D U A L - S Y S T E M

VHS-Kurs

DOS für Einsteiger

von Uwe Koch

Zur Darstellung von Zahlen gibt es verschiedene Zahlensysteme. So unterscheidet man zunächst zwischen Additionssystemen und Stellenwertsystemen. Bei Additionssystemen haben einzelne Ziffern einen festen Wert und werden zur Bildung von Zahlen addiert, wie z.B. bei den römischen Zahlen. Dort hat *X* den Zahlenwert *10* und die Zahl *30* wird durch dreimaliges Addieren von 10 dargestellt, also *XXX*. Bei den Stellenwertsystemen existieren verschiedene Ziffern, deren Wert nicht nur durch die Ziffer, sondern auch durch die Stelle der Ziffer innerhalb der Zahl bestimmt wird. In unserem Dezimalsystem hat die Ziffer *3* also den Wert *drei*, wenn sie als letzte Ziffer vor dem Komma erscheint, und den Wert *dreißig*, wenn sie als vorletzte Ziffer erscheint, u.s.w..
Normalerweise rechnen wir mit dem Dezimalsystem, das heißt, wir verwenden Zahlen, die aus den Ziffern *0,1,2,3,4,5,6,7,8* und *9* zusammengesetzt sind. Es existieren also insgesamt zehn Ziffern. Eine Zahl besteht aus mehreren Ziffern, deren Positionen innerhalb der Zahl den Wert der Stelle darstellen. Dieses System heißt Dezimalsystem, weil es zehn verschiedene Ziffern hat, und die einzelnen Stellen der Zahl Wertigkeiten haben, die Potenzen von Zehn sind. Das heißt, die erste Stelle hat den Wert *Eins*, die zweite den Wert *Zehn*, die dritte den Wert *Hundert* und so weiter. Die Zahl *fünftausenddreihundertundzwölf* wird also dargestellt als :

$$5 * 1000 + 3 * 100 + 1 * 10 + 2 * 1$$

Dieses kann man mathematisch auch so darstellen :

$$5 * 10^3 + 3 * 10^2 + 1 * 10^1 + 2 * 10^0$$

Ebenso läßt sich die Zahl *dreikommasiebenfünf* so darstellen :

$$3 * 1 + 7 * 0.1 + 5 * 0.01$$

oder als :

$$3 * 10^0 + 7 * 10^{-1} + 5 * 10^{-2}$$

Wie man sieht lassen sich alle Stellen einer Zahl als Faktor für die jeweilige Zehnerpotenz darstellen. Dieses Zehnersystem hat sich wegen seiner Übersichtlichkeit und der Möglichkeit die Hände als 'Rechenmaschine' zu verwenden bei uns eingebürgert.

Aber vor nicht allzulanger Zeit war auch das Zwölfersystem noch gebräuchlich (Duzend, Gros usw.). In einigen Ländern werden auch heute noch Maße durch andere Zahlensysteme dargestellt. (Z.B. in Amerika die Maßeinheiten für Flüssigkeiten : Gallonen, Quarts, Pints usw. die alle nicht im Zehnersystem liegen.). Daraus ist zu erkennen, daß das Dezimalsystem zwar recht nützlich und einfach ist, jedoch ist es nicht die einzige Art Zahlen darzustellen.

Besonders für Computer ist es recht ungünstig mit dem Dezimalsystem zu arbeiten. Ein Computer besteht ja bekanntlich aus elektrischen Schaltkreisen, und kann daher auch Zahlen nur durch elektrische Werte darstellen. Dabei wäre es zwar möglich, die Ziffern des Dezimalsystems z.B. durch zehn verschiedene Spannungen darzustellen, aber wegen der Unzulänglichkeiten der elektrischen Schaltkreise wären damit keine genauen Berechungen möglich.
Deshalb arbeiten Computer nach einem anderen Verfahren. Sie verwenden nur die beiden Werte *Spannung an* und *Spannung aus*. Hierbei können keine Ungenauigkeiten auftreten. Allerdings läßt sich nun auch das Dezimalsystem nichtmehr verwenden. Im Computer existieren also nur zwei Zustände, *Spannung an* und *Spannung aus*. Diese beiden Zustände kann man auch durch die beiden Ziffern *0* und *1* darstellen. Wir haben beim Dezimalsystem gesehen, daß bei zehn Ziffern die einzelnen Stellen eine Wertigkeit von Zehnerpotenzen hatten. Hier haben wir nur zwei Ziffern und brauchen nun Wertigkeiten von Zweierpotenzen. D.h. die letzte Ziffer vor dem Komma hat den Wert *1*, die vorletzte den Wert *2*, die nächste *4*, die nächste *8* und so weiter. Zur besseren Übersichtlichkeit kann man die Ziffern in ein Schema eintragen.

	128	64	32	16	8	4	2	1			
1.	0	1	0	1	1	0	0	1	=	64+16+8+1	= 89
2.	1	0	1	0	0	1	1	0	=	128+32+4+2	= 166
3.	0	0	0	0	0	0	0	0	=	0+...+0+0	= 0
4.	1	1	1	1	1	1	1	1	=	128+64+32+16+ 8+4+2+1	= 255

Für diese Beispiele habe ich Zahlen im Zweiersystem (Dualsystem) verwendet, die acht Stellen haben. Eine einzelne Stelle nennt man *Bit*. Eine Kombination aus acht Bit nennt man *Byte*. Die obigen Beispiele bestehen also aus einem Byte. Man sieht auch, daß die kleinste Zahl, die man mit einem Byte darstellen kann die *Null* ist (Beispiel 3). Die größte Zahl ist in Beispiel 4 dargestellt, nämlich die *255*. Viele Arbeitsplatz-Rechner arbeiten intern mit Bauteilen, die jeweils ein Byte in einem Arbeitsgang bearbeiten können. Der Computer kann also zunächst nur Zahlen zwischen *0* und *255* bearbeiten. Alle anderen Rechnungen im Computer werden durch Programme im Betriebssystem so umgeformt, daß nur Werte in diesem Bereich vorkommen. Für manche Dinge ist allerdings dieser Zahlenbereich zu klein, zum Beispiel für die interne Speicherverwaltung. Selbst einfachste Spielcomputer wie der C 64 haben wenigstens 64 Kilobyte Speicher. Das bedeutet, daß in diesen Rechnern 64 * 1024 (65536) Speicherplätze für je ein Byte vorhanden sind. Diese Speicherplätze nehmen immer nur ein Byte auf. Daher müssen sämtliche Informationen so dargestellt werden, daß sie in ein Byte passen. Da heißt, der Computer muß sich intern auch alle Buchstaben und Grafik-Symbole in einem Byte merken. Dazu hat das amerikanische Normungsinstitut eine Nummerierung der einzelnen Zeichen festgelegt, den sogenannten ASCII-Code. (Siehe Anhang)

Aber zurück zur Speicherverwaltung. Diese 65536 Speicherplätze haben alle eine Nummer von *0* bis *65535*. Der Mikroprozessor (die eigentliche Rechenschaltung) muß die Nummer des Speicherplatzes an den Speicher ausgeben, wenn er einen Zahlenwert aus diesem Speicher haben will. Dafür muß er auch Zahlen in diesem Bereich berechnen können. Die Zahl 65535 läßt sich aber durch zwei Bytes darstellen, d.h. durch 16 Bits.Die Mikroprozessoren in Heim- und kleinen Personalcomputern arbeiten mit 8 Bit für Daten und 16 (+ 4) Bit für Adressen. Wenn man in Maschinensprache programmieren will, oder in Basic die speziellen Fähigkeiten des Rechners (Musik, Grafik, Tricks usw.) nutzen will, dann wird die Verwendung des Dualsystems mit den vielen Nullen und Einsen bald unübersichtlich und führt leicht zu Fehlern. Daher faßt man oft vier Dualziffern zu einer *Hexadezimalziffer* zusammen. Vier Dualziffern können die Zahlen *0* bis *15* darstellen.

8	4	2	1		
0	0	0	0	=	0
0	0	0	1	=	1

:	:	:	:	:	:
1	1	1	1	=	15

Diese Zahlen von 0 bis 15 stellt man dann durch 16 Ziffern dar. Unser Dezimalsystem hat aber nur die zehn Ziffern von *0* bis *9*, daher nimmt man für die Ziffer *10* (keine Zahl !) den Buchstaben *A*, für die *11* den Buchstaben *B* usw. bis *15 = F*. So entstehen also die sechzehn Ziffern *0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F* des Hexadezimalsystems. Hieraus kann man ebenfalls alle Zahlen als Faktoren von Sechzehnerpotenzen darstellen.

16^3	16^2	16^1	16^0		
4096	256	16	1		
0	0	0	0	=	0
0	0	1	0	=	16
0	0	A	2	=	162
1	0	1	0	=	4112
1	1	1	1	=	4369
F	F	F	F	=	65535

Zwei Byte mit sechzehn Dualstellen werden also durch eine vierstellige Hexadezimalzahl dargestellt, ein Byte entsprechend durch eine zweistellige Hexadezimalzahl. Die Dualzahlen werden nur verwendet, wenn man wirklich die einzelnen Bits erkennen will. Normalerweise werden die Hexadezimalzahlen verwendet, die manchmal auch als *Sedezimalzahlen* bezeichnet werden. Damit man Hexadezimalzahlen von Dezimalzahlen unterscheiden kann, werden sie häufig durch ein vorangestelltes Dollarzeichen oder ein angehängtes *h* gekennzeichnet. Bei gemischter Verwendung der Zahlensysteme gibt es dann das angehängte *d* für Dezimalzahlen und das *b* für Binärzahlen, also Dualzahlen.

			h		b		d
z.B.:	$5A	=	5A	=	01011010	=	90

Mit all diesen Zahlensystemen kann man natürlich auch rechnen.Dabei werden dieselben Rechenregeln angewendet wie bei den Dezimalzahlen. D.h. es entstehen genauso Überträge in die höheren Stellen usw.. Beim Addieren von Dualzahlen werden die Zahlen am Besten untereinander geschrieben und dann Stelle für Stelle addiert. Dabei gilt:

0	+	0	=	0	
0	+	1	=	1	
1	+	0	=	1	
1	+	1	=	0	und die 1 in die nächste Stelle

			b			d
z.B.			01010101			85
			b			d
		+	00110011		+	51
			b			d
	Übertrag:		11101110			00
			---			---
			b			d
			10001000			136

Genauso funktioniert das Addieren von Hexadezimalzahlen:

			h			d
z.B.			0D			13
			h			d
		+	1E		+	30
			h			d
	Übertrag:		10			00
			---			---
			h			d
			2B			43

Schwieriger wird schon das Subtrahieren, denn wir haben ja bisher immer nur Zahlen

zwischen 0 und 255 für ein Byte gehabt. In diesem Zahlenbereich können aber keine negativen Zahlen dargestellt werden, und daher wäre z.B. die Rechnung *3 - 5* nicht möglich. Da man aber häufig subtrahieren muß, haben sich die ersten Anwender des Dualsystems mehrere Abhilfemöglichkeiten überlegt.

Zunächst gibt es die Möglichkeit, von den 8 Bit eines Bytes nur die hinteren 7 zur Darstellung des Zahlenwertes zu verwenden und die erste Stelle als Vorzeichen. Es wären dann Zahlen im Bereich von -127 bis +127 möglich, denn mit 7 Bits läßt sich als größte Zahl die 127 darstellen. Eine Null als erste Stelle würde dann bedeuten, daß die Zahl positiv ist und eine Eins als erstes Bit wäre das negative Vorzeichen. Damit wäre jede Zahl zwischen 0 und 127 sowohl als positive als auch als negative Zahl möglich. Allerdings leider auch die Null, d.h. es gäbe *+0* und *-0*.

0	1	1	1	1	1	1	1	127
0	0	1	1	1	1	1	1	63
0	0	0	0	0	1	1	1	7
0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	-0
1	0	0	0	0	0	0	1	-1
1	0	0	0	0	0	1	0	-2
1	0	0	0	0	1	1	1	-7
1	0	1	1	1	1	1	1	-63
1	1	1	1	1	1	1	1	-127

Dabei würden große Schwierigkeiten beim Subtrahieren auftreten, da jetzt eine Null zuviel zwischen den positiven und den negativen Zahlen liegt. Daher verwendet man diese Methode nicht so, sondern ändert sie zunächst noch etwas. Dazu vertauscht man bei den negativen Zahlen die Nullen und die Einsen für den Zahlenwert, aber nicht beim Vorzeichen.

aus	1	0	0	0	0	0	0	1	für	-1
wird	1	1	1	1	1	1	1	0	für	-1

Das allein behebt das Problem mit der doppelten Null noch nicht. Man muß zu der so entstandenen Zahl, dem sogenannten 'Einerkomplement', noch eine 1 addieren und erhält dann das Zweierkomplement.

aus	1	1	1	1	1	1	1	0	für	-1
wird	1	1	1	1	1	1	1	1	für	-1

Damit sieht die Zahlenreihe wie folgt aus :

0	1	1	1	1	1	1	1	127
0	0	1	1	1	1	1	1	63
0	0	0	0	0	1	1	1	7
0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	-1
1	1	1	1	1	1	1	0	-2
1	1	1	1	1	0	0	1	-7
1	1	0	0	0	0	0	1	-63
1	0	0	0	0	0	0	1	-127
1	0	0	0	0	0	0	0	-128

Wie man sieht, ist jetzt die negative Null verschwunden und dafür eine weitere negative Zahl, die -128, hinzugekommen. Außerdem ist die Zahl -128 im vorzeichenlosen Dualsystem genau um einen Wert größere als +127. Man muß also immer wissen, welches Zahlensystems

verwendet wird. Der C64 arbeitet bei einstelligen Bytes fast ausschließlich mit vorzeichenlosen Dualzahlen, d.h. es gibt nur Zahlen zwischen 0 und 255. Bei den Speicheradressen verwendet der C64 beide Systeme gemischt. Eingaben vom Benutzer werden als vorzeichenlose Werte zwischen 0 und 65535 entgegengenommen, Ausgaben werden aber im Zweierkomplement dargestellt. Das heißt die Zahlen liegen zwischen -32768 und 32767.Wenn man bei dem Befehl 'PRINT FRE(0)' ein negatives Ergebnis erhält, muß man zu dem Wert noch 65536 addieren um das richtige Ergebnis zu erhalten. Durch die Darstellung im Zweierkomplement geht der Zahlenbereich zwar von +127 direkt nach -128, bzw. von +32767 nach-32768 bei Doppelbytes, über, dafür ist jetzt aber die Subtraktion sehr einfach geworden. Um eine Zahl von eine anderen zu subtrahieren, wird einfach der negative Wert nach den bekannten Regeln addiert.

1.Beispiel: 6 - 3 wird umgewandelt in 6 + (-3)

		0	0	0	0	0	1	1	0	6
	+	1	1	1	1	1	1	0	1	-3
Übertrag:		1	1	1	1	1	1	1		
		---	---	---	---	---	---	---	---	---
Ergebnis:	(1)	0	0	0	0	0	0	1	1	3

2.Beispiel: 5 - 7 wird umgewandelt in 5 + (-7)

		0	0	0	0	0	1	0	1	5
	+	1	1	1	1	1	0	0	1	-7
Übertrag:								1		
		---	---	---	---	---	---	---	---	---
Ergebnis:		1	1	1	1	1	1	1	0	-2

In Beispiel 1. entsteht ein Übertrag zur neunten Stelle, (in Klammern), der aber direkt bei der Berechnung ignoriert wird, da der Prozessor nur die hinteren acht Bit des Ergebnisses abspeichert.

Die Multiplikation von Dualzahlen ist etwas umständlich und wird auch nur in Maschinenprogrammen benötigt, da der Basicinterpreter diese Aufgaben für uns übernimmt, wenn wir in Basic programmieren. Daher möchte ich die Multiplikation hier nicht weiter erklären.

Es gibt aber noch zwei andere Verknüpfungen bei Dualzahlen, außer Addition/Subtraktion und Multiplikation/Division. Das sind die 'UND - Verknüpfung' und die 'ODER - Verknüpfung'.

Bei der UND-Verknüpfung zweier Dualzahlen steht im Ergebnis nur an den Stellen eine 1 , an denen in der ersten UND in der zweiten Zahl eine 1 steht.

Beispiel:		1	0	0	1	1	0	0	1	153
	UND	1	0	1	0	1	0	1	0	UND 170
		---	---	---	---	---	---	---	---	---
		1	0	0	0	1	0	0	0	136

Man erkennt, daß eine solche Rechnung für Dezimalzahlen nicht sinnvoll ist, da nicht ersichtlich ist, warum eine Verknüpfung von 153 mit 170 als Ergebnis 136 hat. Andererseits kann man bei Dualzahlen leicht feststellen, welche Stellen zweier Zahlen gleich sind.

Bei der ODER-Verknüpfung erscheint im Ergebnis an den Stellen eine 1, an denen entweder in der ersten ODER in der zweiten Zahl eine 1 steht.

Beispiel:		1	1	0	0	1	0	0	1	201
	ODER	1	0	1	0	1	0	1	1	ODER 170
		---	---	---	---	---	---	---	---	---
		1	1	1	0	1	0	1	1	235

Auch hier ist eine solche Verknüpfung für Dezimalzahlen nicht sinnvoll. Bei Dualzahlen steht aber im Ergebnis eine 1 an allen Stellen, an denen bei einer von beiden Zahlen eine 1 stand, bzw. es steht im Ergebnis eine 0 an allen Stellen, an denen eine 0 in beiden Zahlen stand.

Man erkennt daran, daß die UND-Verknüpfung für Einsen einer ODER-Verknüpfung für Nullen entspricht und umgekehrt. Diese beiden Verknüpfungen braucht man z.B. wenn man nur ein einzelnes Bit innerhalb eines Bytes verändern will. Dieses kommt bei hochauflösender Grafik und bei Sound-Programmen sowie bei der Verwendung des USER - Ports vor, da hier einzelne Bits eine eigene besondere Bedeutung haben.

Beispiel: Es soll nur das 3. Bit in dem Byte auf Eins gesetzt werden, unabhängig von den anderen Bits.

Byte vorher:	1	0	0	0	1	0	1	1	
ODER	0	0	0	0	0	1	0	0	(hier 3. Bit Eins)
Byte nachher:	1	0	0	0	1	1	1	1	

Beispiel: Es soll nun das 2. Bit in dem Byte auf Null gesetzt werden, unabhängig von den anderen Bits.

Byte vorher:	1	0	0	0	1	1	1	1	
UND	1	1	1	1	1	1	0	1	(hier 2. Bit Null)
Byte nachher:	1	0	0	0	1	1	0	1	

Es gibt noch mehr solcher sogenannter 'logischer' Verknüpfungen, die aber beim Programmieren in Basic nicht vorkommen. Die beiden gerade verwendeten Formen gibt es allerdings auch in Basic. Dort heißt die UND-Verknüpfung 'AND' und die ODER-Verknüpfung 'OR'. Diese beiden Befehle kann man in unterschiedlichen Zusammenhängen gebrauchen. Z.B. bei :

```
IF (A<5) AND (A>2) THEN ...
```

Hier wird zuerst '(A<5)' kontrolliert und das Ergebnis in einem Byte gespeichert. Dabei bedeutet JA = 00000000 und NEIN = 11111111. Dann wird '(A>2)' kontrolliert und das Ergebnis in einem weiteren Byte vermerkt. Danach werden die beiden Bytes mit UND verbunden und das Ergebnis für die IF-Bedingung verwendet.

Beispiel: A = 3

Dann ergibt (A<5) ein JA also 00000000
und (A>2) ergibt auch JA also 00000000

Jetzt kommt die UND-Verknüpfung:

00000000
UND 00000000
00000000 Ergebnis JA für die IF-Anweisung

Beispiel: A = 1

Dann ergibt (A<5) wieder ein JA also 00000000
aber (A>2) ergibt ein NEIN also 11111111

Die UND-Verknüpfung ergibt :

00000000
UND 11111111
11111111 Ergebnis NEIN für die IF-Anweisung

Entsprechendes gilt für die OR bzw. ODER - Anweisung.

A n h a n g :
==========

Zusammenfassung der ASCII-Zeichen nach der amerikanischen Norm:

Nummer	Zeichen	Nummer	Zeichen	Nummer	Zeichen	Nummer	Zeichen
0/00	(NUL)	32/20	(Space)	64/40		96/60	'

1/01	(SOH)	33/21	!	65/41	A	97/61	a
2/02	(STX)	34/22	"	66/42	B	98/62	b
3/03	(ETX)	35/23	#	67/43	C	99/63	c
4/04	(EOT)	36/24	$	68/44	D	100/64	d
5/05	(ENQ)	37/25	%	69/45	E	101/65	e
6/06	(ACK)	38/26	&	70/46	F	102/66	f
7/07	(BEL)	39/27	'	71/47	G	103/67	g
8/08	(BS)	40/28	(	72/48	H	104/68	h
9/09	(HT)	41/29	)	73/49	I	105/69	i
10/0A	(LF)	42/2A	*	74/4A	J	106/6A	j
11/0B	(VT)	43/2B	+	75/4B	K	107/6B	k
12/0C	(FF)	44/2C	,	76/4C	L	108/6C	l
13/0D	(CR)	45/2D	-	77/4D	M	109/6D	m
14/0E	(SO)	46/2E	.	78/4E	N	110/6E	n
15/0F	(SI)	47/2F	/	79/4F	O	111/6F	0
16/10	(DLE)	48/30	0	80/50	P	112/70	p
17/11	(DC1)	49/31	1	81/51	Q	113/71	q
18/12	(DC2)	50/32	2	82/52	R	114/72	r
19/13	(DC3)	51/33	3	83/53	S	115/73	s
20/14	(DC4)	52/34	4	84/54	T	116/74	t
21/15	(NAK)	53/35	5	85/55	U	117/75	u
22/16	(SYN)	54/36	6	86/56	V	118/76	v
23/17	(ETB)	55/37	7	87/57	W	119/77	w
24/18	(CAN)	56/38	8	88/58	X	120/78	x
25/19	(EM)	57/39	9	89/59	Y	121/79	y
26/1A	(SUB)	58/3A	:	90/5A	Z	122/7A	z
27/1B	(ESC)	59/3B	;	91/5B	Ä	123/7B	ä
28/1C	(FS)	60/3C	<	92/5C	Ö	124/7C	ö
29/1D	(GS)	61/3D	=	93/5D	Ü	125/7D	ü
30/1E	(RS)	62/3E	>	94/5E	^	126/7E	ß
31/1F	(US)	63/3F	?	95/5F	_	127/7F	(Del)