Vorlesung Programmieren
Thema 1: Grundbegriffe

Olaf Herden
Fakultät Technik
Studiengang Informatik

$((F \vee G) \vee H) \equiv (F \vee (G \vee H))$

MB GB TB

0000010101010101010

Stand: 11/2023 (1)

Gliederung

- Aussagen(logik)
- Informationsdarstellung
- Zahlendarstellung im Rechner
- Geschichte der Rechner

Aussagen

- (Elementare) Aussage: Behauptung, der wahr (TRUE) bzw. falsch (FALSE) zugeordnet werden kann
- Beispiele:
  - „Die Erde ist eine Scheibe“ $\Rightarrow$ FALSE
  - „Die Erde ist größer als der Mond“ $\Rightarrow$ TRUE
  - „Komm her!“ $\Rightarrow$ Keine Aussage
- Elementare Aussagen werden mittels der logischen Operatoren UND (AND, $\wedge$), ODER (OR, $\vee$) und NICHT (NOT,$\neg$) zu komplexen Aussagen zusammengesetzt
- Beispiele:
  - „Die Erde ist eine Scheibe“ OR „Die Erde ist größer als der Mond“ $\Rightarrow$ TRUE
  - „Die Erde ist eine Scheibe“ AND „Die Erde ist größer als der Mond“ $\Rightarrow$ FALSE
  - NOT „Die Erde ist eine Scheibe“ $\Rightarrow$ TRUE
- Bei komplexeren Aussagen: NOT stärker als AND stärker als OR, Klammerungen möglich

Wahrheitstafeln

- Resultate der Verknüpfungen über sog. Wahrheitstafeln:

F	G	$F \wedge G$
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

F	G	$F \vee G$
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

F	$\neg F$
TRUE	FALSE
FALSE	TRUE

- Aussage enthält Variablen: Zuordnung wahr/falsch nur unter bestimmter Belegung (Zuordnung von Werten zu den Variablen) möglich
- Beispiel:
  - (x<y) AND (z>3)
  - Belegung (x=7, y=9, z=5) $\Rightarrow$ TRUE
  - Belegung (x=7, y=3, z=5) $\Rightarrow$ FALSE

Umformungsregeln (I)

- Umformung komplexer Aussagen (logische Äquivalenz, Symbol: $\equiv$)
- Regeln:
  - $(F \wedge F) \equiv F$

    $(F \vee F) \equiv F$ (Idempotenz)
  - $(F \wedge G) \equiv (G \wedge F)$

    $(F \vee G) \equiv (G \vee F)$ (Kommutativität)
  - $((F \wedge G) \wedge H) \equiv (F \wedge (G \wedge H))$

    $((F \vee G) \vee H) \equiv (F \vee (G \vee H))$ (Assoziativität)
  - $(F \wedge (G \vee H)) \equiv ((F \wedge G) \vee (F \wedge H))$

    $(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H))$ (Distributivität)
  - $(F \wedge (F \vee G)) \equiv F$

    $(F \vee (F \wedge G)) \equiv F$ (Absorption)

Umformungsregeln (II)

- $\neg \neg F \equiv F$ (Doppelnegation)
- $\neg (F \wedge G) \equiv (\neg F \vee \neg G)$

  $\neg (F \vee G) \equiv (\neg F \wedge \neg G)$ (Regeln von de Morgan)
- $TRUE \wedge F \equiv F$
- $FALSE \vee F \equiv F$ (Neutrale Elemente)

- Beweis der Regeln über Wahrheitstafeln

Beispiel: Beweis Distributivgesetz

- Behauptung: $(F \wedge (G \vee H)) \equiv ((F \wedge G) \vee (F \wedge H))$
- Beweis:

F	G	H	$G \vee H$	$(F \wedge (G \vee H))$	$F \wedge G$	$F \wedge H$	$((F \wedge G) \vee (F \wedge H))$
T	T	T	T	T	T	T	T
T	T	F	T	T	T	F	T
T	F	T	T	T	F	T	T
T	F	F	F	F	F	F	F
F	T	T	T	F	F	F	F
F	T	F	T	F	F	F	F
F	F	T	T	F	F	F	F
F	F	F	F	F	F	F	F

- Anm.: Erweiterung Aussagenlogik zur Prädikatenlogik (siehe Informatik-VL)

Implikation (I)

- Wenn man aus einer wahren Aussage A schließen kann, dass dann auch die Aussage B wahr ist, spricht man von einer Implikation
- Notation: $A \Rightarrow B$ oder $A \rightarrow B$
- Sprechweisen:
  - Aus A folgt B
  - Unter der Voraussetzung A gilt B
  - A impliziert B
  - Wenn A gilt dann gilt auch B
  - B ist notwendig für A
  - A ist hinreichend für B
- Beispiele:
  - Wenn es regnet, ist die Straße nass.
  - n ist teilbar durch 6 $\Rightarrow$ n ist teilbar durch 3

Implikation (II)

- Umgangssprachlich lässt man sich gelegentlich zu weiteren - falschen - Aussagen verleiten
- Beispiele:
  - Es regnet nicht $\Rightarrow$ Die Straße ist nicht nass
  - Folgerung unzulässig, Straße kann aus anderen Gründen nass sein (Rohrbruch, Feuerwehrübung, ...)
  - n ist nicht durch 6 teilbar $\Rightarrow$ n ist nicht durch 3 teilbar
  - Auch diese Folgerung falsch, z.B. Zahl 15 ist nicht durch 6 teilbar, wohl aber durch 3
- Das bedeutet:
  - Wenn Folgerung A $\Rightarrow$ B wahr ist, dann erhält man aus der Aussage ¬A keine Aussage über B, B kann wahr oder falsch sein
- Wahrheitstafel:

F	G	F $\Rightarrow$ G
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	TRUE
FALSE	FALSE	TRUE

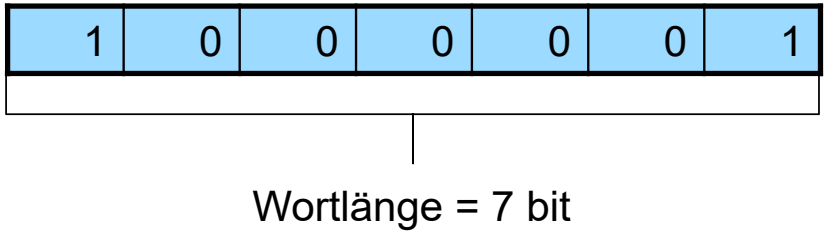
Gliederung

- Aussagen(logik)
- Informationsdarstellung
- Zahlendarstellung im Rechner
- Geschichte der Rechner

Binärzeichen/Maschinenwort/Wortlänge

- Darstellung Daten im Rechner mit physikalischen Größen (Signalen) dargestellt
- Können zwei unterschiedliche Werte annehmen:
  - Hohe/niedrige Spannung auf einer Leitung
  - Elektrische/keine elektrische Ladung in einer Speicherzelle
  - Abstraktion von physikalischer Darstellung; Zuordnung der Werte 0 bzw. 1 (Binärzeichen)

- Zusammenfassung Binärzeichen zu Folgen fester Länge (Maschinenwort)
- Wortlänge := Anzahl der Binärzeichen eines Maschinenworts (Einheit Bit)
- Bsp.:

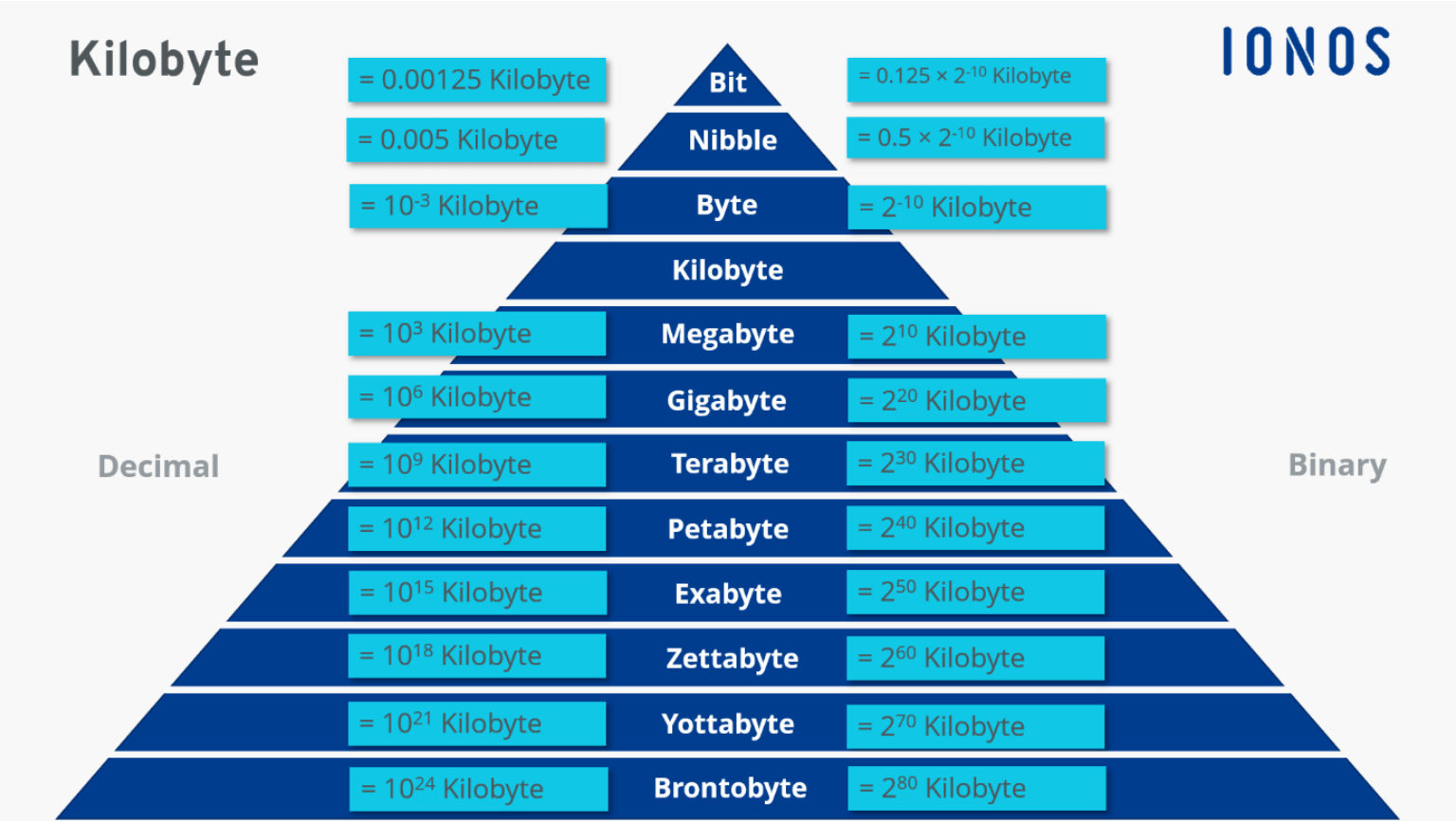


Byte und Wort

- Statt Bit ist Byte eine üblichere Einheit
- 1 Byte := 8 Bit
- Als weitere Einheiten werden Wort, Doppelwort, Vierfachwort, ... etc.
- Dabei: Wort = 16, 32 oder 64 Bit (Herstellerabhängig)

Präfixe

- Einheiten können zur Handhabung großer Werte Präfixe erhalten

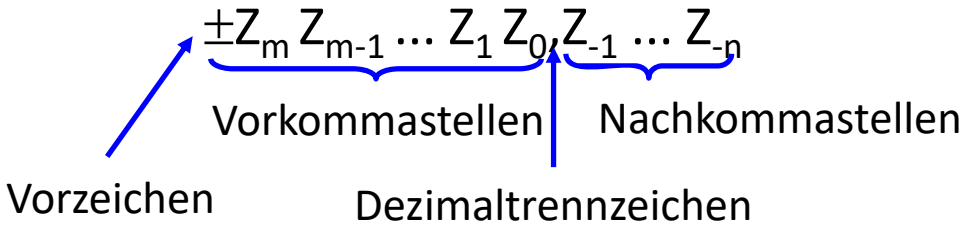


[https://www.ionos.co.uk/digitalguide/websites/web-development/what-is-a-kilobyte/]

Zahlensysteme (I)

- Zahlen werden in einem bestimmten Stellenwertsystem dargestellt
- Jedes Stellenwertsystem wird durch seine Basis definiert
- Im Tagesgebrauch: Dezimalsystem (Basis = 10)

- Allgemein lässt sich jede beliebige Zahl durch eine Ziffernfolge beschreiben:



- Der Beitrag einer Ziffer zum Wert bestimmt ihre Position (sei B die Basis des Zahlensystems):

$$X = \pm\left(Z_m B^m + Z_{m-1} B^{m-1} + \ldots + Z_1 B^1 + Z_0 B^0 + Z_{-1} B^{-1} + \ldots + Z_{-n} B^{-n}\right)$$

$$= \pm \sum_{i=-n}^{m} Z_i B^i$$

Zahlensysteme (II)

- In der Informatik wichtige Systeme:
  - Dualsystem (Basis = 2, Ziffern = {0,1})
  - Oktalsystem (Basis = 8, Ziffern = {0,1, ..., 7})
  - Hexadezimalsystem (Basis = 16, Ziffern = {0,1, .., 9, A, ..., F})

    Dabei: A dezimaler Wert 10, B 11 usw.
- Notation: Nachstellen der Basis oder eines kennzeichnenden Buchstabens:
  - Dualsystem: $101010{,}0101_2$ oder 101010,0101B
  - Oktalsystem: $7601{,}55_8$ oder 7601,55Q
  - Dezimalsystem: $9531{,}55_{10}$ oder 9531,55D
  - Hexadezimalsystem: $1AC{,}B8_{16}$ oder 1AC,B8H

Dualsystem (I)

- Grundrechenarten analog zum Dezimalsystem: Operation stellenweise durchführen, Weitergabe von möglichen Überträgen

Addition	Subtraktion	Multiplikation	Division
0 + 0 = 0	0 - 0 = 0	0 · 0 = 0	0 : 0 verboten
0 + 1 = 1	0 - 1 = 1 Ü=-1	0 · 1 = 0	0 : 1 = 0
1 + 0 = 1	1 - 0 = 1	1 · 0 = 0	1 : 0 verboten
1 + 1 = 0 Ü=1	1 - 1 = 0	1 · 1 = 1	1 : 1 = 1

- Beispiele:

```
     101101
+    011001
Ü   111   1
    1000110
```

```
101 · 1011
      101
     101
    000
   101
Ü   1
   110111
```

Dualsystem (II)

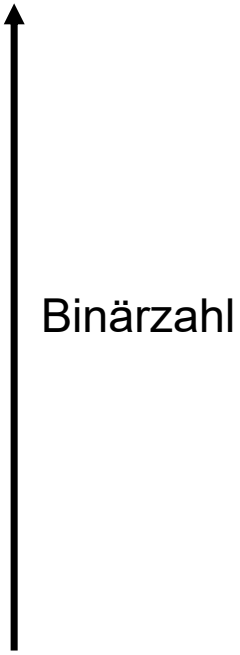
- Auf Dualzahlen Operationen der Komplementbildung (vertauschen von 0 und 1) und des Schiebens (Shift-Operation) von Bedeutung
- Mit ihnen lassen sich die Grundrechenarten darstellen
- So bedeutet die Verschiebung einer Dualzahl um s Stellen nach links/rechts die Multiplikation/Division dieser Zahl mit 2^s
- Siehe auch Informatik/Digitaltechnik-Vorlesung

- Grundrechenarten im Oktal- und Hexadezimalsystem prinzipiell gleich
- Anzahl der Regeln aufgrund größeren Ziffernvorrats größer

Konvertierung zwischen Systemen (I)

- Allgemein: Verwenden der Formel von Folie „Zahlensysteme (I)“
- Konvertierung von Dezimalsystem nach Binärsystem unter Verwendung Divisionsmethode
- Beispiel: Vom Dezimal- ins Binärsystem

```
6235 : 2 = 3117 Rest 1
3117 : 2 = 1558 Rest 1
1558 : 2 =  779 Rest 0
 779 : 2 =  389 Rest 1
 389 : 2 =  194 Rest 1
 194 : 2 =   97 Rest 0
  97 : 2 =   48 Rest 1
  48 : 2 =   24 Rest 0
  24 : 2 =   12 Rest 0
  12 : 2 =    6 Rest 0
   6 : 2 =    3 Rest 0
   3 : 2 =    1 Rest 1
   1 : 2 =    0 Rest 1
```



Konvertierung zwischen Systemen (II)

- Verfahren auch bei Kommazahlen
- Dabei: Nachkommastellen mit neuer Basis multiplizieren
- Beispiel: $11{,}625_{10}$ vom Dezimal- ins Binärsystem wandeln

```
11 : 2 = 5 Rest 1
 5 : 2 = 2 Rest 1
 2 : 2 = 1 Rest 0
 1 : 2 = 0 Rest 1
```

Vorkommastellen der Binärzahl

```
0,625 * 2 = 1,25
0,25  * 2 = 0,5
0,5   * 2 = 1
```

Nachkommastellen der Binärzahl

- Damit gilt: $11{,}625_{10} = 1011{,}101_2$
- Anmerkung: Analoges Vorgehen für Umrechnung Dezimal- nach Oktal-/Hexadezimalsystem

Konvertierung zwischen Systemen (III)

- Umwandlung andere Basis ins Dezimalsystem
- Vorgehen: Jede Stelle mit Potenz der Basis der Stelle multiplizieren und diese Werte addieren
- Beispiel: $66,A1_{16}$ ins Dezimalsystem verwandeln

  $66,A1_{16}$

  $= 6 \cdot 16^1 + 6 \cdot 16^0 + 10 \cdot 16^{-1} + 1 \cdot 16^{-2}$

  $= 102,62890625_{10}$
- Beispiel: $1100101,1101_2$ ins Dezimalsystem verwandeln

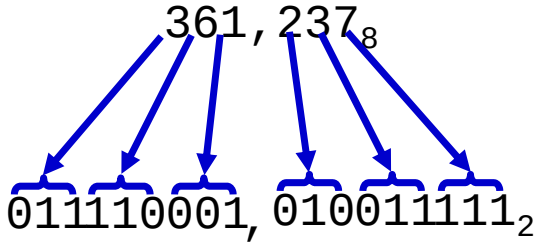
  $1100101,1101_2$

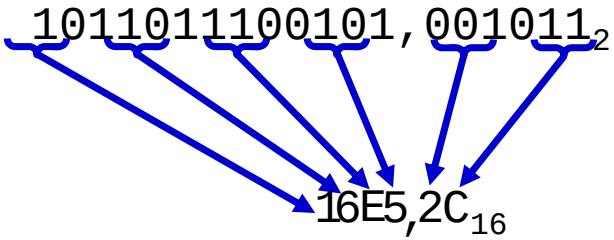
  $= 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^2 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-4}$

  $= 101,8125_{10}$

Konvertierung zwischen Systemen (IV)

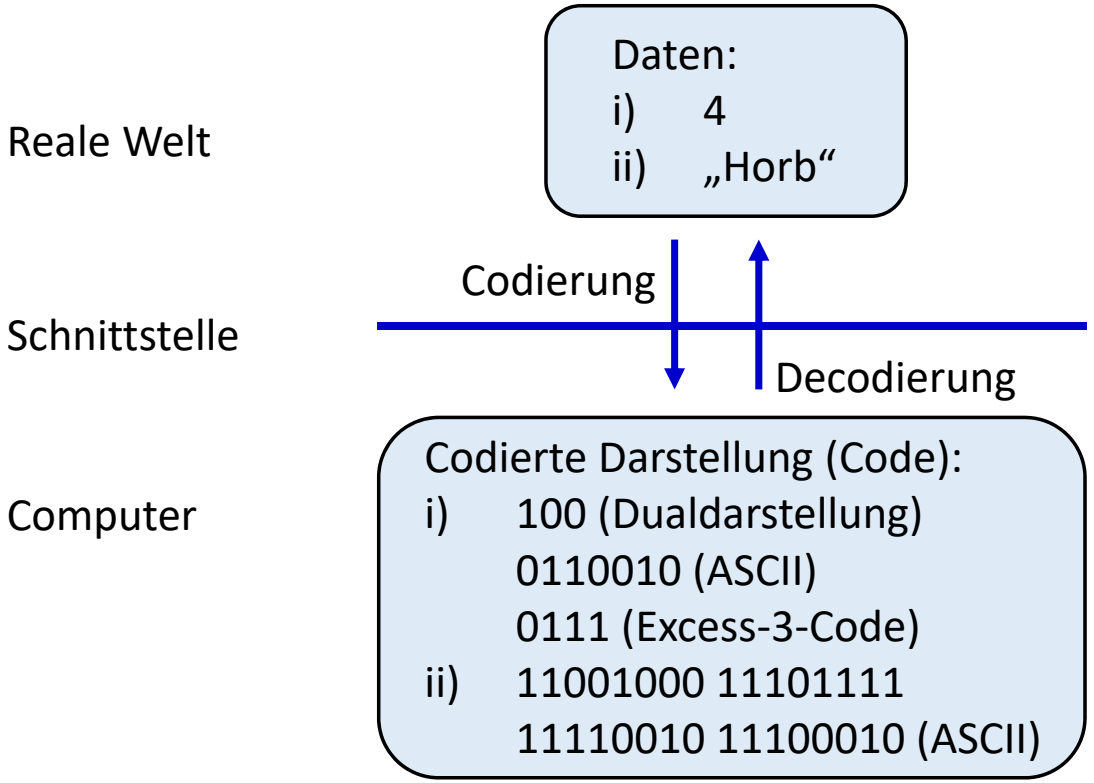
- Umwandlung zwischen Dual/Oktal/Hexadezimalsystem
- Basen aller drei Systeme sind Potenzen von 2
- Oktalziffer ist Gruppe von drei Dualziffern
- Hexadezimalziffer ist Gruppe von vier Dualziffern
- Beispiele:





Codierung (I)

- Problemstellung: Werte der realen Welt müssen in Rechner-lesbarer Form (Folge von 0 und 1) repräsentiert werden

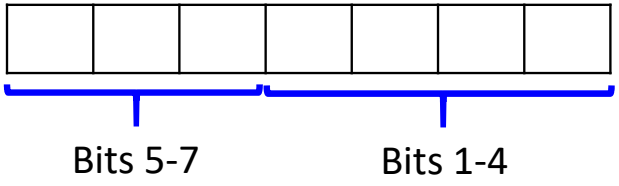


Codierung (II)

- Code: Zuordnungsvorschrift, die jedem Element einer Menge von Informationen einen Wert (Codewort) zuweist
- Binärcode: Alle Codeworte binär
- Codierung: Abbildung Wert auf Codewort
- Decodierung: Abbildung Codewort auf Wert
- Prüfbits/Parity Bits: Zusätzliche Bits zur Codesicherung

ASCII-Code (I)

- ASCII: American Standard Code for Information Interchange
- Erste Standardisierung 1968: ANSI X3.4 bzw. ISO 8859/1.2
- Enthält alle Buchstaben, Ziffern und Steuerzeichen
- Ursprünglich 7-Bit-Code mit 128 Zeichen
- Code:



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

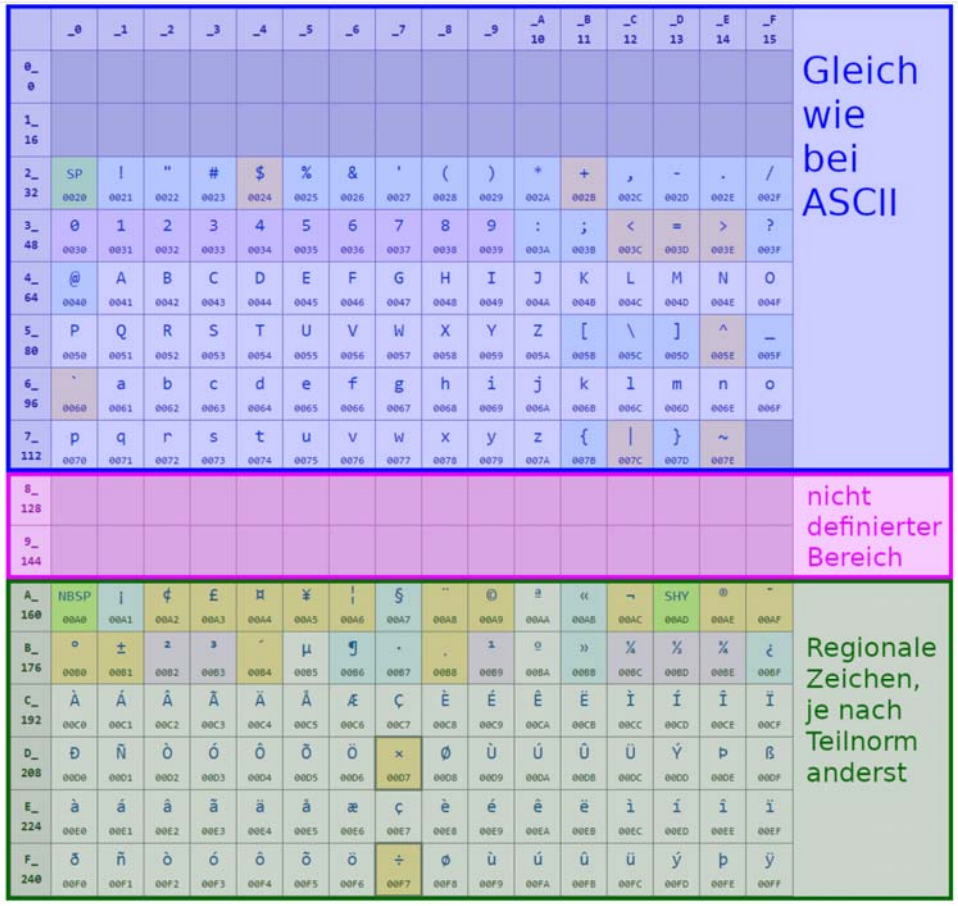
[https://www.cfd.tu-berlin.de/Lehre/EDV1/skripte/alles/node77.html]

ASCII-Code (II)

- (De)codierung erfolgt durch „Ablesen“ in Tabelle
- Beispiele:
  - 100 0111 -> G
  - ? -> 011 1111
- Nachteil: Wesentliche Zeichen europäischer Sprachen fehlen
- Daher:
  - Nutzung achtes Bit
  - Neben Sonderzeichen auch mathematische Zeichen und grafische Zeichen
- Trotz allem: Vielzahl an (verschiedenen) Sonderzeichen nicht handhabbar
- Ausweg ISO 8859:
  - 8 Bit ASCII Zeichensatz
  - Erste 128 Zeichen gleich (entsprechen 7 Bit ASCII)
  - Zweite 128 Zeichen variabel zur Anpassung an Sprachraum

ASCII-Code (III)

- Aufbau normierte ISO 8859 Zeichensätze:



ISO 8859	
-1	*Latin-1*, Westeuropäisch
-2	*Latin-2*, Mitteleuropäisch
-3	*Latin-3*, Südeuropäisch
-4	*Latin-4*, Nordeuropäisch
-5	Kyrillisch
-6	Arabisch
-7	Griechisch
-8	Hebräisch
-9	*Latin-5*, Türkisch
-10	*Latin-6*, Nordisch
-11	Thai
~~-12~~	(existiert nicht)
-13	*Latin-7*, Baltisch
-14	*Latin-8*, Keltisch
-15	*Latin-9*, Westeuropäisch
-16	*Latin-10*, Südosteuropäisch

Für Deutschland:
ISO-8859-1 (Latin-1)

[https://www.marcelwicki-softwaredesign.ch/2018/08/08/zeichencodierungen/]

Unicode (I)

- Idee: Alle (Sonder-)Zeichen aller Sprachen in einem Code
- Erweiterung von 8 auf 32 Bit
- D.h. $2^{32} \approx$ 4,29 Milliarden Zeichen
- Bezeichnung eines 16 Bit codierten Unicodezeichens:
  - U+XXXX
  - Jedes X hexadezimale Ziffer
- Kompatibilität zu ASCII-Code:
  - ASCII: U+0000 bis U+007F
  - ISO 8859-1: U+0080 bis U+00FF
- Unicode weist Zeichen Code zu
- Keine Aussage über Länge der Codierung
- UTF: Unicode Transformation Format

Unicode (II)

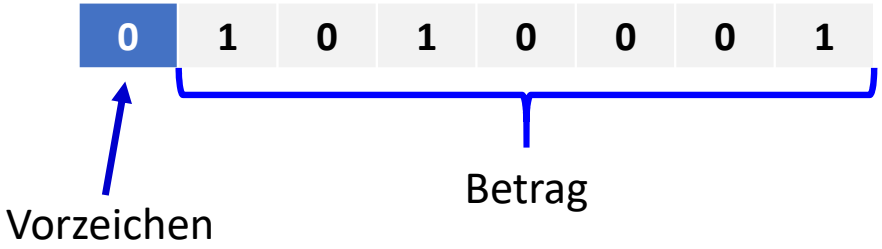
- UTF-32:
  - Immer 32 Bit
  - Sehr speicherplatzintensiv
- UTF-16 (Windows, OS-X):
  - Für „gebräuchliche“ Zeichen 2 Byte pro Zeichen
  - 4 Byte für „exoterische“ Zeichen
- UTF-8 (www, E-Mail, Unix):
  - Zwischen einem und vier Byte pro Zeichen
  - Vorteil:
    - 1 Byte-Codierung entspricht 7 Bit ASCII
    - Sehr effizient bei lateinischen Zeichen (gegenüber 2 Byte in UTF-16)
    - Bei anderen Schriften typischerweise 3 Byte (gegenüber 2 Byte in UTF-16)

Gliederung

- Aussagen(logik)
- Informationsdarstellung
- Zahlendarstellung im Rechner
- Geschichte der Rechner

Darstellung ganzer Zahlen

- Anforderungen an Darstellung:
  - Bei einer Wortbreite von n Bit sollen 2^n Zahlen dargestellt werden können
  - Es sollen gleich viel positive wie negative Zahlen dargestellt werden
  - Arithmetischen Operationen sollen möglichst einfach/effizient ausführbar sein
- Verschiedene Möglichkeiten:
  - Vorzeichen-Betrag-Darstellung
  - Excess-3-Darstellung
  - 2-Komplement-Darstellung
  - 1-Komplement-Darstellung
- Vorzeichen-Betrag-Darstellung:



Excess-3-Code

- Ordne Dezimalziffern 0 bis 9 jeweils 4-Bit-Code (Tetrade) zu:

Dezimalziffer	Code
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

- Beispiele:
  - 234 hat den Excess-3-Code 0101 0110 0111
  - Der Exzess-3-Code 1001 0110 stellt die Zahl 63 dar
- Name „Excess-3“: Jeder Code um 3 höhere Zahl als entsprechende Binärzahl
- Anmerkungen:
  - Allgemein gibt es Excess-q-Codes, wobei q jeweils den Versatz zur jeweiligen Binärzahl angibt
  - Addition und Subtraktion recht einfach möglich, Multiplikation und Division dagegen sehr aufwändig $\Rightarrow$ Nur in Spezialfällen einsetzbar

1- bzw. 2-Komplement-Darstellung (I)

- 2-Komplement einer Binärzahl a ist Binärzahl b, für die $a + b = 2^n$ gilt
- 1-Komplement einer Binärzahl a ist Binärzahl b, für die $a + b = 2^n - 1$ gilt
- Berechnung:
  - 1-Komplement: Jedes Bit „drehen“
  - 2-Komplement: 1-Komplement bilden, Addition 1
- Beispiel (Sei n=4):
  - 1-Komplement von 0110 ist 1001
  - 2-Komplement von 0110 ist 1010

1- bzw. 2-Komplement-Darstellung (II)

- Beispiel (n=4):

2-Komplement	Dezimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

1- bzw. 2-Komplement-Darstellung (III)

- Anwendung:
  - 2-Komplement ist die Darstellung der negativen Zahl
  - Leichte Realisierung arithmetische Operationen mit 2-Komplement:
    - Addition erfolgt „ganz normal“
    - Subtraktion durch Addition der Zahl im 2-Komplement
    - Vernachlässigen der (n+1)-Stelle

1- bzw. 2-Komplement-Darstellung (IV)

- Addition von 5 und 3 bzw. Subtraktion von 3 von 5
- Erwartete Ergebnisse: 5 + 3 = 8 bzw. 5 - 3 = 2
- Dualdarstellung von 5: `00000101`
- Dualdarstellung von 3: `00000011`
- 2-Komplement von 3: `11111100` + 1 = `11111101`
- Addition:

```
  00000101
+ 00000011
----------
  00001000
```

- Subtraktion durch Addition des 2-Komplements:

```
  00000101
+ 11111101
----------
 100000010
```

1- bzw. 2-Komplement-Darstellung (V)

- Funktioniert auch für negative Zahlen
- Bsp.: 5 + (-3) = 2
- Dualdarstellung von 5: `00000101`
- Dualdarstellung von -3:
  - Darstellung von 3: `00000011`
  - 1-Komplement: `11111100`
  - 2-Komplement: `11111100` + `1` = `11111101`
- Addition:

```
  00000101
+ 11111101
-----------
 100000010
```

- Subtraktion durch Addition der neg. Zahl, d.h. $5 - (-3) = 5 + (+3) = 8$:

```
  00000101
+ 00000011
-----------
  00001000
```

Überlauf

- Überlauf: Überschreiten Wertebereich bei Rechenoperationen (z.B. bei Addition zweier großer Zahlen)
- Beispiel:
  - Sei $n = 8$, Darstellung nur positiver Zahlen
  - D.h. Wertebereich von 0 bis $2^8 - 1 = 255$
  - Es sollen 161 und 127 addiert werden
  - Dualdarstellung von 161: `10100001`
  - Dualdarstellung von 127: `01111111`
  - Addition:

```
  10100001
+ 01111111
-----------
 100100000
```

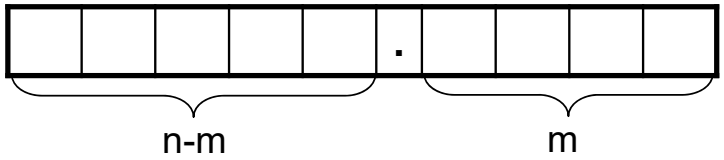
  - Richtiges Resultat 288, Abschneiden vorderste Position (wegen $n = 8$), falsches Resultat `00100000` $= 32$

Darstellung reeller Zahlen

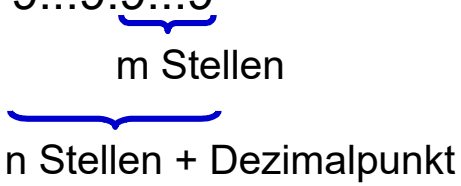
- Darstellung reeller und rationaler Zahlen als Binärzahl
- Probleme:
  - Nicht alle Zahlen darstellbar (Endlichkeit des Rechners)
  - Bei irrationalen Zahlen (z.B. π, e, Wurzel aus 2) nur näherungsweise Darstellung
- Zwei Möglichkeiten:
  - Festkommadarstellung
  - Gleitkommadarstellung

Festkommadarstellung

- Darstellung:



- Ziele:
  - Exakte Darstellung von m (≤ n) "Nachkommastellen" der Dualdarstellung einer Zahl
  - Sowohl für positive als auch für negative reelle Zahlen
- Vorgehen:
  - Umwandlung der zu codierenden Zahl durch geeignetes Multiplizieren und Auf- bzw. Abrunden in eine ganze Zahl
  - Dann: Verwendung der Darstellungsmöglichkeiten für ganze Zahlen
- Darstellung wird relativ selten verwendet
- Datentyp in Programmiersprachen:
  - `NUMBER(N,M)`



Gleitkommadarstellung (I)

- Gleitkommazahl lässt sich in der Form $m \cdot B^e$ schreiben
- m Mantisse, e Exponent, B Basis
- Für dieselbe Zahl viele Möglichkeiten, Beispiel:

$$34{,}78 = (+34{,}78)\cdot 10^0 = (+347{,}8)\cdot 10^{-1} = (+0{,}0003478)\cdot 10^5$$
$$= (+3{,}478)\cdot 10^1 = (+0{,}3478)\cdot 10^2$$

- Normalisierte Gleitkommazahl: erste von 0 verschiedene Mantissenziffer unmittelbar vor oder hinter dem Komma (je nach Definition)
- Letzten beide Darstellungen im Beispiel normalisiert
- Je nach Rechner/Prozessor gilt damit: $10 > m \geq 1$ oder $1 > m \geq 0{,}1$
- 0 lässt sich nicht normalisieren $\Rightarrow$ Sonderbehandlung notwendig

Gleitkommadarstellung (II)

- Struktur Gleitkommawort:

S	C	M

- S Vorzeichenbit (0=positiv, 1=negativ)
- Für e relativ kleiner Wertebereich (z.B. -50 bis $+49$) ausreichend, um sowohl extrem kleine als auch extrem große Zahlen darzustellen
- C Charakteristik, enthält Exponenten, Vermeidung negativer Exponenten durch Addition Verschiebekonstante K_v, K_v Betrag des kleinsten e-Wertes
- Beispiel: e: -50 ... 0 ... +49

Addition der Verschiebekonstante K_v (hier 50)

C: 0 ... 50 ... 99

- M Mantissenbetrag

IEEE 754-Norm (I)

- Gleitpunktformat nach IEEE 754-Norm:

	Einfache Genauigkeit	Doppelte Genauigkeit	Erweiterte Genauigkeit
Charakteristik	8 bit	11 bit	15 bit
Mantisse	23 bit	52 bit	112 bit
K_V	127	1023	16383

IEEE 754-Norm (II)

- Beispiel – $5{,}625$ in $IEEE\ 754$-Norm mit einfacher Genauigkeit darstellen
- Vorgehen:
  - Vorzeichenbestimmung: Zahl negativ $\Rightarrow$ Vorzeichen (linkeste Stelle Binärzahl) = 1
  - Vorkommastellen in Binärzahl umwandeln: $5_{10} = 101_2$
  - Nachkommastellen in Binärzahl umwandeln:

    $0{,}625 * 2 = 1{,}250$

    $0{,}250 * 2 = 0{,}5$

    $0{,}5 * 2 = 1$

    $\Rightarrow$ Nachkommastellen sind , 101_2
  - Insgesamt: $101{,}101_2$
  - Normalisieren, d.h. Verschieben nach rechts, bis linkste 1 unmittelbar vor Komma, Anzahl der verschobenen Positionen merken: $101{,}101_2 = 1{,}01101_2$ mit Verschiebefaktor 2

IEEE 754-Norm (III)

- Vorgehen (Fortsetzung):
  - Wegen Normalisierung Stelle vor Komma immer 1 $\Rightarrow$ Weglassen (implizites Bit in der Mantisse)
  - Vorteil: Darstellung um eine Zweierpotenz größere Zahl möglich
  - Also: 01101_2 ist Mantisse
  - Berechnung Charakteristik:

    Verschiebekonstante K_V + Verschiebefaktor

    = $127_{10} + 2_{10} = 129_{10} = 10000001_2$
  - Insgesamt:

    Vorzeichenbit s: 1

    Charakteristik C: 10000001_2

    Mantisse M: 01101_2

    Ergibt $IEEE\ 754$ Single Precision Darstellung: $11000000101101 0 \ldots 0$

IEEE 754-Norm (IV)

- Spezialfälle:

Darstellung von	Charakteristik	Mantisse
Null	0	0
Unendlich	255	0
NaN	255	Nicht alle Null

  - NaN: „Not a number“
  - Aufruf von Operationen mit unzulässigen Argumenten (z.B. Wurzel mit negativem Radikand) $\Rightarrow$ NaN als Hinweis auf undefiniertes Ergebnis
  - Anm.: 255 als Charakteristik bezieht sich auf Single Precision

Gleitkommaarithmetik

- Berücksichtigung Ausnahmefälle:
  - Überlauf des Exponenten ("overflow"), (z.B. bei Multiplikation sehr großer Zahlen: 0.9E64*0.9E64 nicht einfachlang darstellbar)
  - Unterlauf des Exponenten ("underflow"), (z.B. bei Multiplikation sehr kleiner Zahlen)
  - Erhebliche Rundungsfehler möglich (z.B. durch Exponentenangleich bei Addition sehr unterschiedlich großer Zahlen oder durch Stellenauslöschung bei Subtraktion gleich großer Zahlen):
    - Gegeben: $1{,}75 \cdot 10^{10}$, addiere beleibig oft $2{,}34 \cdot 10^{-10}$
    - Ergebnis wird immer $1{,}75 \cdot 10^{10}$ sein

    $\Rightarrow$ Ergebnisse von Gleitkommaberechnungen können u.U. erheblich vom exakten Wert abweichen!!
- Übliche Rechengesetze (Assoziativ-/Distributiv-/Kommutativgesetz) gelten i.A. nicht, siehe auch Beispiel oben

Übersicht

- Aussagen(logik)
- Informationsdarstellung
- Zahlendarstellung im Rechner
- Geschichte der Rechner

Die frühen Jahre

- Altertum bis Mittelalter: Abakus
- 17. Jhd.: Mechanische Rechenmaschinen (Schickardt, Pascal, Leibniz)
- 1838: Charles Babbage
  - Programmgesteuerte Maschine mit Zahlenspeicher, Rechenwerk, Steuereinheit und Programmspeicher
- 1886: Hollerith
  - Lochkartenverarbeitungsmaschine mit verschiedenen Programmen
  - Einsatz bei US-Volkszählung 1890

Zuse und Aiken (I)

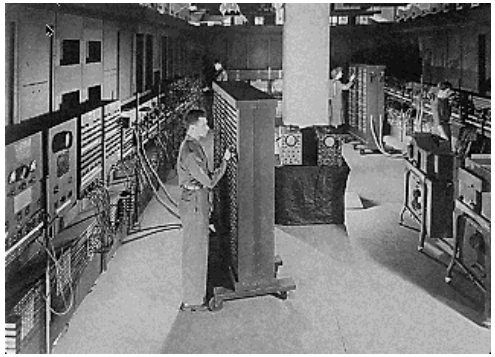
- Erste grundlegende Arbeiten (etwa zeitgleich, aber unabhängig):
  - Konrad Zuse:
    - 1936: Patentanmeldung
    - 1941: Bau des Z3
      - Rechengeschwindigkeit von 20 Additionen/Sekunde
      - Speicherkapazität von 1408 Bit
      - 2600 Relais
      - Relais-Arbeitsspeicher
      - Lochstreifen als Programmspeicher

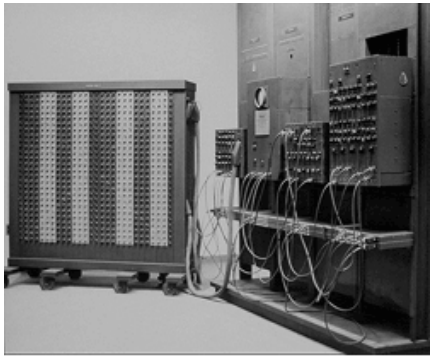
Zuse und Aiken (II)

- Howard Aiken:
  - 1944: Bau Mark I
    - Relaisrechner mit 72 Addierwerken
    - Über 700000 Einzelteile
    - Gewicht 15t
- Charakteristika dieser Rechner:
  - Programmgesteuert durch Lochstreifen, d.h. nicht durch festes mechanisch oder elektrisch fixiertes Programm festgelegt
  - Durch Einsatz mehrerer Lochstreifenleser waren Unterprogramme möglich
  - Aber: Programme und Daten streng getrennt

ENIAC

- 1946: ENIAC (Electronic Numerical Integrator and Computer):
  - 19.000 Röhren und 1.500 Relais
  - Programmierung über Schalttafeln und Stecker
  - Rechengeschwindigkeit 1000 Additionen/Sekunde
  - Speicherkapazität von 20 Dezimalzahlen
  - 150 qm Fläche, 30t Gewicht, Leistungsaufnahme von 200 kW
  - Kosten von ca. 2 Millionen Dollar (im Jahre 1945)



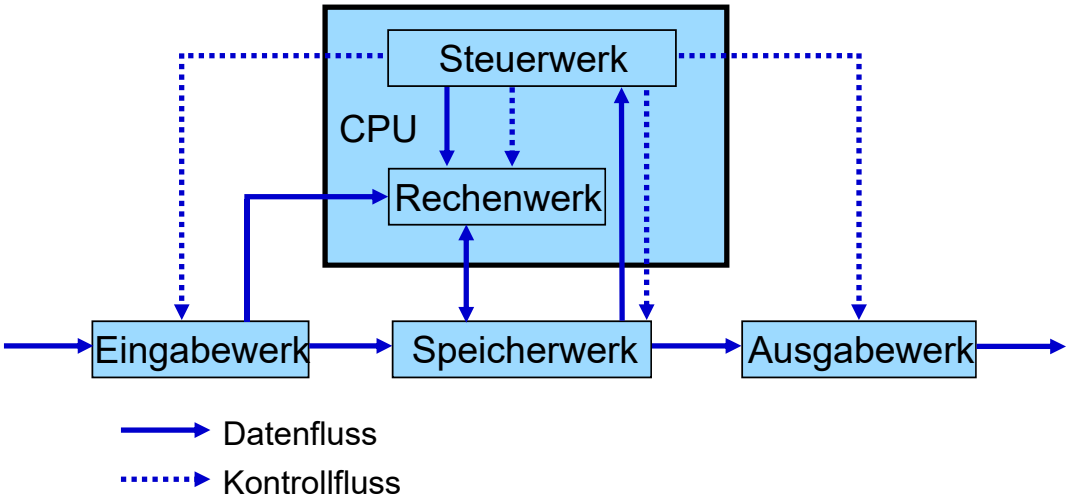


The ENIAC Today

[http://ei.cs.vt.edu/~history/ENIAC.Richey.HTML]

von-Neumann-Rechner (I): Aufbau

- Klassisches Konzept eines Universalrechners (von Neumann 1946)



  - Speicherwerk: Speicherung von Programmen und Daten
  - Rechenwerk: Ausführung arithmetischer (+, -, *, /) und logischer ($\wedge$,$\vee$, $\neg$) Operationen
  - Steuerwerk (auch Leitwerk): Steuerung des Programmablaufs
  - Rechen- und Steuerwerk bilden Zentraleinheit (CPU = Central Processing Unit)
  - Eingabe- und Ausgabewerk: Eingabe von Programmen/Daten in den Speicher; Ausgabe von Ergebnissen vom Speicher "nach außen"

von-Neumann-Rechner (II): Prinzipien (I)

- Struktur unabhängig von Problemstellung:
  - Für jedes Problem Bearbeitungsvorschrift (Programm) "von außen" eingeben und im Speicher ablegen
- Programme und Daten im selben Speicher:
  - Unterteilt in gleich große Einheiten
    - Register, Speicherzelle
    - mittels Zahlen, sogenannter Adressen, durchnummeriert

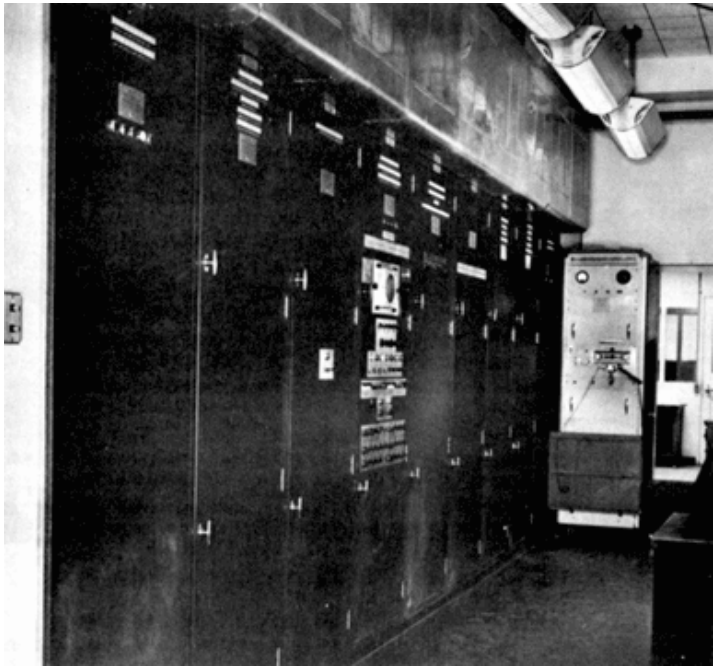
  $\Rightarrow$ Direkter Zugriff auf Speicherzellen
- Aufeinanderfolgende Befehle in aufeinanderfolgenden Speicherzellen
  - Ansprechen nächster Befehl: Erhöhen der Befehlsadresse um 1
  - Ausnahme: Sprungbefehle:
    - Unbedingte Sprungbefehle: `GOTO label:` setze Befehlsadresse auf `label`
    - Bedingte Sprungbefehle: `IF test THEN label:` setze Befehlsadresse auf `label`, falls `test` wahr ist, sonst erhöhe Befehlsadresse um 1
- Programme und Daten werden binär codiert

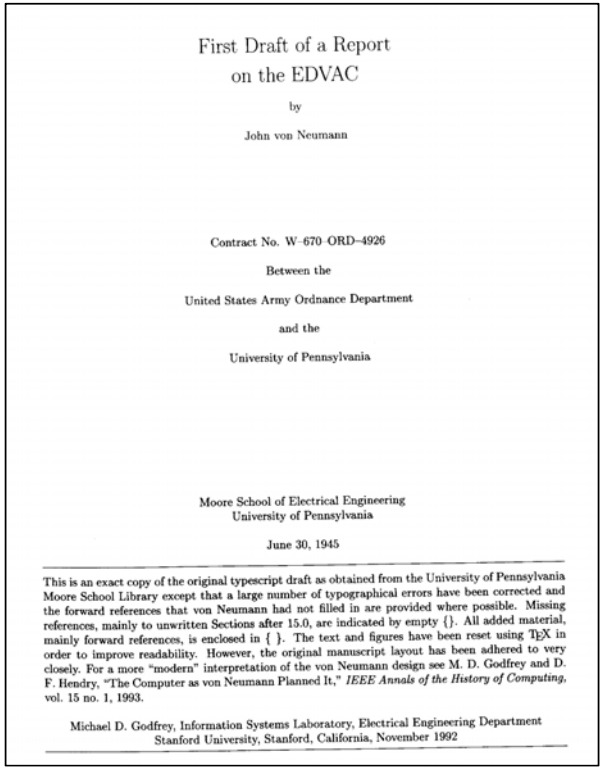
von-Neumann-Rechner (III): Prinzipien (II)

- Mehrzahl heutiger Computer folgen diesem Konzept, dem sog. von-Neumann-Prinzip bzw. von-Neumann-Architektur
- Funktionseinheiten heutiger von Neumann-Rechner:
  - Arbeitsspeicher
  - Rechenwerk
  - Steuereinheit
  - Ein- / Ausgabeeinheit
  - Interne Datenwege für den Informationsaustausch zwischen diesen Funktionseinheiten
- Aber: In der Regel
  - Speicherhierarchie (CPU/Register/Cache/Arbeitsspeicher)
  - Mehrere Rechen-/Steuerwerke
- Funktionsprinzip der von Neumann-Rechner:
  - Ausführung von Berechnungen durch Folgen globaler Zustandsänderungen (d.h. durch Folge von Änderungen des Speicherinhalts)

EDVAC

- EDVAC (Electronic Discrete Variable Arithmetic Computer):
  - Umsetzung von Neumann-Prinzip





First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W-670-ORD-4926

Between the

United States Army Ordnance Department

and the

University of Pennsylvania

Moore School of Electrical Engineering
University of Pennsylvania

June 30, 1945

This is an exact copy of the original typescript draft as obtained from the University of Pennsylvania Moore School Library except that a large number of typographical errors have been corrected and the forward references that von Neumann had not filled in are provided where possible. Missing references, mainly to unwritten Sections after 15.0, are indicated by empty {}. All added material, mainly forward references, is enclosed in { }. The text and figures have been reset using TEX in order to improve readability. However, the original manuscript layout has been adhered to very closely. For a more "modern" interpretation of the von Neumann design see M. D. Godfrey and D. F. Hendry, "The Computer as von Neumann Planned It," *IEEE Annals of the History of Computing*, vol. 15 no. 1, 1993.

Michael D. Godfrey, Information Systems Laboratory, Electrical Engineering Department
Stanford University, Stanford, California, November 1992

[http://web.soi.city.ac.uk/archive/image/lists/computers.html]
[https://web.mit.edu/STS.035/www/PDFs/edvac.pdf]

UNIVAC

- 1952: UNIVAC I :
  - Erster Großrechner in Serie (46 Stück)
  - Halbleiter-Transistoren statt Elektronenröhren
  - Vorteile:
    - Geringere Wärmeentwicklung, niedrigere Störanfälligkeit, kleinere Abmessungen
    - Ferritkernspeicher als Hauptspeicher, Magnetbänder als externe Speicher
    - 10000 Additionen/Sekunde
    - Speicherkapazität ca. 100 Bit/cm³

[http://www.computerhistory.org]

Weitere Entwicklung (I)

- Ab 50er Jahren:
  - Kommerzielle Rechnerproduktion
  - Einteilung in Generationen:
    - Erste Rechnergeneration bis 1955
    - Zweite Rechnergeneration bis 1962
    - Dritte Rechnergeneration bis 1970
    - Vierte Rechnergeneration bis 1985
    - Fünfte Rechnergeneration

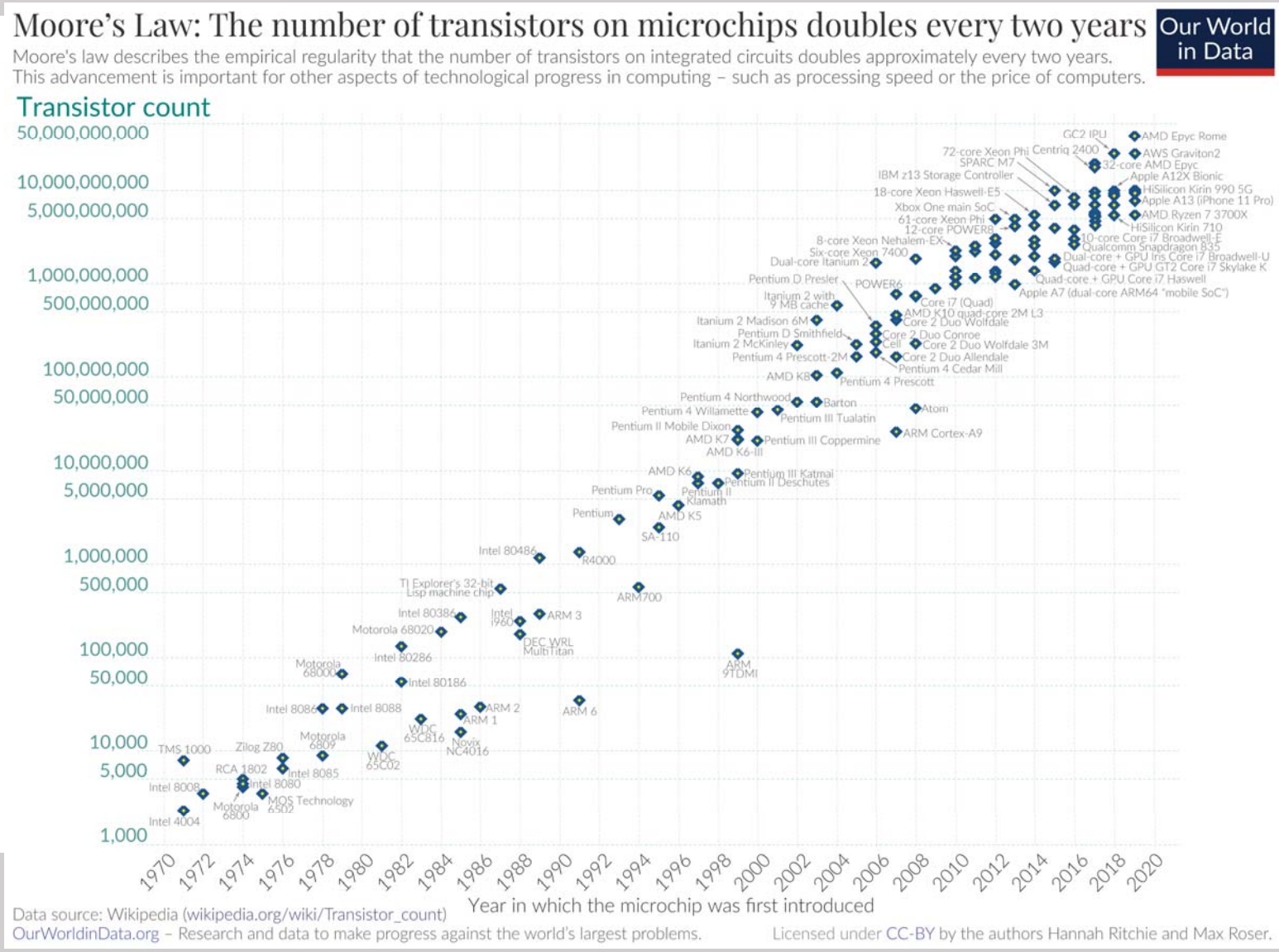
Weitere Entwicklung (II)

Gen.	Betriebssystem	Hardware	Software	Speicher
1	Keins, Einzelprogramme	Elektronenröhren	Maschinen-sprache	Lochkarte
2	Stapelverarbeitung	Transistoren	Assembler, FORTRAN	Magnetkern, Magnettrommel, Bänder
3	Stapelverarbeitung, Timesharing, Dialog	Integrierte Schaltkreise	Strukturierte Sprachen (C, Pascal)	Festplatte, Diskette
4	Portable BS, UNIX, MS/DOS	Hochintegrierte Schaltkreise, Mikroprozessoren	Problem-orientierte Sprachen	Festplatte, Diskette
5	Netzwerk-, Mehrprozessor-BS	VLSI	Objektorien-tierung	SAN

Weitere Entwicklung (III)

- 90er Jahre:
  - Immer weitere Integration
  - Immer höhere Rechengeschwindigkeiten
  - Immer größere Speicherkapazität pro Flächeneinheit
  - Moore‘s Law
- Ab 2000:
  - Erreichen prinzipieller physikalischer Grenzen
  - Neue Ideen/Konzepte:
    - Mehr-Kern-Systeme
    - GPU-Rechner
  - Mobile Systeme
  - Sensoren, IoT (Internet of Thing)
  - (Sehr) rechenintensive Systeme (KI, Big Data)

Moore‘s Law


Moore’s Law: The number of transistors on microchips doubles every two years
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years.
This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.
Our World in Data
Transistor count
50,000,000,000
10,000,000,000
5,000,000,000
1,000,000,000
500,000,000
100,000,000
50,000,000
10,000,000
5,000,000
1,000,000
500,000
100,000
50,000
10,000
5,000
1,000
GC2 IPU
AMD Epyc Rome
72-core Xeon Phi
Centriq 2400
AWS Graviton2
SPARC M7
32-core AMD Epyc
IBM z13 Storage Controller
Apple A12X Bionic
18-core Xeon Haswell-E5
HiSilicon Kirin 990 5G
Xbox One main SoC
Apple A13 (iPhone 11 Pro)
61-core Xeon Phi
AMD Ryzen 7 3700X
12-core POWER8
HiSilicon Kirin 710
8-core Xeon Nehalem-EX
10-core Core i7 Broadwell-E
Six-core Xeon 7400
Qualcomm Snapdragon 835
Dual-core Itanium 2
Dual-core + GPU Iris Core i7 Broadwell-U
Quad-core + GPU GT2 Core i7 Skylake K
Pentium D Presler
POWER6
Quad-core + GPU Core i7 Haswell
Itanium 2 with 9 MB cache
Apple A7 (dual-core ARM64 "mobile SoC")
Core i7 (Quad)
AMD K10 quad-core 2M L3
Itanium 2 Madison 6M
Core 2 Duo Wolfdale
Pentium D Smithfield
Core 2 Duo Conroe
Itanium 2 McKinley
Cell
Core 2 Duo Wolfdale 3M
Pentium 4 Prescott-2M
Core 2 Duo Allendale
Pentium 4 Cedar Mill
AMD K8
Pentium 4 Prescott
Pentium 4 Northwood
Barton
Pentium 4 Willamette
Atom
Pentium III Tualatin
Pentium II Mobile Dixon
ARM Cortex-A9
AMD K7
Pentium III Coppermine
AMD K6-III
AMD K6
Pentium III Katmai
Pentium II Deschutes
Pentium Pro
Pentium II Klamath
Pentium
AMD K5
SA-110
Intel 80486
R4000
TI Explorer's 32-bit Lisp machine chip
ARM700
Intel 80386
Intel i960
ARM 3
Motorola 68020
DEC WRL MultiTitan
Intel 80286
ARM 9TDMI
Motorola 68000
Intel 80186
Intel 8086
Intel 8088
ARM 2
ARM 1
ARM 6
WDC 65C816
Novix NC4016
Motorola 6809
Zilog Z80
WDC 65C02
TMS 1000
RCA 1802
Intel 8085
Intel 8080
Intel 8008
MOS Technology 6502
Motorola 6800
Intel 4004
1970
1972
1974
1976
1978
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
Year in which the microchip was first introduced
Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world’s largest problems.
Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

Zusammenfassung (I)

- Aussagen(logik):
  - Aussagen
  - Verknüpfung von Aussagen
  - Wahrheitstafeln
  - Logische Äquivalenzen
  - Implikation

- Informationsdarstellung:
  - Maschinenwort
  - Stellenwertsysteme (Dual, Oktal, Dezimal, Hexadezimal)
  - Umrechnen zwischen Systemen
  - Codierung
  - ASCII
  - Unicode

Zusammenfassung (II)

- Zahlendarstellung im Rechner:
  - Ganze Zahlen:
    - Excess-3
    - Komplementdarstellungen
  - Reelle Zahlen:
    - Festkommadarstellung
    - Gleitkommadarstellung, insb. IEEE 754-Norm

- Geschichte der Rechner:
  - Historische Entwicklung
  - von Neumann-Architektur: Aufbau und Prinzipien

Aufgaben (I)

- **Aufgabe 1**
  a) Beweisen Sie mittels Wahrheitstafeln das erste Gesetz von de Morgan!
  b) Es sei die folgende Variablenbelegung gegeben: x=3, y=5, z=7.
  Sind die folgenden Aussagen wahr oder falsch?
  b1) x<y
  b2) ((2*x+y) < (y+z)) OR (3*z=21)
  b3) ((2*x+y = 11) AND (y=z)) OR (3*z=19)
  c) Geben Sie an, ob die folgenden Sätze wahre oder falsche oder keine Aussagen sind!
  c1) „Eine Primzahl ist eine ganze Zahl, die ohne Rest nur durch sich selbst und 1 teilbar ist!“
  c2) „Komm her!“
  c3) „Die Erde ist eine Scheibe.“
  c4) „Bergab geht’s schneller als zu Fuß.“

Aufgaben (II)

d) Gegeben seien die Aussagen:

p: 2 ist die kleinste Primzahl.

q: 8 > 5.

r: Alle Quadratzahlen sind gerade.

Geben Sie die Wahrheitswerte der folgenden Verknüpfungen an:

d1) $(p \wedge q) \vee r$

d2) $(p \vee q) \wedge r$

d3) $(p \wedge (\neg q)) \vee (p \wedge r)$

e) Stellen Sie die Implikation $p \Rightarrow q$ mittels der Operatoren $\wedge$, $\vee$ und $\neg$ dar und beweisen Sie dies mit einer Wahrheitstafel.

Aufgaben (III)

- **Aufgabe 2**

  Neben den in der Vorlesung vorgestellten Codes Binärcode, Oktalcode, "normale" Dezimal zahlendarstellung und hexadezimale Zahlen-darstellung, gibt es weitere Codes.
  Im BCD-Code (Binary coded decimal) wird z.B. jede Stelle einer Dezimalzahl als die entsprechende 4-bit breite Binärzahl kodiert.
  Beispiel: 59(10) = 0101 1001(BCD).
  Der Gray-Code ist ein sog. einschrittiger Code, d.h. beim Wechsel von einer Zahl zur nächsten wechselt jeweils nur ein Bit. Die Tabelle gibt die konkreten Codierungen an.

Dezimal	Binär	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Aufgaben (IV)

Führen Sie die folgenden Umrechnungen durch:

a) 111111111(2) nach BCD
b) AF9(H) nach Binär
c) 10111111(2) nach Dezimal
d) 109(10) nach Hexadezimal
e) 1110(Gray) nach Dezimal
f) 657(8) nach Hexadezimal
g) FFF(H) nach Dezimal
h) 10011000(2) nach Hexadezimal
i) 126.375(10) nach Binär
j) AF5F4(16) nach Oktal

Aufgaben (V)

- **Aufgabe 3**

  Berechnen Sie im jeweiligen Zahlensystem:

  a) 11001(2) + 11001(2)
  b) 11101(2) – 1011(2)
  c) 9FC3(16) + 25AC(16)
  d) 11001(2) * 11001(2)

Aufgaben (VI)

- **Aufgabe 4**

  a) Stellen Sie die folgenden Zahlen als 8-bit-Dualzahlen im 2-Komplement dar!

  a1) –22

  a2) –57

  a3) 66

  a4) 312

  b) Lösen Sie die folgenden Rechnungen mit 8-bit-Dualzahlen im 2-Komplement!

  b1) 42 + 17

  b2) 42 – 17

  b3) – 42 + 17

  b4) – 42 – 17

Aufgaben (VII)

c) Fritz ist seit einiger Zeit interessierter Hobby-Programmierer. Er berichtet, dass er zwei positive Zahlen festgelegt hat. Bei der Addition hat er eine negative Zahl als Resultat bekommen.
Erläutern Sie dieses Phänomen!

- **Aufgabe 5**
  Überlegen Sie sich, an welchen Stellen der von Neumann-Architektur Engpässe auftreten können und wie diese behoben werden können!

Aufgaben (VIII)

- **Aufgabe 6**

  a) Stellen Sie die Dezimalzahlen

  a1) 173,265625

  a2) -3216,9375

  a3) -7,0078125

  a4) 1,25

  als binäre Gleitkommazahlen nach IEEE 754-Standard mit einfacher Genauigkeit dar!

Aufgaben (VIII)

b) Stellen Sie die folgenden binären Gleitkommazahlen nach IEEE 754-Standard mit ein facher Genauigkeit als Dezimalzahlen dar:

b1) 10010100100010...0

b2) 011111111000...0

b3) 011111111010...0

b4) 0111111111000010...0

b5) 011110111000...0

b6) 000000001110...0