Dualzahlen (1/3)

- Bitfolgen auch als Zahlenwerte auffassen:
  $1_b = 1$, $10_b = 2$, $11_b = 3$, $100_b = 4$, $101_b = 5$ ...
- **Dualzahlen:** Zahlensystem, das zur Darstellung von Zahlen nur die Ziffern 0 und 1 verwendet. Auch **Binärzahlen** genannt
- Rechnen mit Dualzahlen funktioniert wie mit normalen Zahlen. Hier:
  - Addieren + Subtrahieren
  - Multiplizieren

Dualzahlen (2/3)

- Addieren:

```
  001001001
+ 010101110
      1        Übertrag
------------
= 011110111
```

- Subtrahieren:

```
  011010111
- 010101110
    1 1        Übertrag
------------
= 000101001
```


Dualzahlen (3/3)

- Multiplizieren

```
101001 x 1001
-------------
      101001
     000000
    000000
   101001
      1        Übertrag
-------------
   101110001
```

Probe:

$101001_b = 41$
$1001_b = 9$
41 x 9 = 369
$101110001_b = 369$

Umrechnung dual ↔ dezimal (1/3)

- Grundlage: Wie funktionieren unsere Zahlen?
- $4982 = 4 \times 1000 + 9 \times 100 + 8 \times 10 + 2 \times 1$
  $= 4 \times 10^3 + 9 \times 10^2 + 8 \times 10^1 + 2 \times 10^0$
  (von rechts nach links: Einser, Zehner, Hunderter, Tausender, ...)
- Bei Dualzahlen geht das genauso:
- $10011_b = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
  $= 1 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$
  $= 19$

Umrechnung dual ↔ dezimal (2/3)

umgekehrt: dezimal → dual etwas komplizierter

49 als Dualzahl?

49	: 2 = 24,	Rest	1	Einser
24	: 2 = 12,	Rest	0	Zweier
12	: 2 = 6,	Rest	0	Vierer
6	: 2 = 3,	Rest	0	Achter
3	: 2 = 1,	Rest	1	
1	: 2 = 0,	Rest	1	

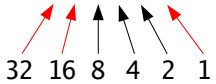
-> 110001_b (32+16+1)

Warum geht das? Rechnen Sie mal direkt

$110001_b : 2...$ $\quad 110001_b = 2 \times 11000_b + 1$

Umrechnung dual ↔ dezimal (3/3)

- oder durch „Kopfrechnen“:
  - kleinste 2er-Potenz (2, 4, 8, 16, ...), die in die Zahl 49 „passt“: 32 (nächst größere: 64)
  - also 49 = 32 + ? = 32 + 17
  - 17 = 16 + ? = 16 +1
  - Damit: 49 = 32 + 16 + 1
    = 100000 + 010000 + 000001
    = 110001



Überblick: Bit, Byte, (Doppel-) Wort

Einheit	Erklärung	Werte	Anzahl Werte
Bit	Binary Digit	0, 1	2
Byte	8 Bit	0 – 255	256
Wort	2 Byte, 16 Bit	0 – 65535	65536
Doppelwort	2 Worte (4 Byte)	0 – 4294967296	4294967296

Kodierung (1/3)

- Was der Computer gut kann:
  Bits und Bytes speichern; auch (Doppel-) Worte und längere Bitfolgen (mehrere Bytes verwenden)
- Problem: allgemeine Daten speichern
- Lösung: „beliebige“ Daten auf Bitfolgen abbilden

Kodierung (2/3)

- Beispiel: „A-Z“ als 1-26 speichern, binär:
  A = 00001, B = 00010, ...

								unbenutzt
A	00001	H	01000	O	01111	V	10110	00000
B	00010	I	01001	P	10000	W	10111	11011
C	00011	J	01010	Q	10001	X	11000	11100
D	00100	K	01011	R	10010	Y	11001	11101
E	00101	L	01100	S	10011	Z	11010	11110
F	00110	M	01101	T	10100			11111
G	00111	N	01110	U	10101			

- Also 5 Bit pro Buchstabe
- HALLO → 01000 00001 01100 01100 01111

Kodierung (3/3)

- Praxis: „nur Großbuchstaben“ unbrauchbar
- Kleinbuchstaben, Zahlen, Sonderzeichen
- erster Standard: ASCII, enthält die Zeichen:
  0-9, A-Z, a-z, das Leerzeichen „ “ und:
  `!"#$%&\'()*+,-./:;<=>?@[\]^_`{|}~`

ASCII-Tabelle (1/2)

- **ASCII** = **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange:

32		48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	

ASCII-Tabelle (2/2)

- ASCII – und was ist mit Umlauten?
  - „einfach drauf verzichten“ – ae, oe, ue, Ae, Oe, Ue und Ss stoeren hoechstens die Uebergenauen
  - Zeichen aus der Zeichentabelle werfen und durch Umlaute ersetzen, etwa
    [→ ä,] → ö, \ → ü,
    { → Ä, } → Ö, | → Ü usw.
  - ASCII verwendet nur 7 Bit (0-127); über achtes Bit neuen Zeichensatz mit Sonderzeichen definieren, z. B. ISO-8859-15 (Westeuropa mit €-Zeichen). Dort: 0-127 wie ASCII, 128-255 Zusatzzeichen

ISO-8859-15

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
NBSP	¡	¢	£	€	¥	Š	§	š	©	ª	«	¬	SHY	®	¯
°	±	²	³	Ž	µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Quelle: http://de.wikipedia.org/wiki/ISO_8859-15

Oktal- und Hexadezimalzahlen (1/9)

- Noch zwei weitere Zahlensysteme
- Bisher:
  - Dualzahlen – Basis 2
  - Dezimalzahlen – Basis 10
- Jetzt:
  - Oktalzahlen – Basis 8
  - Hexadezimalzahlen – Basis 16

Oktal- und Hexadezimalzahlen (2/9)

Oktalzahlen

- Basis 8, also 8 Ziffern: 0, 1, 2, 3, 4, 5, 6, 7
- $7_o + 1_o = 10_o$

oktal	dez.	binär	oktal	dez.	binär
0	0	0	14	12	1100
1	1	1	15	13	1101
2	2	10	16	14	1110
3	3	11	17	15	1111
4	4	100	20	16	1000
5	5	101	...		
6	6	110	77	63	111111
7	7	111	100	64	1000000
10	8	1000	101	65	1000001
11	9	1001	102	66	1000010
12	10	1010	103	67	1000011

Oktal- und Hexadezimalzahlen (3/9)

Hexadezimalzahlen

- Basis 16, also 16 Ziffern:
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  10 11 12 13 14 15
- $9_h + 1_h = A_h$, $F_h + 1_h = 10_h$ (mit Wert 16)

Oktal- und Hexadezimalzahlen (4/9)

hex.	dez.	binär	hex.	dez.	binär	hex.	dez.	binär
0	0	0	10	16	10000	20	32	100000
1	1	1	11	17	10001	21	33	100001
2	2	10	12	18	10010	22	34	100010
3	3	11	13	19	10011	23	35	100011
4	4	100	14	20	10100	24	36	100100
5	5	101	15	21	10101	25	37	100101
6	6	110	16	22	10110	26	38	100110
7	7	111	17	23	10111	27	39	100111
8	8	1000	18	24	11000	...		
9	9	1001	19	25	11001	FF	25	11111111
A	10	1010	1A	26	11010	100	26	100000000
B	11	1011	1B	27	11011	101	27	100000001
C	12	1100	1C	28	11100	102	28	100000010
D	13	1101	1D	29	11101	...		
E	14	1110	1E	30	11110	FFF	30	111111111111
F	15	1111	1F	31	11111	1000	31	1000000000000

Oktal- und Hexadezimalzahlen (5/9)

- Umrechnen Dual ↔ Oktal
  und Dual ↔ Hexadezimal
  ist also leicht:
- Jede Ziffer einer Oktalzahl wird zu drei Bits:
  $307_o = 011\ 000\ 111_b$
- Jede Ziffer einer Hex.-Zahl wird zu vier Bits:
  $3AF_h = 0011\ 1010\ 1111_b$

Oktal- und Hexadezimalzahlen (6/9)

Umrechnen in Dezimalzahlen: Wie bei Dualzahlen, aber mit anderer Basis

- $1101_b = 1 \times \underline{2}^3 + 1 \times \underline{2}^2 + 0 \times \underline{2}^1 + 1 \times \underline{2}^0 = 13$
- $1734_o = 1 \times \underline{8}^3 + 7 \times \underline{8}^2 + 3 \times \underline{8}^1 + 4 \times \underline{8}^0 = 988$
- $1A3F_h = 1 \times \underline{16}^3 + 10 \times \underline{16}^2 + 3 \times \underline{16}^1 + 15 \times \underline{16}^0 = 6719$
- zur Erinnerung im Dezimalsystem:
  $3921 = 3 \times 10^3 + 9 \times \underline{10}^2 + 2 \times \underline{10}^1 + 1 \times \underline{10}^0 = 6719$

Oktal- und Hexadezimalzahlen (7/9)

Umrechnen von Dezimal- in Oktal- oder Hexadezimalzahlen:

- entweder in zwei Schritten – erst in Dualzahlen umrechnen, dann 3er- bzw. 4er-Gruppen zusammenfassen:
  $80 = 001\ 010\ 000_b = 120_o = 0101\ 0000_b = 50_h$
- oder durch schriftliches Dividieren wie bei Umrechnen in Dualzahl, aber diesmal mit Divisor 8 oder 16

Oktal- und Hexadezimalzahlen (8/9)

942 als Hexadezimalzahl?

942 : 16 = 58,	Rest	14 (E)	Einser	(16°)
58 : 16 = 3,	Rest	10 (A)	16er	(16^1)
3 : 16 = 0,	Rest	3	256er	(16^2)

→ $3AE_h$ (3 x 256 + 10 x 16 + 14 x 1)

Warum geht das? Rechnen Sie mal direkt

$3AE_h$: 16... $3AE_h = 16 \times 3A_h + E_h$

Oktal- und Hexadezimalzahlen (9/9)

942 als Oktalzahl?

942 : 8 = 117,	Rest	6	Einser	(8°)
117 : 8 = 14,	Rest	5	8er	(8^1)
14 : 8 = 1,	Rest	6	64er	(8^2)
1 : 8 = 0,	Rest	1	512er	(8^3)

→ 1656_o (1 x 512 + 6 x 64 + 5 x 8 + 6 x 1)

Wie bei Oktalzahlen: Rechnen Sie mal direkt

1656_o : 8... $1656_o = 8 \times 165_o + 6_o$

Warum hexadezimal? (1/3)

- deckt praktische (häufig benutzte) Bereiche ab
- z. B.: Home-Computer mit 64 KByte RAM
  64 KByte = 64 x 1024 Byte = 65 536 Byte
  → hexadezimal: 10000_h Byte
  also Adressen: 0000_h – $FFFF_h$
- z. B.: PC mit 1 GByte RAM
  1 GByte = 1024 x 1024 x 1024 Byte
  = 1 073 741 824 Byte
  → hexadezimal: 40000000_h Byte
  also Adressen: 00000000_h – $3FFFFFFF_h$

Warum hexadezimal? (2/3)

- Kodierung von Bytes (z. B.: ASCII-Zeichen), also 8 Bit, durch zwei Hex-Zahlen:

hex	dez	hex	dez	hex	dez	hex	dez	hex	dez	hex	dez	hex	dez	hex	dez		hex	dez
00	0	10	16	20	32	30	48	40	64	50	80	60	96	70	112		F0	240
01	1	11	17	21	33	31	49	41	65	51	81	61	97	71	113		F1	241
02	2	12	18	22	34	32	50	42	66	52	82	62	98	72	114		F2	242
03	3	13	19	23	35	33	51	43	67	53	83	63	99	73	115		F3	243
04	4	14	20	24	36	34	52	44	68	54	84	64	100	74	116		F4	244
05	5	15	21	25	37	35	53	45	69	55	85	65	101	75	117		F5	245
06	6	16	22	26	38	36	54	46	70	56	86	66	102	76	118	...	F6	246
07	7	17	23	27	39	37	55	47	71	57	87	67	103	77	119		F7	247
08	8	18	24	28	40	38	56	48	72	58	88	68	104	78	120		F8	248
09	9	19	25	29	41	39	57	49	73	59	89	69	105	79	121		F9	249
0A	10	1A	26	2A	42	3A	58	4A	74	5A	90	6A	106	7A	122		FA	250
0B	11	1B	27	2B	43	3B	59	4B	75	5B	91	6B	107	7B	123		FB	251
0C	12	1C	28	2C	44	3C	60	4C	76	5C	92	6C	108	7C	124		FC	252
0D	13	1D	29	2D	45	3D	61	4D	77	5D	93	6D	109	7D	125		FD	253
0E	14	1E	30	2E	46	3E	62	4E	78	5E	94	6E	110	7E	126		FE	254
0F	15	1F	31	2F	47	3F	63	4F	79	5F	95	6F	111	7F	127		FF	255

Warum hexadezimal? (3/3)

- Für viele Aufgaben ist Hex-Darstellung der Standard, z. B. Analyse von Binärdateien: