Freie Universität Berlin

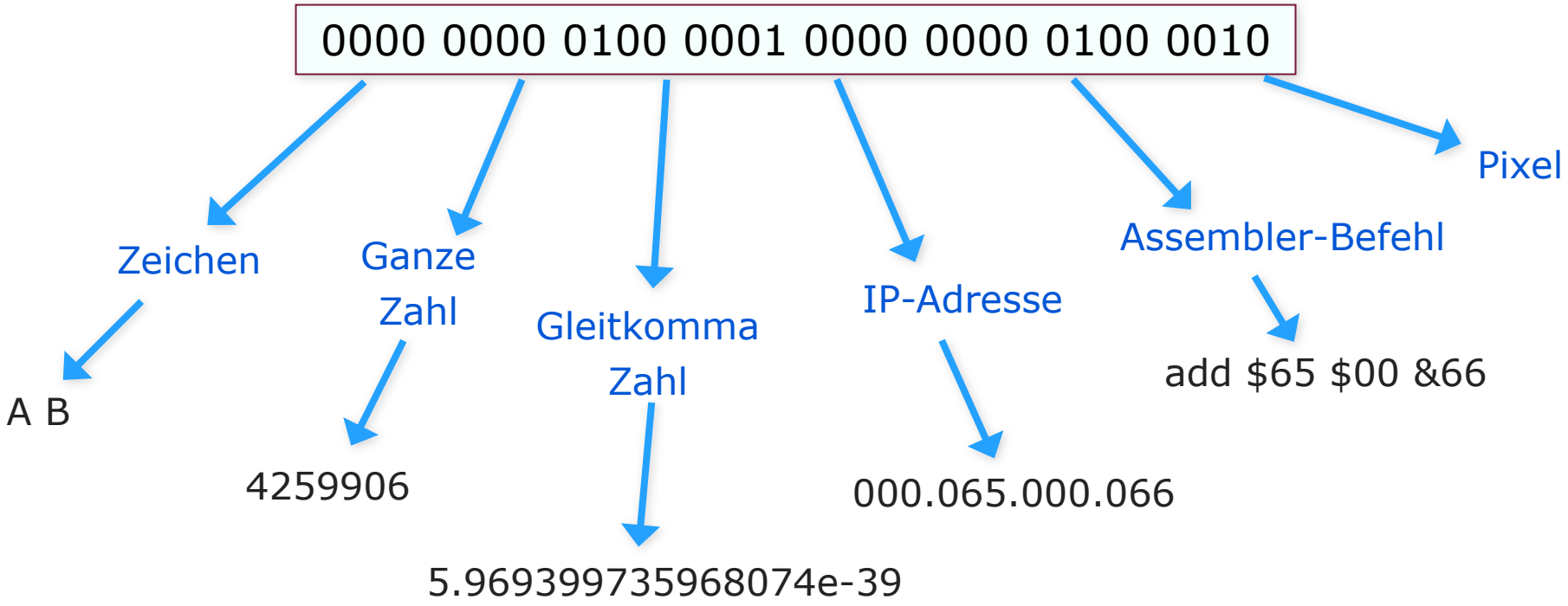
Informationskodierung

SS 2012

Prof. Dr. Margarita Esponda

Freie Universität Berlin

Eine Speicheradresse mit 32 Bits kann sehr unterschiedlich interpretiert werden.



Binärsystem → Zehnersystem

(mit Zahlen ohne Vorzeichen)

Das Binärsystem benötigt nur 2 verschiedene Symbole, um alle Zahlen zu kodieren. (**0** oder **1**)

Binärdarstellung **Dezimaldarstellung**

$$\underbrace{1\,0\,1\,0\,1\,0_{\mathbf{2}}}_{2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0} = 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0$$

$$= 32 + 8 + 2$$

$$= 42_{\mathbf{10}}$$

Zahlen in einem Stellenwertsystem

$$z_n z_{n-1} z_{n-1} \cdots z_2 z_1 z_0 = \sum_{i=0}^{n} z_i \cdot b^i$$

In dem Binärsystem sind die Ziffern $z_i \in \{0;1\}$ *und* $b = 2$

Im Dezimalsystem sind die Ziffern $z_i \in \{0;1;2;3;4;5;6;7;8;9\}$ *und* $b = 10$

Freie Universität Berlin

Zehnersystem → Binärsystem

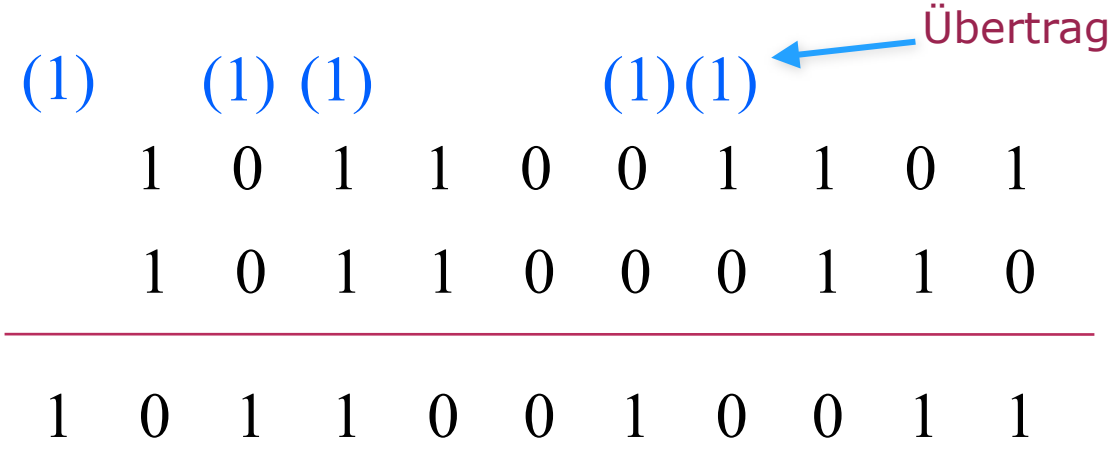
(Zahlen ohne Vorzeichen)

$42_{10} = 101010_{2}$

	Rest
42	0
21	1
10	0
5	1
2	0
1	1
0	

Freie Universität Berlin

Addition mit Binärzahlen



Freie Universität Berlin

Multiplikation mit Binärzahlen

	1	0	1	1
x		1	0	1
	1	0	1	1
0	0	0	0	
1	0	1	1	
1	1	0	1	1	1

= 32 + 16 + 4 + 2 + 1 = **55**

	11
x	**5**
	55

Freie Universität Berlin

Positive und Negative Zahlen

Mit einem **32**-Bit-Wort können $\mathbf{2^{32}}$ verschiedene Zahlen dargestellt werden.

Das erlaubt einen Wertebereich von **0** bis $\mathbf{2^{32}-1}$, wenn wir nur positive Zahlen damit darstellen wollen.

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = 1_{10}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = 2_{10}$
.....
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = 4\ 294\ 967\ 294_{10}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = \mathbf{4\ 294\ 967\ 295_{10}}$

Wie kann dieser Wertebereich aufgeteilt werden, so dass positive und negative Zahlen dargestellt werden können?

Positive und Negative Zahlen

1. Lösung **Balancierte Teilung:**

Vorzeichen **0 = positive** **1 = negative**

0000 0000 0000 0000 0000 0000 0000 0000

Probleme:

0000 0000 0000 0000 0000 0000 0000 0000 = **+0**

1000 0000 0000 0000 0000 0000 0000 0000 = **-0**

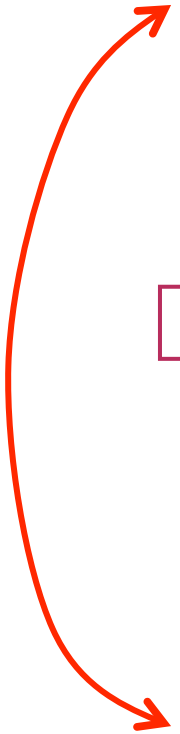
Zwei verschiedene Kodierungen für Null!

Sehr schlecht für eine Hardware-Implementierung!

Freie Universität Berlin

Positive und Negative Zahlen

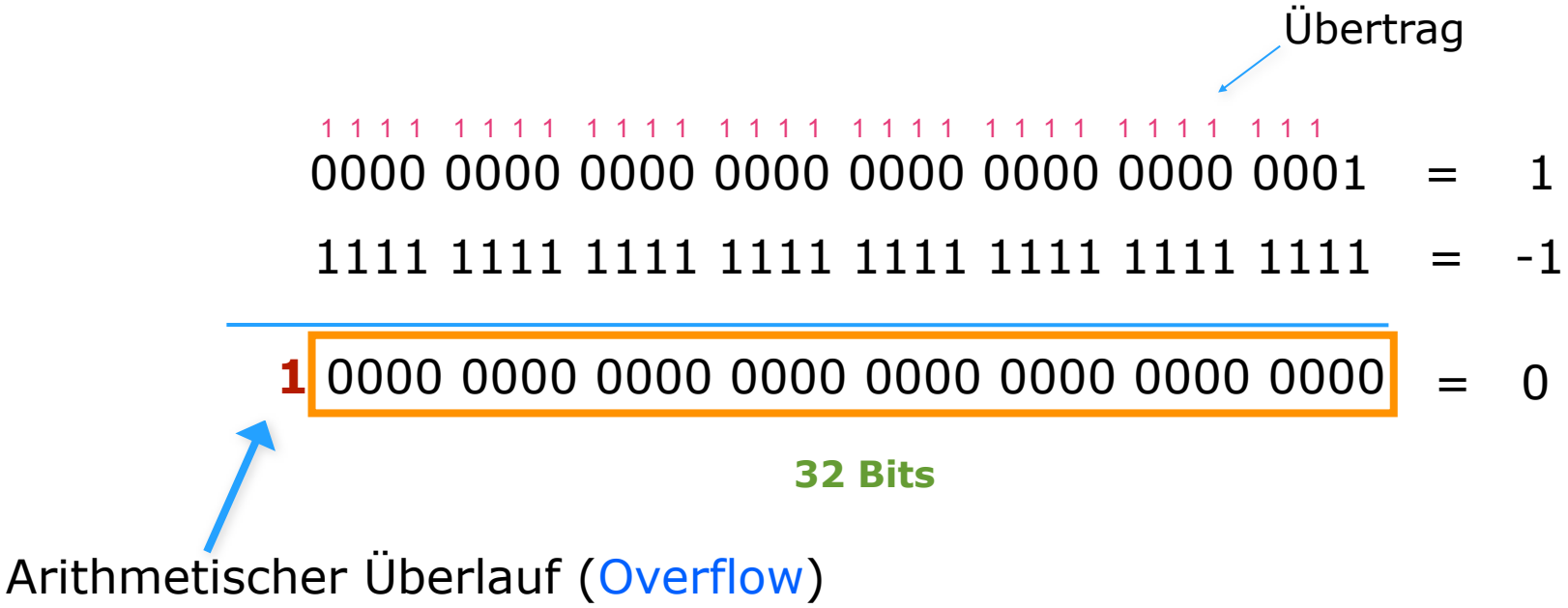
2. Lösung **Unbalancierte Teilung:**



0000 0000 0000 0000 0000 0000 0000 0000 = 0
0000 0000 0000 0000 0000 0000 0000 0001 = 1
0000 0000 0000 0000 0000 0000 0000 0010 = 2
. . . . .
0111 1111 1111 1111 1111 1111 1111 1110 = 2 147 483 646
0111 1111 1111 1111 1111 1111 1111 1111 = 2 147 483 647
1000 0000 0000 0000 0000 0000 0000 0000 = **-2 147 483 648**
1000 0000 0000 0000 0000 0000 0000 0001 = -2 147 483 647
1000 0000 0000 0000 0000 0000 0000 0010 = -2 147 483 646
. . . . .
1111 1111 1111 1111 1111 1111 1111 1101 = -3
1111 1111 1111 1111 1111 1111 1111 1110 = -2
1111 1111 1111 1111 1111 1111 1111 1111 = -1

Es gibt eine negative Zahl mehr!

Zweierkomplementdarstellung



Positive und negative Zahlen können einfach addiert werden und das Ergebnis ist richtig!

Zweierkomplementdarstellung

Wie kann ich aus n die Zahl –n kodieren?

Beispiel: $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_2 = \mathbf{7_{10}}$

Schritt 1: Alle Bits werden umgekippt

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1000_2$

Schritt 2: Eine 1 wird addiert

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1001_2 = \mathbf{-7_{10}}$

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_2 = 7_{10}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1001_2 = -7_{10}$

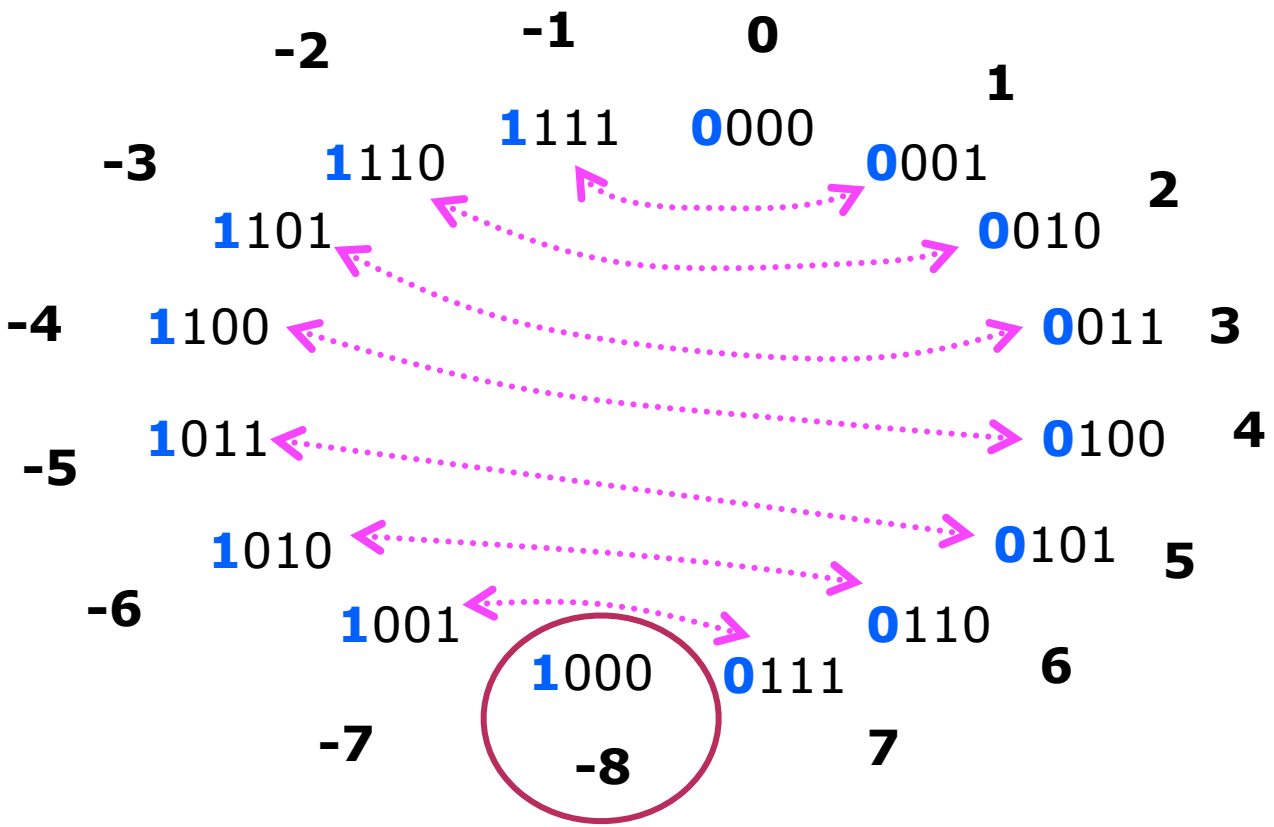
1 0000 0000 0000 0000 0000 0000 0000 0000

overflow

Freie Universität Berlin

Zweierkomplementdarstellung

mit nur 4 Bits



Freie Universität Berlin

Zweierkomplementdarstellung

-1_{10}		1_{10}		
1111	+	0001	→	0000
11111111	+	00000001	→	00000000
1111111111111111	+	0000000000000001	→	0000000000000000

```
  01111111111111111111111111111111   → 2 147 483 647
+ 00000000000000000000000000000001
--------------------------------------
  10000000000000000000000000000000   → -2 147 483 648
```

Subtraktion

8 Bits

77_{10} ⋯▸ 0 1 0 0 1 1 0 1

70_{10} ⋯▸ 0 1 0 0 0 1 1 0

Zweierkomplement

(1) (1) (1) (1)

77_{10} ⋯▸ 0 1 0 0 1 1 0 1

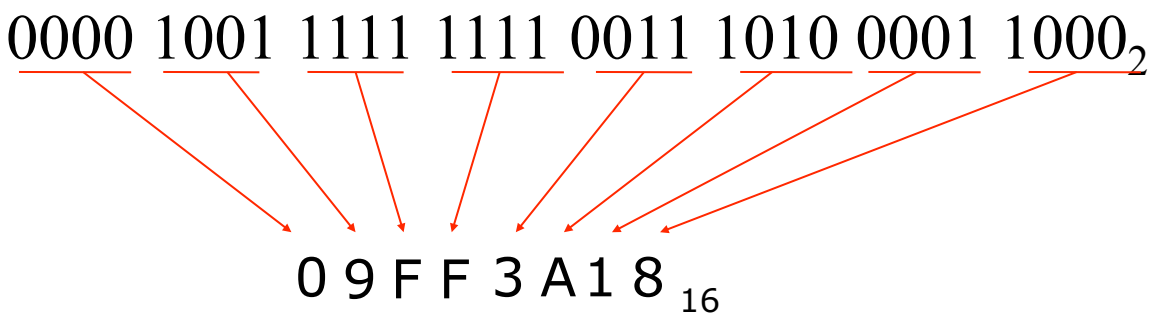
$-\ 70_{10}$ ⋯▸ 1 0 1 1 1 0 1 0

(1) 0 0 0 0 0 1 1 1 ⋯▸ 7_{10}

Freie Universität Berlin

Hexadezimal-Darstellung

0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**A**
1011	**B**
1100	**C**
1101	**D**
1110	**E**
1111	**F**



Beispiel: Farben

Hexadezimal: **FF 00 AD** HTML

Dezimal: **255, 000, 173** Java

Freie Universität Berlin

Hexadezimal-Kodierung

Basis = 16

Ziffern = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Hexadezimal → Dezimal

Beispiel:

$$1\,A\,4\,F\,B_{16} = 1\cdot16^4 + A\cdot16^3 + 4\cdot16^2 + F\cdot16^1 + B$$

16^4 16^3 16^2 16^1 16^0

$$= 107771_{10}$$

Dezimal → Hexadezimal

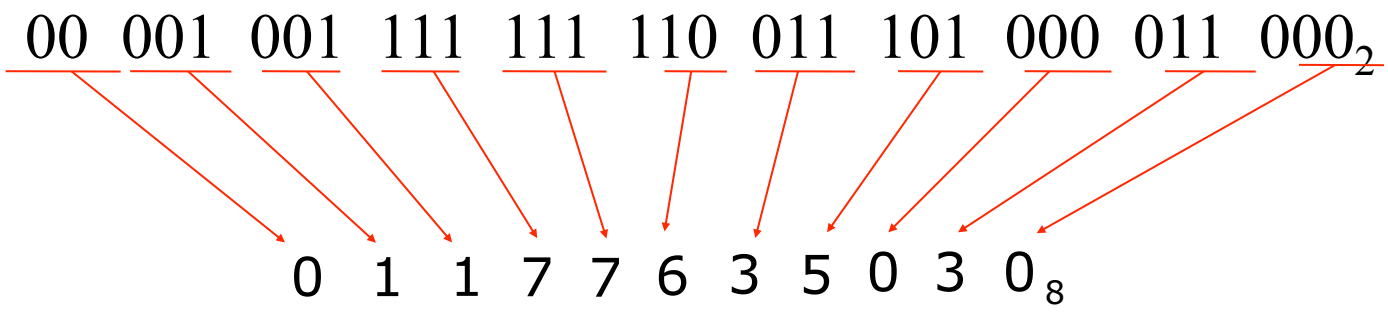
	Rest
107771	11
6735	15
420	4
26	10
1	1
0	

$1\,A\,4\,F\,B_{16}$

Freie Universität Berlin

Oktaldarstellung

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7



Oktal → **Hexadezimal**