Zahlendarstellungen und Umrechnungen

Bernhard Kilger, Nov. 2018

Für Nina

Inhaltsverzeichnis

1 Die verschiedenen Zahlensysteme

Jede natürliche Zahl lässt sich als Kombination von Ziffern darstellen, deren Reihenfolge eine wichtige Rolle spielt. Wenn wir die Ziffernfolge 2018 hinschreiben, so nehmen wir ganz gewohnheitsmäßig an, dass der 2 ein höherer Stellenwert zukommt als der 8, und dass wir eigentlich die Summe 2000 + 10 + 8 bilden. Es gibt nun je nach dem verwendeten Ziffernvorrat ganz unterschiedliche Zahlensysteme dieser Art, die übrigens auch auf gebrochene Zahlen angewandt werden können. Im folgenden sollen allerdings nur ganze Zahlen, also Elemente aus der Menge **N**, näher betrachtet werden.

1.1 Binärsystem

Die primitivste Variante besteht aus nur zwei Ziffern, 0 und 1, und nennt sich duales oder binäres Zahlensystem. Dies ist auch die Variante, mit der Computer (oder überhaupt elektronische Bauteile) umgehen, weil diese beiden Ziffern mit den Schaltzuständen *aus – ein* korrelieren und deshalb eine einfache physikalische Entsprechung besitzen (z.B. in Transistoren). Ein einzelner Schalter dieser Art hat den Informationsgehalt von einem **Bit**, und acht davon fasst man im Computerwesen bekanntlich zu einem **Byte** zusammen.

Eine einzelne Ziffer hat bei dieser Darstellung eben nur das Repertoire 0 bis 1, was ein wenig dürftig ist. Wenn du zwei Euro in der Tasche hast, wäre das auf diese Weise schon nicht mehr darstellbar. Man behilft sich also mit der besagten Kombination von Ziffern, und zwar derart, dass die Position einer Ziffer Auskunft über deren Wertigkeit gibt. Diese nimmt **von rechts nach links** in einer Potenzentwicklung **zu.** Für das Binärsystem legen wir folgendes Schema zugrunde:

Pos	**n**	**. . .**	**. . .**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Wert	**2^n**	**. . .**	**. . .**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**

Jede dieser Positionen kann nun mit einer der Ziffern aus dem gegebenen Vorrat besetzt werden, beim Binärsystem also mit 0 oder 1. Mit dieser Ziffer ist dann der Wert an der betr. Position zu multiplizieren.

Beispiel:

1	**1**	**0**	**0**	**0**	**1**	**0**	**1**

Dies ergibt die (dezimale) Zahl: 1x128 + 1x64 + 0x32 + 0x16 +0x8 + 1x4 +0x2 +1x1 = **197**

Auf alle führenden Nullen, also im Beispiel links von Position 7, kann man verzichten, da diese am Gesamtwert nichts ändern. Dennoch sollte man, wenn man sich überhaupt mit Zahlendarstellungen bei Computern und den zugehörigen Programmen beschäftigt, stets **eine** führende Null hinschreiben, um kenntlich zumachen, dass es sich bei diesem Datenelement überhaupt um eine **Zahl** handelt und nicht um einen symbolischen Namen oder ein Textfragment. Die Basis (in diesem Fall 2) kann man, um Missverständnisse zu vermeiden, als symbolische Kennung rechts daneben schreiben. Es hat sich hierfür das b für das Binär-, d für das Dezimal- und h (oder auch x) für das weiter unten zu besprechende Hexadezimalsystem eingebürgert. Eine andere Notation ist ein tiefgestellter Index für die Basiszahl.

Die Kurzform für das obige Beispiel würde also so hingeschrieben:

011000101b = 0197d

(oder auch: **$011000101_2 = 0197_{10}$**)

Wenn man die Position mit p und den Wert an der betr. Stelle mit a_p bezeichnet, kommt man auf folgende allgemeine Formel für die Darstellung einer Binärzahl mit (n+1) Ziffern:

$$\text{Binärwert} = \sum_{p=0}^{n} a_p \, 2^p$$

(In Worten: Summe über die Produkte $\mathbf{a_p \cdot 2^p}$ für alle p von 0 bis n)

1.2 Dezimalsystem

Jetzt hat sich – gewollt oder nicht – bereits der Begriff des Dezimalsystems eingeschlichen. Aus Sicht der Maschine ist das eine eher unerwünschte und auch unnötige Komplizierung, für uns Menschen hingegen die primäre und beinahe schon angeborene Art, mit Zahlen umzugehen. Dies liegt augenscheinlich hauptsächlich an der Fünfgliedrigkeit aller Säugetiere (Pentadaktylie), die uns dazu bringt, manche Dinge „an fünf Fingern" abzuzählen. Dann ist es bis zu zehn Fingern nur noch ein kleiner Schritt.

Beim Dezimalsystem ändert sich gegenüber dem bisher Gesagten eigentlich nur eine Kleinigkeit, nämlich dass die Basis nicht mehr 2, sondern 10 ist. Es gibt folglich auch 10 verschiedene Ziffern: 0, 1, 2, 3, 4, 5, 6, 7, 8. 9 (bitte niemals die **0** vergessen; sie ist die wichtigste Ziffer in der Mathematik überhaupt).

Das Tabellenschema von oben bekommt nun – in etwas kürzerer Form – folgende Gestalt:

Pos	**n**	**. . .**	**5**	**4**	**3**	**2**	**1**	**0**
Wert	**10^n**	**. . .**	**100000**	**10000**	**1000**	**100**	**10**	**1**

Jede der Positionen kann mit einer der Ziffern 0 bis 9 besetzt werden.

Wieder ein Beispiel:

3	**5**	**0**	**1**	**9**

Ergebnis: 3x10000 + 5x1000 + 0x100 + 1x10 + 9x1 = **35019,**

was jetzt natürlich niemanden überrascht. Ebenso trivial ist die Kurzform:

035019d = 035019d

Das Suffix **d** kann man übrigens ebenso wie die führende **0** bei Dezimalzahlen im allgemeinen weglassen. Sie verstehen sich meist von selbst.

In der allgemeinen Formel von oben muss man nur die 2 durch eine 10 ersetzen:

$$\text{Dezimalwert} = \sum_{p=0}^{n} a_p \, 10^p$$

Die Dezimaldarstellung ist für uns Menschen schon eine große Erleichterung, weil sie in kurzer Form den Umgang mit riesigen Zahlen erlaubt. Würde man die Zahl im genannten Beispiel binär hinschreiben, so wäre das Resultat bereits ein ganz schöner Rattenschwanz:

035019d = 1000100011001011b

Die Zahl 35019 ist natürlich viel handlicher, und jedermann kann sich darunter etwas vorstellen.

Von den Umrechnungen wird in Kap. 2 näher die Rede sein.

1.3 Hexadezimalsystem

Dieser Begriff wurde oben schon kurz erwähnt und lässt sich nach dem Bisherigen mit wenigen Worten erklären: Die Basis beträgt jetzt nicht 2 und auch nicht 10, sondern **16**. Sprachlich ist der Begriff nicht ganz korrekt, weil in ihm ein lateinisches und ein griechisches Fragment vermischt werden. Man würde besser „sedezimal" sagen. Das tut aber keiner.

Wozu diese auf den ersten Blick ganz „krumme" Darstellung? Hierzu muss man sich wieder auf die Sichtweise des Computers begeben, der sich sozusagen am wohlsten fühlt, wenn er es immer nur mit 0 und 1 zu tun hat. Der Clou ist nun der, dass die Zahl 16 ihrerseits ebenfalls eine Zweierpotenz ist ($=2^4$) und dass dies obendrein auch noch für den Exponenten 4 gilt. Die 16 besteht sozusagen durch und durch aus Zweiern und ist somit dem Binärsystem wesensverwandt. Dadurch wird der Umgang mit ihr einem Programm und letzten Endes dem Prozessor sehr erleichtert. Auf der anderen Seite ist die 16 größenordnungsmäßig nicht allzu weit von der 10 entfernt und daher auch für den menschlichen Operator nicht

ganz unverständlich. Erfahrene Programmierer denken und rechnen „in Hex" genauso reibungslos und treffsicher wie mit Dezimalzahlen.

So gesehen ist die Basis 16 gar nicht so „krumm". Die Zahl 10, also das Doppelte von 5, ist viel krummer. Es besteht – abgesehen von der erwähnten Pentadaktylie – überhaupt kein sinnvoller Grund, ausgerechnet die 10 als Basis für das alltägliche Zählen und Rechnen zu verwenden. Ich bin sicher, dass der Mensch von Anfang an kein anderes Zahlensystem gekannt hätte als das Hexadezimalsystem, wenn der Liebe Gott uns 16 Finger verliehen hätte. Hat er aber nicht. Und somit sind wir zu einer Nachbesserung gezwungen, die in einer Erweiterung des Ziffernvorrats besteht. Aber das bereitet keine großen Schwierigkeiten, denn wir haben ja noch beliebig viele andere Zeichen, z.B. die Buchstaben, zur Verfügung. Wir ergänzen die Ziffernserie einfach um die Buchstaben A bis F und haben genau das, was wir brauchen, nämlich 16 Ziffern: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Die höchste Ziffer F hat den Dezimalwert 15 (nicht etwa 16, denn die Reihe beginnt wie immer mit 0!).

Die Tabelle bekommt jetzt folgende Gestalt:

Pos	**n**	**.**	**5**	**4**	**3**	**2**	**1**	**0**
Wert	**16^n**	**.**	**1048576**	**65536**	**4096**	**256**	**16**	**1**

Hierzu wieder ein Beispiel:

1	**A**	**0**	**9**	**F**

Ergebnis: 1x65536 + 10x4096 + 0x256 + 9x16 + 15x1 = **106655.**

Kurzform (anstelle des Suffix **h** ist auch das **x** gebräuchlich):

01A09Fh = 0106655d

In der allgemeinen Formel von oben muss – wen wundert's – die 2 durch eine 16 ersetzt werden:

$$\text{Hexadezimalwert} = \sum_{p=0}^{n} a_p \, 16^p$$

Der Wert an der Position 5 entspricht übrigens dem vielzitierten Megabyte (MB). Das „Mega" ist hier, wie man sieht, nicht ganz genau = 1,000,000, wie es die griechischen Präfixe verlangen würden. Aber man lässt hier die Kirche im Dorf, weil es so schön (beinahe) passt. Ähnliches gilt auch für das kilo in kB (= 1,024) und das Giga in GB (= 1,073,741,824).

Um das Verständnis abzurunden, sei darauf hingewiesen, dass es durchaus nicht der bereits vorhandenen Buchstaben A bis F bedurft hätte, um den Satz von 10 Ziffern auf 16 zu erweitern. Man hätte auch ganz andere Zeichen wählen oder auch neue Fantasieziffern erfinden und sie neu benennen können, etwa die sechs verknitterten Krünkel ϒ ⊥ ⊃ } { ∝ mit den Namen Ditz, Otz, Dutz, Tretz, Quatz und Fütz. Eine derartige Notation hätte durchaus einige Vorteile gehabt: Die Zeichen hätten automatisch als numerisch gegolten, sie hätten die führende Null erübrigt und wären nicht mit Textfragmenten verwechselbar gewesen. Aber dieser Zug ist natürlich längst abgefahren.

1.4 Verallgemeinerung

Wenn man die bisher genannten Definitionen verallgemeinert, kommt man zu einem System mit Potenzen einer beliebigen Grundzahl **g**. Diese werden so aneinander gereiht, dass man ganz rechts mit dem Exponenten 0 beginnt und ihn, indem man schrittweise nach links rückt, immer um 1 erhöht. Man beachte, dass der Exponent 0 immer die Einerstelle vertritt (x^0 ist stets = 1).

Dies geschieht also gemäß dem Schema:

$$\text{Zahl} = a_n g^n + \ldots + a_2 g^2 + a_1 g + a_0$$

Das System mit der Grundzahl **g** nennt man allgemein auch „**g-adische**" Zahlendarstellung. **g** ist bei dem uns geläufigen dekadischen System = 10, beim Binärsystem (duadisch) = 2 und beim Hexadezimalsystem (sedekadisch) = 16. Man könnte also genausogut vom 2-adischen, 10-adischen und 16-adischen System

reden. Wegen der herausragenden Bedeutung dieser drei Grundzahlen macht das aber keiner.

Die Koeffizienten aller Terme im obigen Summenschema, also die Werte a_p (p läuft wieder von 0 bis n) haben einen Ziffernvorrat, der von 0 bis (g-1) reicht. Dies sind die Mengen, die oben schon vorgestellt wurden, also {0, 1} für das Binär-, {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} für das Dezimal- und {0, ... 9, A ... F} für das Sedezimalsystem. Man beachte, dass die Elemente **0** und **1** grundsätzlich in allen Systemen vorkommen. Die **0** alleine wäre hingegen sinnlos, weil eine solche Darstellung überhaupt keine Unterscheidung und daher auch keine Information zulassen würde.

(Eine 1-adische Darstellung wäre vielleicht für die AfD ganz gut geeignet, weil diese Leute grundsätzlich zu allem **nein** sagen, also nur die **0** kennen. Da sie für weitere Differenzierungen ohnehin zu dumm sind, brauchen sie keine anderen Ziffern.)

1.5 Ungebräuchliche Grundzahlen

Nach der obigen Verallgemeinerung dürfte klar geworden sein, dass die bisher genannten drei Ausprägungen zwar sehr häufig verwendet werden, aber in keiner Weise irgendwie privilegiert sind.

Früher hat beispielsweise auch das **Oktalsystem** mit der Basis 8 eine gewisse Rolle gespielt. Hier wird also jede Zahl aus den acht Ziffern 0 bis 7 gebildet, die ihrerseits mit 3 Bits darstellbar sind. Frühere Computer- und Betriebssysteme, bei denen mit jedem Bit gegeizt werden musste, haben sich dieses Systems bedient. Dies galt z.B. für die berühmte PDP8 (Digital Equipment Corporation), die zu einem klassischen Museumsstück der Computergeschichte geworden ist.

In manchen älteren Kulturen (etwa bei den Babyloniern) wurde mit dem 12er-System (**Duodezimalsystem**) gearbeitet. Hierbei gibt es also außer den Ziffern 0 bis neun zwei weitere für die Zahlen 10d und 11d. Die Zwölf selbst benötigt wohlgemerkt keine eigene Ziffer, sondern wird durch die Darstellung 10_{12} repräsentiert. Wir kennen das ja schon.

Die Zwölf geistert auch in unserem Kulturkreis und Sprachgebrauch noch herum:

- Das Dutzend als gebräuchliche Mengenangabe
- 12 Tierkreiszeichen
- 12 Monate pro Jahr
- 12 Zahlen auf dem Zifferblatt der Uhr, dementsprechend Einteilung des Tages in 2 x 12 h
- Gradeinteilung des Vollkreises. Auch hierbei ist eine gewisse Verwandtschaft zur 12 zu erkennen, denn 360° = 12 x 30°, wobei 30° ein spezieller Winkel mit manchen Besonderheiten ist.

Nun ist die Zahl 12 für sich allein genommen noch kein „System", aber es gab (und gibt) auch die kaufmännische Mengenangabe **Gros**, die 12x12 = 12^2 beträgt. Hier zeigt sich ein rudimentärer Ansatz eines „Duodezimalsystems".

Ansonsten, vom mathematischen Standpunkt, sind alle Grundzahlen grundsätzlich gleichberechtigt. Niemand hindert uns daran, als Basis etwa die 7 oder 51 zu wählen.

1.6 Die Basis 128

Wir wollen das einmal aus einem bestimmten Grund mit der Basis **128** tun und den Werten 0 bis 127 nachstehende Zeichen zuordnen (die Positionen 0 bis 31 wollen wir erst einmal außer acht lassen):

NUL	SOH	SOT	EOT	EOX	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

SUB	ESC	FS	GS	RS	EOF		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51

4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E	F	G	H	I	J	K	L	M
52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77

N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	'	a	b	c	d	e	f	g
78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103

h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Was wir hier aufgeschrieben haben, ist der bekannte **ASCII-Zeichensatz** (American Standard Code for Information Interchange), der eine herausragende Rolle bei der Zeichendarstellung auf dem PC spielt. ASCII ist ein 7-Bit-Code und reicht von 00h bis 7Fh, d.h. das Byte, das üblicherweise zur Repräsentation eines Zeichens verwendet wird, wird nur zur Hälfte genutzt. Das High-Bit kann für andere Zwecke verwendet werden, die anwendungs- und auch herstellerbezogen verschieden sein können. U.a. können in der zweiten Hälfte deutsche oder anderssprachige Umlaute dargestellt werden, das kyrillische, griechische oder arabische Alphabet, eine primitive Grafik und vieles mehr. Von diesem Code wird weiter unten noch ausführlicher die Rede sein.

Auf dieser Basis könnte man ein **128-adisches System** definieren und diese Grundzahl rechts durch einen tiefgestellten Index kenntlich machen. Dann würde man zum Beispiel gemäß der obigen Tabelle folgende Zahl bilden können:

$$\mathbf{0X7\$_{128} = 88x128^2 + 55x128 + 36 = 1{,}448{,}868d}$$

Der Tabelle liegt mit Absicht (noch) keine Matrixstruktur zugrunde, sondern es handelt sich um eine rein lineare Aneinanderreihung von Zeichen mit ihren dezimalen Werten. Diese Zeichen, 128 an der Zahl, müsste man ständig im Kopf haben, wenn man mit Zahlenrepräsentationen wie im genannten Beispiel hantieren wollte.

Weiter: Man könnte die Zeichenreihe im Prinzip beliebig nach rechts erweitern und so die unendliche Menge der Natürlichen Zahlen abbilden, wobei wohlgemerkt jede Zahl einer speziellen Ziffer zugeordnet wäre, von denen es dann ebenfalls unendlich viele geben müsste. Dies würde dazu führen, dass wir generell auf eine Stellensystematik verzichten könnten.

Um das einmal überspitzt zu illustrieren, stellen wir uns vor, welche Art von Dialogen auf diese Weise zustande kommen könnten:

- Was ist dein Lieblings-Deo? Ч Kölnisch Wasser. (Ч = „Kolle“ = 4711)

- Kennst du die Loschmidt-Zahl? Ja, sie beträgt Щ. (Щ = „Losse“ = 602300000000000000000000)

- Hast du sie nicht mehr alle? Doch, ich habe alle ƨ Stück. (ƨ = „Pimpel“ = 591827)

Ist dies wohl eine bequeme und übersichtliche Art, mit großen Zahlen umzugehen?

Nein, man hätte zwar den (zweifelhaften) Vorteil, dass man auf Exponentialdarstellungen verzichten könnte, aber so etwas tut natürlich kein anständiger Mathematiker geschweige denn Informationstechniker. Eine solche lineare Aneinanderreihung mit entsprechend vielen Symbolen wäre schlicht und einfach idiotisch.

In Kap. 3 werden wir auf diese Problematik noch einmal zurückkommen.

2 Konversionsvorschriften

Zwar nicht im Alltag, aber umso häufiger im Computerwesen und bei der Programmierung steht man vor der Aufgabe, eine Zahl von einer Darstellung in eine andere umzuwandeln. Der Wert der Zahl bleibt dabei natürlich unverändert. Im Folgenden wird zunächst ein allgemeingültiger Algorithmus vorgestellt, bei dem Quell- und Zielbasis beliebig sind. Sodann werden ein paar Spezialfälle erörtert, die häufig vorkommen und leicht zu bewerkstelligen sind.

2.1 Allgemeines Konversionsverfahren

Im allgemeinen Fall verfahren wir in zwei Schritten, für die wir eine Zwischendarstellung benötigen. Hierfür wird eine Basis verwendet, die besonders bequem und vertraut ist, nämlich am besten wieder einmal das gewohnte Dezimalsystem. (Klarer Fall: Für Computerprogramme wäre das Mittel der Wahl wieder das Binärsystem.)

Es soll eine Zahl mit der Basis **u** in die gleiche Zahl mit der Basis **v** umgewandelt werden. Ein Beispiel folgt weiter unten.

Schritt 1

Im ersten Schritt wird die Quelldarstellung in eine Summe von Potenzen der **Basis u** umgewandelt. Es ist egal, ob man hierfür von links nach rechts oder umgekehrt vorgeht; die Richtung rechts nach links hat den kleinen Vorteil, dass man sich das Abzählen der höchsten Potenz sparen kann, denn diese ergibt sich im Laufe der Summenbildung von selbst.

Wenn eine Ziffernfolge a_n ... a_2 a_1 a_0 gegeben ist, so wird im ersten Schritt also die Summe **S** gebildet (vgl. hierzu Kap. 1.4):

$$\mathbf{S = a_n u^n + ... + a_2 u^2 + a_1 u + a_0}$$

Hierfür müssen vorher die einzelnen Potenzen $\mathbf{u^p}$ von p = 0 bis n errechnet werden. Dabei kann man ganz gut einen Taschenrechner einsetzen, wobei es sich bestätigt, dass das gewohnte Dezimalsystem an dieser Stelle am zweckmäßigsten ist. (Es gibt natürlich auch wissenschaftliche Taschenrechner, die gleich die gesamte Konversion auf Knopfdruck leisten können.)

Schritt 2

Im zweiten Schritt wird nun die Summe **S** in die Zieldarstellung zu **Basis v** überführt. Der Kern des Verfahrens besteht in der schrittweisen **Division durch v mit Restbildung**, wie wir sie vom Schulunterricht her kennen. Es wird zunächst die oben gebildete Summe **S** durch **v** dividiert, danach das ganzzahlige Ergebnis dieser Division, wiederum mit einem ganzzahligen Ergebnis, und so weiter. Bei jeder Division wird der jeweils gebildete Rest notiert, und zwar in der Richtung von rechts nach links. Dieser Rest ist immer eine **Ziffer** mit einem Wert **< v**.

Die Konversion ist abgeschlossen, wenn der ganzzahlige Divisionswert = 0 wird. Die Aneinanderreihung der Reste ist die gesuchte Zahlendarstellung zur **Basis v**.

Beispiel

Eine Zahlendarstellung mit der **Basis 13**, nämlich $\mathbf{10BA7_{13}}$. soll in die **Basis 15** konvertiert werden.

Es werden von rechts nach links die Potenzen von 13 errechnet: 1, 13, 169, 2197 und 28561. Mehr brauchen wir nicht. Sodann wird mit den Koeffizienten 7, A, B, 0 und 1 die Summe **S** gebildet:

```
      7 ·     1 =      7
A (=10) ·    13 =    130
B (=11) ·   169 =   1859
      0 ·  2197 =      0
      1 · 28561 =  28561
                  ------
              S = 30557
```

Diese Summe wird nun schrittweise, wieder von rechts nach links, durch **v = 15** dividiert:

```
30557 : 15 =  2037  Rest:  2   --> Ziffer: 2
 2037 : 15 =   135  Rest: 12   --> Ziffer: C
  135 : 15 =     9  Rest:  0   --> Ziffer: 0
    9 : 15 =     0  Rest:  9   --> Ziffer: 9
```

Es ergibt sich somit die Ziffernfolge **90C2**, die das Resultat zu Basis 15 darstellt.

In der üblichen Kurzform würde man schreiben: $\mathbf{010BA7_{13} = 090C2_{15}}$

2.2 Spezialfall: Binär nach hexadezimal und umgekehrt

Diese beiden Welten ineinander umzuwandeln, ist ganz einfach, weil im Hexadezimalsystem die Binärwelt noch lebendig ist. Man kann sich dabei die Tatsache zunutze machen, dass sich jede Binärkette in Viererportionen von Bits aufteilen lässt, die dann jeweils einer Hexadezimalziffer entsprechen.

Die nachstehende Tabelle enthält alle möglichen 16 Bitgruppen, die auch als „Nibbles" bezeichnet werden (= Halbbytes). Der Dezimalwert ist der Vollständigkeit halber mit aufgeführt.

Bin	Hex	Dez	Bin	Hex	Dez
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	2	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

Auf diese Weise gestaltet sich die Konvertierung sehr einfach:

Beispiel Bin -> Hex gegeben: **10111100110101010011010000 1010**

Diese Kette wird zunächst mit so vielen führenden Nullen versehen, dass ihre Länge durch 4 teilbar wird, und dann in Viererportionen aufgeteilt:

0010 1111 0011 0101 0100 1101 0000 1010

Diese werden sodann gemäß obiger Tabelle zugeordnet:

2	F	3	5	4	D	0	A

0101111001101010100110100001010b = 02F354D0Ah

Beispiel Hex -> Bin gegeben: **FFFF**

Dieses ist die größte mit zwei Bytes darstellbare natürliche Zahl. Sie wird ganz einfach in eine Kette von lauter Einsen umgewandelt (4x4 = 16 an der Zahl):

0FFFFh = 0111111111111111b

Es handelt sich um die Dezimalzahl 65535, die manchmal schlampig als „64K" bezeichnet wird.

Achtung! Es ist nicht die Zahl 6553**6**, die hier vielleicht erwartet wird. Denn dies ist die **Anzahl** der darstellbaren Zahlen, bei denen auch wieder die Null zu berücksichtigen ist.

2.3 Spezialfall: Binär nach dezimal

Diese Konversion ist bereits implizit behandelt worden. Sie entspricht der Summenbildung aus Kap. 1.1. Im dortigen Beispiel führte die Binärkette **011000101** zur Dezimalzahl **197.**

Ganz entsprechend verfährt man bei der Konversion **Hexadezimal nach dezimal** (vgl. Beispiel in Kap. 1.3).

2.4 Spezialfall: Dezimal nach binär

Diese Konversionsrichtung ist eine Spezialisierung und Vereinfachung des in Kap. 2.1 beschriebenen Divisionsrestverfahrens. Vereinfachung deshalb, weil die Division hier lediglich eine fortgesetzte Halbierung beinhaltet.

Wenn eine Dezimalzahl vorgegeben ist, wird sie also Schritt für Schritt **durch 2** dividiert, wobei der Divisionsrest nur 0 oder 1 betragen kann. Aus diesen Restwerten, in der Richtung **rechts nach links**, wird die Binärzahl Ziffer für Ziffer zusammengesetzt.

Als Beispiel nehmen wir wieder die Zahl **197** von oben. Schritt für Schritt ergibt sich folgende Tabelle:

Schritt	Dividend	Division / 2	Rest	Ergebnis
1	197	98	1	1
2	98	49	0	01
3	49	24	1	101
4	24	12	0	0101
5	12	6	0	00101
6	6	3	0	000101
7	3	1	1	1000101
8	1	0	1	**11000101**

Irgendwann ist das **Divisionsergebnis = 0**, und damit ist der Prozess beendet.

Wie man sieht, entspricht das Endergebnis rechts unten der binären Ausgangszahl im Beispiel in Kap. 1.1. Die Schrittnummer entspricht jeweils der dort genannten Positionsnummer.

2.5 Spezialfall: Dezimal nach hexadezimal

Diese Konversion kann aus dem Bisherigen zusammengesetzt werden. Man bildet zunächst die Binärkette gemäß dem vorigen Abschnitt, bildet dann die im Abschnitt davor genannten Vierergruppen und setzt diese in Hexadezimalziffern um. Auf ein erneutes Beispiel kann verzichtet werden.

3 Erweiterte Spinnereien

3.1 Braucht man überhaupt ein „Zahlensystem”?

Wenn man das Binärsystem betrachtet, findet man einen Ziffernvorrat von zwei verschiedenen Zeichen vor, nämlich 0 und 1. Beim Dezimalsystem sind es bereits 10 und beim Hexadezimalsystem 16 an der Zahl.

Es wurde oben schon gezeigt, dass man das Repertoire an Ziffern prinzipiell beliebig erweitern könnte. Wir könnten ein wenig den Allmächtigen spielen (das würden sich bestimmt viele sehr wünschen) und die Menge der Natürlichen Zahlen als eine Folge selbst erfundener Zeichen darstellen. Es wurde auch schon gezeigt, dass dieses Vorhaben ganz einfach idiotisch wäre, schon allein deshalb, weil es gar nicht gelingen kann. Denn wie sollte man in der Praxis die Unendlichkeit abbilden?

In der Tat: Wir „brauchen” streng genommen kein System von Zahlen, aber alle praktischen Gründe sprechen überwältigend dafür. Es gilt nur, einen sinnvollen Kompromiss zu finden zwischen diesem entsetzlichen 0-1-Salat, mit dem Computer gerne arbeiten, und einer Riesenmenge an Zeichen, die nicht mehr handhabbar ist.

3.2 Eine „anständige“ ASCII-Tabelle

Es hat sich die **Basis 16** als ein für den Menschen zweckmäßiger Kompromiss erwiesen. Die **2** wollen wir nach wie vor dem Computer überlassen. Wenn wir diese Erkenntnisse auf den besagten ASCII-Zeichensatz anwenden, so wird dieser in Form einer Matrix dargestellt, die – wie sollte es anders sein – 16 Spalten enthält:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	SOT	EOT	EOX	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	EOF
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Die Positionsangaben an den Rändern sind wie folgt zu lesen: Die vertikale Spalte am linken Rand bedeutet das höherwertige Halbbyte des betr. Zeichens, die obere horizontale Reihe das niederwertige. Die Diagonale reicht also von 00h links oben bis 7Fh rechts unten. Beispielsweise steht das große 'A' an der Position 41 mit dem Dezimalwert 65. Das Elegante an dieser Systematik besteht darin, dass die Position **identisch** ist mit dem Wert des Zeichens. Man weiß sofort, wo man z.B. den Wert 41h zu suchen hat.

Die beiden ersten Reihen des ASCII-Zeichensatzes sind nichtdruckbare Steuerzeichen, die z.B. zum Vorschub des Druckers, Wagenrücklauf oder dergleichen dienen und meist nur noch historische Bedeutung haben. Man muss daran denken, dass dieser Zeichensatz eine sehr alte Tradition hat und früher auch zur Steuerung von Fernschreibern gedient hat. Mit 20h beginnen die wirklich sichtbaren Zeichen; allerdings gilt das ausgerechnet für 20h selbst streng genommen noch nicht, denn hierbei handelt es sich um das Blank (Leerzeichen), das eben nicht sichtbar ist, sondern nur eine Druckstelle besetzt. Weitere wichtige Steuerzeichen (auch heute noch) sind das Escape (1Bh), das Auswirkungen auf die Interpretation des nachfolgenden Zeichens hat, sowie die Zeichen Line Feed (LF = 0Ah) und Carriage Return (CR = 0Dh), die einen Zeilenumbruch definieren. Auch das akustische Signal (Beep = 07h) gehört hierher.

Dem aufmerksamen Leser wird auffallen, dass sich die Großbuchstaben A – Z von den Kleinbuchstaben a – z jeweils durch die Konstante 20h unterscheiden. Dies ermöglicht eine programmtechnisch sehr einfache und schnelle Konversion nach klein durch die OR-Operation **|** 00100000b und umgekehrt die Konversion nach groß durch die AND-Operation &11011111b. Es wird jeweils genau ein Bit ein- bzw. ausgeschaltet.

3.3 Die Bedeutung der Zeichensätze

Die Hexadezimaldarstellung des ASCII-Zeichensatzes, der für den IT-Fachmann und Programmierer unerlässlich ist und ihm früher oder später in Fleisch und Blut übergeht, ist also die bekannteste und wichtigste Variante. Es gibt noch viele andere Zeichensätze, etwa den ANSI-Code (American National Standards Institute), bei dem, wie oben schon angedeutet, das gesamte Byte einschließlich High-Bit genutzt wird. Dieser Code ist jedoch bei weitem nicht so streng normiert wie ASCII, sondern man findet hier eine Unzahl von „Codepages" vor, die sich manchmal nur in Kleinigkeiten unterscheiden. Besondere Bedeutung hat für uns die westeuropäische Codepage 850, die von Microsoft seit MS-DOS genutzt wird und u.a. die deutschen Umlaute enthält.

Auf IBM-Großrechnern wird der EBCDIC-Zeichensatz verwendet (Extended Binary Coded Decimal Interchange Code), der von Anfang an ein 8-Bit-Code war und ebenfalls in verschiedenen Varianten vorkommt. Hierbei wird in vielfältiger Weise auf die Besonderheiten diverser Landessprachen Rücksicht genommen. Was den Programmierer zur Verzweiflung bringen kann, ist, dass EBCDIC keinerlei Ähnlichkeit mit ASCII hat und, besonders wenn Großrechner mit PCs kombiniert werden, ständige Konvertierungen verlangt.

Eine ständig wachsende Bedeutung hat der Unicode, der seit 1991 existiert. Dies ist ein Code, der in den ersten 7 Bits deckungsgleich mit ASCII ist und bei Bedarf (!) über die Grenze des einen Bytes hinweg ausgedehnt werden kann. In der – sehr häufigen – 2-Byte-Form kann der Code 65536 verschiedene Zeichen darstellen, also eigentlich alles, was es überhaupt in der Welt an irgendwie sinnvoller Schrift gibt. Hier kommen alle Kulturen zu ihrem Recht, z.B. das griechische und kyrillische Alphabet und alle Schriftzeichen aus dem arabischen und fernöstlichen Bereich. Wer einmal unter Word die Funktion „Einfügen Sonderzeichen" benutzt hat, kann die Vielfalt erahnen. Wenn die besagten zwei Bytes nicht ausreichen, kann das Repertoire auf bis zu 4 Bytes (32 Bits) ausgeweitet werden, mit einer nicht mehr nennenswerten Grenze von Möglichkeiten. Solche Zeichensätze kommen dann eigentlich schon der eingangs beschriebenen Idiotenreihe nahe (wer sollte sich vier Milliarden Zeichen merken können?).

Zu erwähnen ist, dass der Unicode Bestandteil von DB2 von IBM ab Version 7 geworden ist und damit Eingang in eines der wichtigsten Datenbanksysteme der Welt gefunden hat.

4 Schlusswort

Was wir hier betrieben haben, ist ein Paradebeispiel der Angewandten Informatik. Die Betrachtung der Menge der Natürlichen Zahlen, **N**, gibt nicht allzu viel her und ist eher trockener Stoff. Natürlich kann man sich mit den Primzahlen, den Fibonacci-Zahlen und allen möglichen anderen Folgen und Reihen beschäftigen, man kommt aber nicht in die Nähe der praktischen Probleme, die der Umgang mit Computeranwendungen mit sich bringt.

Eine wesentliche Besonderheit scheint mir die Behandlung der Unendlichkeit zu sein. In der reinen und theoretischen Mathematik ist das bei der Menge der Natürlichen Zahlen ein Kinderspiel. Es gelten hier zwei Grundsätze:

1. Es gibt ein **Anfangselement** ohne Vorgänger (0 oder 1, je nachdem, ob die Null mit zu **N** gerechnet wird oder nicht)
2. Es gilt das Prinzip der **Vollständigen Induktion**, d.h. der Schluss von n auf n+1. (Genauer: Wenn sich nachweisen lässt, dass bei einer Menge M eine Eigenschaft für das Element n gilt und daraus folgt, dass diese auch für das Element (n+1) gilt, dann ist M die Menge der Natürlichen Zahlen).

Mit diesen Basisüberlegungen ist der Mathematiker fertig. Er kümmert sich nicht darum, dass wir Menschen keine Chance haben, den Begriff der Unendlichkeit zu verinnerlichen, und auch nie eine Chance haben werden. Mir selbst wird es beim Versuch, mir die Unendlichkeit vorzustellen, immer schwindlig. Es gibt zum Beispiel doppelt so viele Natürliche wie Gerade Zahlen, und dennoch sind beide Mengen unendlich und „gleich mächtig". Bei solchen Aussagen versagt meine Kognitionsfähigkeit, und ich glaube, dass es allen Menschen so geht. Wir sind – zumindest auf unserem Evolutionsniveau – nicht in der Lage, das Phänomen der Unendlichkeit richtig zu verstehen.

Der Pragmatiker der Angewandten Informatik plagt sich nicht mit solchen Verständnisproblemen. Für ihn **gibt es einfach keine Unendlichkeit** (genauso wenig, wie es überhaupt in der Natur eine Größe gibt, die wirklich „unendlich" klein oder groß wäre). Vielmehr geht es ihm darum, große, vielleicht echt megariesenhafte Zahlen so darzustellen, dass wir Menschen einigermaßen damit umgehen können. Dafür eignet sich weder die Binärschreibweise (diese ist wiederum für Maschinen das Richtige) noch die wiederholt erwähnte lineare „Idiotenreihe", bei der jede Zahl ihre eigene Ziffer hat. Sondern es gilt, einen sinnvollen Kompromiss zu finden.

Als solcher hat sich die Hexadezimaldarstellung bestens bewährt. So sagen die Informatiker und sind zufrieden.

Das ist Pragmatismus pur.

Ich liebe ihn.