitkommazahlen nach IEEE 754

Beispiel:

0	1000 1111	1000 1001 1010 1011 1100 000
Vorzeichen	Charakteristik	fraction

$$+ 1{,}1000\ 1001\ 1010\ 1011\ 1100\ 000 \cdot 2^{1000\ 1111 - 0111\ 1111}$$
$$= + 1{,}1000\ 1001\ 1010\ 1011\ 1100\ 000 \cdot 2^{0001\ 0000}$$
$$= (+\ 1\ 1000\ 1001\ 1010\ 1011{,}1100\ 000)_2 = (+\ 100.779{,}75)_{10}$$

Die Stellenanzahl der Mantisse entscheidet über die Genauigkeit der Gleitkomma-Zahl, während die Stellenanzahl des Exponenten die Größe des darstellbaren Zahlenbereichs angibt:

Größte positive Zahl:

Vorzeichen	Charakteristik	fraction	Wert
0	1111 1110	1111 1111 1111 1111 1111 111	$= (2 - 2^{-23}) \cdot 2^{127}$
			$\approx 1 \cdot 2^{128}$
			$= 3{,}4 \cdot 10^{38}$

Kleinste positive Zahl:

Vorzeichen	Charakteristik	fraction	Wert
0	0000 0001	0000 0000 0000 0000 0000 000	$= 1 \cdot 2^{-126}$
			$= 1{,}2 \cdot 10^{-38}$

Größte negative Zahl:

Vorzeichen	Charakteristik	fraction	Wert
1	0000 0001	0000 0000 0000 0000 0000 000	$= -1 \cdot 2^{-126}$

Kleinste negative Zahl:

Vorzeichen	Charakteristik	fraction	Wert
1	1111 1110	1111 1111 1111 1111 1111 111	$= -(2 - 2^{-23}) \cdot 2^{127}$
			$\approx -1 \cdot 2^{128}$

Trägt man die Bereiche auf einer Zahlengeraden auf, dann sieht man, dass um den Nullpunkt herum eine kleine Lücke besteht.

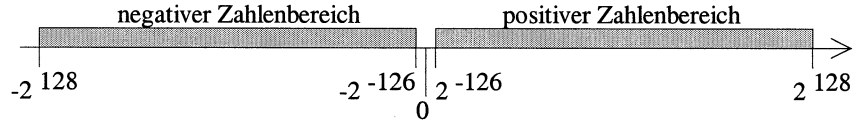

Bild 3-10: Zwischen dem positiven und negativen Zahlenbereich ist eine kleine Lücke (hier viel zu groß gezeichnet). Dadurch kann die Null nicht dargestellt werden.

Die Null selbst kann man nach dem beschriebenen Verfahren nicht darstellen. Der Grund dafür liegt in der Normalisierung: Die Mantisse m kann nur Werte im Intervall $\{m \mid 1 \leq m < 2\}$ und nicht den Wert 0 annehmen. Oder anders ausgedrückt: Die

Zahl 0 kann man nicht normalisieren. Deshalb benötigt man für die Null eine Ersatzdarstellung. Dazu verwendet man die dafür reservierten Werte der Charakteristik. Nach dem IEEE Standard 754 gilt:

Normalisiert	±	0 < Charakteristik < max	Beliebiges Bitmuster
Nicht normalisiert	±	0	Beliebiges Bitmuster ≠ 0
Null	±	0	0
Unendlich	±	111 ... 1 (max)	0
Keine Zahl	±	111 ... 1 (max)	Beliebiges Bitmuster ≠ 0

(NaN: not a number)

Wie sind die Gleitkommazahlen nun in den beiden Zahlenbereichen verteilt? Betrachten wir nur den positiven Zahlenbereich. Zu Anfang dieses Kapitels haben wir festgestellt, dass das Bit mit der höchsten Wertigkeit den Zahlenbereich halbiert (→ Bild 3-2). Diese obere Hälfte wird dann durch die Mantisse linear geteilt. Die Stellenanzahl n des Fractions bestimmt dabei die Anzahl der Teilungen, nämlich 2^n.

Die untere Hälfte wird wiederum halbiert. Deren obere Hälfte (Exponent – 1) enthält genau so viele Gleitkommazahlen wie bei der ersten Halbierung. Folglich ist der Abstand jetzt nur halb so groß. Das setzt sich so fort: Die Gleitkommazahlen werden mit jeder neuen Halbierung doppelt so dicht, wie Bild 3-11 schematisch zeigt.

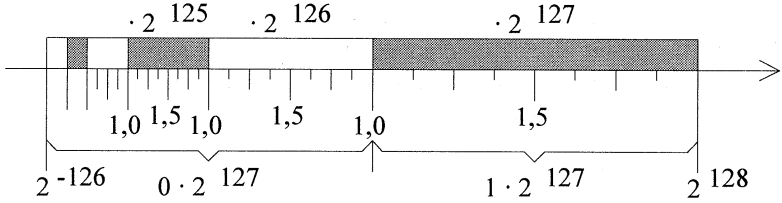

Bild 3-11: Prinzipielle Verteilung der Gleitkommazahlen innerhalb des positiven Zahlenbereichs (hier mit n = 3, d. h. 8 Teilungen).

Die Gleitkomma-Operationen sind komplexer als Festkomma-Operationen:

- Bei Addition bzw. Subtraktion müssen beide Operanden auf den gleichen Exponent gebracht werden.
- Mantisse und Exponent müssen getrennt behandelt werden.

Früher hat man für die Gleitkomma-Arithmetik meist einen Floatingpoint-*Coprozessor* (intel 80x87, Motorola 68881 und 68882) verwendet. Heute sind sie in den Mikroprozessoren integriert. Ohne Coprozessor werden die Gleitkomma-Operationen per Software realisiert, und zwar mit erheblichen Ausführungszeiten.

3.4.3 Vergleich von Festkomma- und Gleitkomma-Darstellung

Geht man von der gleichen Wortlänge aus, so ist die Genauigkeit der Gleitkomma-Darstellung kleiner als die der Festkomma-Darstellung, da Stellen für den Exponenten reserviert werden müssen. Der Bereich der darstellbaren Zahlen dagegen ist wesentlich größer.

Wählt man die Darstellungen (Wortlänge 32 bit)

Festkomma: 1 bit Vorzeichen + 31 bit Zahlenwert
Gleitkomma: 1 bit Vorzeichen + 8 bit Charakteristik + 23 bit fraction,

dann ergibt sich folgender Vergleich:

	Genauigkeit	Zahlenbereich (gerundet)
Festkomma	31 Binärstellen	$+2^{31}$ bis -2^{31} (bei ganzen Zahlen)
Gleitkomma	24 Binärstellen	$+2^{128}$ bis -2^{128}.

Die Festkomma-Darstellung wird man überall dort verwenden, wo es auf eine hohe Genauigkeit ankommt und der Zahlenbereich nicht sehr groß sein muss. Das gilt vorwiegend im kaufmännischen Bereich. Dagegen benötigt man im wissenschaftlichen Bereich häufig einen großen Zahlenbereich bei etwas geringerer Genauigkeit. Allerdings muss man dafür eine längere Rechenzeit in Kauf nehmen.

3.5 Andere Binärcodes

Der Dualcode ist, vergleichbar mit dem Dezimalcode, ein „normaler" Stellenwertcode: Jede Stelle ist mit einer Zweierpotenz gewichtet. Bei anderen Binärcodes ist der Stellenwert entweder anders festgelegt oder kann gar nicht angegeben werden, da er sich von Zahl zu Zahl ändert. Hier sollen nur wenige Codes erwähnt werden.

3.5.1 Unbeschränkte Binärcodes

In der Gruppe von Binärcodes, die keine feste Länge haben, ist als wichtigster Vertreter der *Gray-Code* zu nennen.

Dezimalzahl	Gray-Code
0	00000
1	00001
2	00011
3	00010
4	00110
5	00111
6	00101
7	00100

Dezimalzahl	Gray-Code
8	01100
9	01101
10	01111
11	01110
12	01010
13	01011
14	01001
15	01000

Dezimalzahl	Gray-Code
16	11000
17	11001
18	11011
19	11010
20	11110
21	11111
22	11101
23	11100

Tabelle 3-6: Der Gray-Code

Der Gray-Code ist dadurch charakterisiert, dass sich benachbarte Codeworte nur in einer Stelle unterscheiden. Er gehört damit zur Gruppe der so genannten *einschrittigen Codes*, die für viele technische Anwendungen sehr wichtig sind.

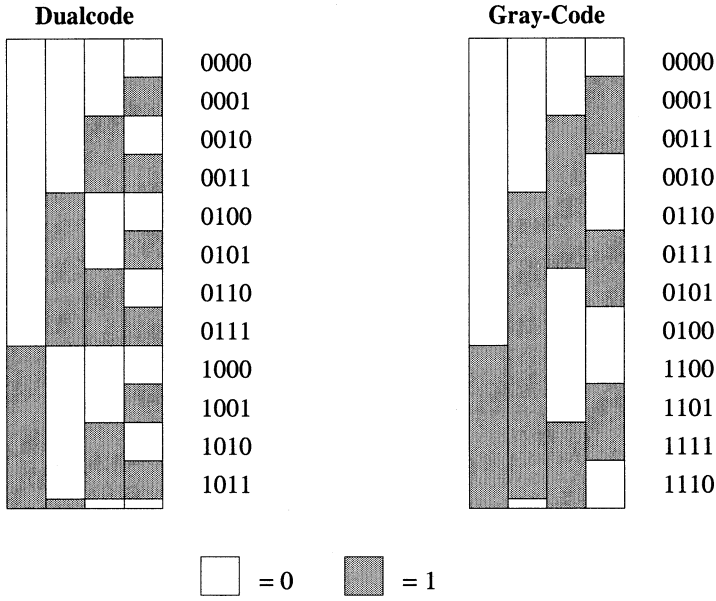

Bild 3-12: Beim Gray-Code ändert sich von einem Codewort zum nächsten nur 1 Bit.

3.5.2 Tetradencodes

Zur Darstellung von Dezimalzahlen benutzt man häufig sogenannte *Tetradencodes* (tetrade, griechisch: Vierergruppe). Dabei wird jede Dezimalziffer durch eine vierstellige Binärzahl dargestellt. Eine n-stellige Dezimalzahl benötigt also n Tetraden.

Beispiel:

Dezimalzahl	4	7	1	1	5
BCD-Zahl	0100	0111	0001	0001	0101

Die Tetradencodes nennt man auch *BCD-Codes* (binary coded decimals). Leider ist der Begriff „BCD" doppeldeutig:

- Zum einen bezeichnet man allgemein alle Tetradencodes, die die zehn Dezimalziffern codieren, als BCD-Codes und
- andererseits versteht man darunter nur einen speziellen Code, bei dem jede Tetrade im *Dualcode* dargestellt wird.

Letzteren Fall bezeichnet man manchmal auch als NBCD-Code (normal BCD).

Dezimal-ziffer	(N)BCD-Code 2^3 2^2 2^1 2^0
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
	1 0 1 0
	1 0 1 1
	1 1 0 0
	1 1 0 1
	1 1 1 0
	1 1 1 1

Dezimal-ziffer	3-Exzess-Code
	0 0 0 0
	0 0 0 1
	0 0 1 0
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0
	1 1 0 1
	1 1 1 0
	1 1 1 1

Dezimal-ziffer	Aiken-Code 2^1 2^2 2^1 2^0
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
	0 1 0 1
	0 1 1 0
	0 1 1 1
	1 0 0 0
	1 0 0 1
	1 0 1 0
5	1 0 1 1
6	1 1 0 0
7	1 1 0 1
8	1 1 1 0
9	1 1 1 1

Bild 3-13: Die 3 bekanntesten Tetradencodes: (N)BCD-, 3-Exzess- und Aiken-Code

Bei 4 binären Stellen sind insgesamt 16 verschiedene Kombinationen (Tetraden) möglich. Davon werden nur 10 Tetraden zur Darstellung der 10 möglichen Dezimalziffern benötigt. Die restlichen 6 Kombinationen werden nicht gebraucht. Man bezeichnet sie als *Pseudotetraden.*

Den Vorteil der leichten Umrechnung zwischen Dezimal- und BCD-Code erkauft man sich durch längere Datenworte (Speicherbedarf!). Außerdem ist das Rechnen im BCD-Code aufwendiger, weil die ungenutzten Kombinationen „übersprungen" werden müssen.

Beispiel: Die Addition 5147 + 2894 = 8041 würde im BCD-Code zunächst ergeben:

```
5147: 0101  0001  0100  0111
2894: 0010  1000  1001  0100
      0111  1001  1101  1011
```

Da dieses Zwischenergebnis in den beiden rechten 4 bit-Folgen (Tetraden) keine gültigen BCD-Codes enthält, muss es von rechts beginnend korrigiert werden. Durch Addition von „6" erhält man das richtige Ergebnis, weil das genau die Anzahl der Code-Kombinationen ist, die beim BCD-Code übersprungen werden und dadurch ungenutzt bleiben:

0111	1001	1101	1011	
			0110	Addition wegen ungültiger BCD-Stelle
		1	0001	Addition des Übertrages
0111	1001	1110	0001	
0111	1001	1110	0001	
		0110		Addition wegen 2. ungültiger BCD-Stelle
	1	0100	0001	Addition des Übertrages
0111	1010	0100	0001	
0111	1010	0100	0001	
	0110			Addition wegen 3. ungültiger BCD-Stelle
1	0000	0100	0001	Addition des Übertrages
1000	0000	0100	0001	

Dieses Beispiel zeigt, wie umständlich durch die zahlreichen Korrekturschritte eine einfache Addition sein kann. Es ist zwar bewusst so extrem gewählt, aber deshalb nicht unrealistisch. Nur mit einem speziellen Hardware-BCD-Rechenwerk können die Rechenoperationen schnell genug durchgeführt werden. Zum Beispiel erfolgen bei der Motorola 680x0 Mikroprozessor-Familie Addition und Subtraktion byteweise, also nur zwei Dezimalstellen gleichzeitig. Deshalb ist die BCD-Arithmetik recht langsam. Dadurch wird der BCD-Code heute nur noch in Ausnahmen verwendet.

3.6 Oktal- und Hexadezimalcode

Diese beiden Codes sind zwar vom Dualcode abgeleitet, gehören aber nicht zu den Binärcodes. Da die Dualzahlen im Vergleich zu den Dezimalzahlen drei- bis viermal so viele Stellen benötigen, fasst man beim *Oktalcode* (d. h. Basis 8) 3 Stellen und beim *Hexadezimalcode* (d. h. Basis 16) 4 Stellen zusammen (beginnend bei 2^0, d. h. beim Komma).

Beispiel: Dualzahl 010100101001

Dualzahl	010	100	101	001
Oktalzahl	2	4	5	1

Dualzahl	0101	0010	1001
Hexadezimalzahl	5	2	9

Oktal- und Hexadezimalsystem sind auch Stellenwertsysteme:

- Jeder Stelle beim Oktalsystem ist eine Potenz von 8 zugeordnet.
- Jeder Stelle beim Hexadezimalsystem ist eine Potenz von 16 zugeordnet.

Beim Hexadezimalcode werden die Ziffern, die größer als 9 sind, mit den Großbuchstaben A bis F bezeichnet.

Dualzahl	Oktalziffer
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

Dualzahl	Hexadezimalziffer
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

Bild 3-14: Oktal- und Hexadezimalcode

3.7 ASCII-Code

Bisher haben wir kennen gelernt, wie man ein- oder mehrstellige Zahlen darstellen kann. Daneben will man aber auch Texte speichern und verarbeiten können. Deshalb hat man für Buchstaben, Ziffern, Sonderzeichen und auch verschiedene Steuerzeichen einen Code definiert, der sich international durchgesetzt hat. Er heißt *ASCII-Code* (american standard code for information interchange) und ist auch in einer deutschen Norm (DIN 66003) festgelegt (→ Bild 3-15).

In diesem Zeichensatz fehlen aber die nationalen Sonderzeichen. Deshalb ersetzt man die Codes 5B bis 5D und 7B bis 7E, das sind die Sonderzeichen [, \,], {, |, } und _ , bei der deutschen Version durch die Umlaute und das „ß“ (→ Bild 3-16).

Mit dieser Version können deutsche Texte erfasst und bearbeitet werden. Will man aber in dem Text z. B. französische oder spanische Wörter verwenden oder gar Formeln mit griechischen Buchstaben einfügen, dann fehlen die entsprechenden nationalen bzw. griechischen Zeichen. Deshalb hat IBM mit ihren PCs einen Zeichensatz definiert, der aus 256 Zeichen (also 8 bit) besteht und viele wichtige Schrift- und Grafikzeichen enthält.

	Hexdez.	0.	1.	2.	3.	4.	5.	6.	7.
Hexdez.	Binär	000....	001....	010....	011....	100....	101....	110....	111....
.0	...0000	NUL	DLE	SP	0	@	P	`	p
.1	...0001	SOH	DC1	!	1	A	Q	a	q
.2	...0010	STX	DC2	"	2	B	R	b	r
.3	...0011	ETX	DC3	#	3	C	S	c	s
.4	...0100	EOT	DC4	$	4	D	T	d	t
.5	...0101	ENQ	NAK	%	5	E	U	e	u
.6	...0110	ACK	SYN	&	6	F	V	f	v
.7	...0111	BEL	ETB	´	7	G	W	g	w
.8	...1000	BS	CAN	(	8	H	X	h	x
.9	...1001	HT	EM	)	9	I	Y	i	y
.A	...1010	LF	SUB	*	:	J	Z	j	z
.B	...1011	VT	ESC	+	;	K	[	k	{
.C	...1100	FF	FS	,	<	L	\	l	|
.D	...1101	CR	GS	-	=	M	]	m	}
.E	...1110	SO	RS	.	>	N	~	n	¯
.F	...1111	SI	US	/	?	O	_	o	DEL

Abkürzungen:

Übertragungssteuerung	**Formatsteuerung**	**allgem. Steuerzeichen**
SOH start of heading	BS backspace	NUL null
STX start of text	HT horizontal tabulation	SO shift out
ETX end of text	LF line feed	SI shift in
EOT end of transmission	VT vertical tabulation	SUB substitution
ENQ enquiry	FF form feed	ESC escape
ACK acknowledge	CR carriage return	FS file separator
BEL bell		GS group separator
DLE data link escape		RS record separator
DC .. device control # ..		US unit separator
NAK negative acknowledge		SP space
SYN synchronisation		DEL delete
ETB end of transmission block		
CAN cancel		
EM end of medium		

Bild 3-15: ASCII-Code (7 bit)

	Hexdez.	0 .	1 .	2 .	3 .	4 .	5 .	6 .	7 .
Hexdez.	Binär	000....	001....	010....	011....	100....	101....	110....	111....
. A	...1010	LF	SUB	*	:	J	Z	j	z
. B	...1011	VT	ESC	+	;	K	Ä	k	ä
. C	...1100	FF	FS	,	<	L	Ö	l	ö
. D	...1101	CR	GS	-	=	M	Ü	m	ü
. E	...1110	SO	RS	.	>	N	~	n	ß
. F	...1111	SI	US	/	?	O	_	o	DEL

Bild 3-16: Deutsche Version des ASCII-Codes

In der Norm ISO-8859 sind 8 bit-Zeichensätze festgelegt. Allerdings gibt es inzwischen über 10 verschiedene Varianten, die sich in den oberen 128 Zeichen unterscheiden und Erweiterungen z. B. für west- und osteuropäische Sprachen enthalten.

Um einen weltweit einheitlichen Zeichensatz zu erreichen, ist das *Unicode*-Konsortium gegründet worden. Mit dem Unicode-Standard 3.0 hat es 49194 Zeichen mit einem 32 bit-Code festgelegt. Wichtige Designkriterien waren dabei, dass

- bisherige Zeichensätze möglichst erhalten bleiben,
- identische Zeichen nur einmal vorkommen
- und viele Zeichen zusammengesetzt werden (z. B. Zeichen und Akzent).

Für die Ein/Ausgabe von Unicode setzt sich UTF-8 durch. Mit einem Byte kann man den bisherigen 7 bit-ASCII-Code angeben. Bis zu 2048 Zeichen kann man in zwei Byte codieren und mit maximal sechs Byte bis zu 2^{31} Zeichen festlegen.

3.8 Zusammenfassung der Zahlensysteme

Die für uns interessanten *Zahlensysteme* sind im Bild 3-17 zusammengestellt. Der ASCII-Code dient nur der Darstellung von einzelnen Ziffern (und Buchstaben) und gehört damit nicht zu den Zahlensystemen.

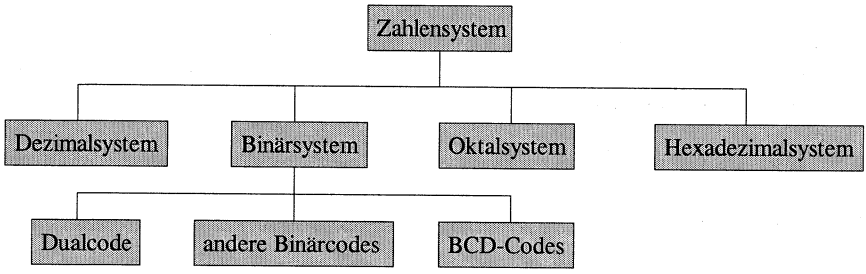

Bild 3-17: Zusammenstellung der wichtigsten Zahlensysteme