- **Schannon'sche Informationstheorie**:
  - *Nachricht* = Folge von Zeichen, die von einem Sender an einen Empfänger übertragen werden. $w(a)$: Wahrscheinlichkeit für das auftreten des Zeichens a.
  - *Informationsgehalt* $I(a)$ einzelner Zeichen a: $I(a) = \log_2 \frac{1}{w(a)} = -\log_2 a$ [umso höher, je unwahrscheinlicher a ist]
  - *Informationsgehalt* $I(a_1, ..., a_n)$ einer Nachricht: $I(a_1, ..., a_n) = \sum_{i=1}^{n} I(a_i)$
  - *Entropie* $H(A, w)$ (mittlerer Informationsgehalt): Sei $A = \{a_1, .., a_n\}$ Alphabet und $w(a_i)$ eine Wahrscheinlichkeitsverteilung über A. Dann gilt: $H(A, w) = \sum_{i=1}^{n} w(a_i) \cdot I(a_i)$ [hohe Entropie $\leftrightarrow$ Gleichverteilung]
  - *mittlere Codewortlänge:* $L(A, w, C) = \sum_{i=1}^{n} w(a_i) \cdot |C(a_i)|$
  - *Schannon'sches Codierungstheorem:* $H(A, w) \leq L(A, w, C)$
- **Zeichenkodierung**:
  - *Huffman-Codes:* idealer Präfixcode (Präfixcode: Keine zwei codierten Zeichen beginnen mit derselben Zeichenfolge = Präfix) bei bekannter Häufigkeitsverteilung.
  - *Lempel-Ziv-Welch:* Beispiel-String: LZWLZ78LZ77LZCLZMWLZAP

Eingabe	erk. Muster	Neurer Eintrag
$\underline{L}ZWLZ78LZ77LZCLZMWLZAP$	L	$LZ(=256)$
$\underline{\underline{Z}}WLZ78LZ77LZCLZMWLZAP$	Z	$ZW(=257)$
$\underline{\underline{W}}LZ78LZ77LZCLZMWLZAP$	W	$WL(=258)$
$\underline{\underline{LZ}}78LZ77LZCLZMWLZAP$	$LZ(=256)$	$LZ7(=259)$
$\underline{\underline{7}}8LZ77LZCLZMWLZAP$	7	$78(=260)$

Eingabe	Ausgabe	Erweiterung	Neuer Eintrag
L	L		
Z	Z	Z	LZ (256)
W	W	W	ZW (257)
< 256 >	LZ	L	WL (258)
7	7	7	LZ7 (259)

Codierter String: $LZW < 256 > 78 < 259 > 7 < 256 > C < 256 > M < 258 > ZAP$

Problem $K\omega K$-Kodierung: Problem beim dekodieren, da Code-Tabelle hier immer eins nachläuft! trick: Verwende beim Auftreten des $K\omega K$-Falles das erste zeichen der letzten Ausgabe als Erweiterungszeichen ($\rightarrow$ gesuchter Code kann erstellt werden).
Beispiel APAPAPAPAPAP $\rightarrow AP < 256 >< 258 >< 257 >< 260 >$

- **Redundante Codes:**
  - *Abstand:* $dist(v, w)$:="Anzahl der unterschiedlichen Bits"
  - *Abstand des Codes C:* $dist(C) := \min \{dist\,(c(a_i), c(a_j))\,|a_i, a_j \in A \text{ mit } a_i \neq a_j\}$ = "minimaler Abstand zwischen zwei Code-Wörtern"
  - Ein Code c fester Länge ist genau dann *k-fehlererkennend*, wenn $dist(c) \geq k + 1$ gilt.
    Beispiel: Parity-Check: füge ein Bit (0/1) an den Code an, sodass er Parity-Check (=true, wenn gerade Anzahl 1-Bits) erfüllt ist. $\Rightarrow$ 1-fehlererkennend
  - Ein Code c fester Länge ist genau dann *k-fehlererkorrigierend*, wenn $dist(c) \geq 2k + 1$ gilt.
    Sei $c : A \rightarrow \{0, 1\}\, m + r$ ein 1-fehlerkorrigierender Code fester Länge von A ($|A| = 2^m$). Dann gilt $r \leq 1 + \lfloor \log_2 m \rfloor$
- **pos. Festkommazahl** zur Basis d $(d)_b = (d_n d_{n-1} ... d_1 d_0 d_{-1} ... d_{-k})_b$: $< d >= \sum_{i=-k}^{n} b^i \cdot d_i$
- **vorzeichenbehaftete Festkommazahlen:**
  - *Betrag und Vorzeichen:* d_n gibt das Vorzeichen an. es gilt: $< d >= (-1)^{d_n} \cdot \sum_{i=-k}^{n-1} b^i \cdot d_i$
    größte Zahl: $2^n - 2^{-k}$, kleinste Zahl: $-(2^n - 2^{-k})$, zwei Darstellungen für Null (z.B. $< 000 >= 0 =< 100 >$).
  - *Komplementdarstellungen:*

1-Komplement: $< d >= \sum_{i=-k}^{n-1} 2^i \cdot d_i - d_n(2^n - 2^{-k})$

a	000	001	010	011	100	101	110	111
$[a]_1$	0	1	2	3	-3	-2	-1	0

$[\overline{d_n ... d_{-k}}]_1 = -[d_n ... d_{-k}]_1$

2-Komplement: $< d >= \sum_{i=-k}^{n-1} 2^i \cdot d_i - d_n 2^n$

a	000	001	010	011	100	101	110	111
$[a]_1$	0	1	2	3	-4	-3	-2	-1

kleinste Zahl: -2^n; größte Zahl: $2^n - 2^{-k}$
Invertieren: alle Bits komplemmentieren und an der niederwertuigsten Stelle 1 addieren. (Bsp: $k = 0, n = 3$: $-[011]_2 = [\overline{011} + 001]_2 = [100 + 001]_2 = [101]_2 = -3$)

- **Gleitkommadarstellung** eine Zahl z mit `single`/`double` precission nach IEEE 754:

31	30..23	22..0
S	Charakteristik C	Mantisse M
S	$e_7..e_0$	$m_{-1}..m_{-23}$

(`double`: 64bit, 11bit Exp., 52bit Mant.)
Es gilt: ($E := C - BIAS$; $BIAS = 2^{n-1} - 1$, n Exponentenbits)

C	M	Wert
255 (2047)	0	INF (∞)
255 (2047)	$\neq 0$	NaN
0	0	0
0	$\neq 0$	$(-1)^S \cdot \left(\sum_{i=-1}^{-k} m_i 2^i\right) \cdot 2^{-126}$
$0 < C < 255$ (2047)	beliebig	$(-1)^S \cdot \left(1 + \sum_{i=-1}^{-k} m_i 2^i\right) \cdot 2^{C-BIAS}$

Addition: 1. Angleichen des kleineren an den größeren Exponenten 2. Addition der Mantissen 3. Normalisierung, Rundung ...
Multiplikation: 1. Multiplizieren der Vorzeichen 2. Multiplizieren der Mantissen 3. Addition der Exponenten: $Char_{erg} = (Char_1 - BIAS) + (Char_2 - BIAS) + BIAS = Char_1 + Char_2 - BIAS$ 4. Normalisierung, Rundung ...

- **Boole'sche Algebren:** $(B, +, \cdot, \neg)$ heißt Boole'sche Algebra, wenn gilt:

Kommutativität	$x + y = y + x$	$\Rightarrow$ *de Morgan-Regel*	$\overline{(x + y)} = \overline{x} \cdot \overline{y}$
Assoziativität	$x + (y + z) = (x + y) + z$	$\Rightarrow$ *Consensus-Regel*	$(x \cdot y) + (\overline{x} \cdot z) = (x \cdot y) + (\overline{x} \cdot z) + (y \cdot z)$
Absorption	$x + (x \cdot y) = x$	*Komplementregel*	$x + (y \cdot \overline{y}) = x$
Distributivität	$x + (y \cdot z) = (x + y) \cdot (x + z)$		

Formeln gelten für $(+, \cdot)$ und $(\cdot, +)$, also auch mit vertauschten Operatoren

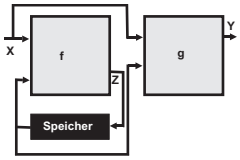
- **Glitches:**
  - statischer 0-Hazard: Ausgangssignal geht kurzzeitig auf 0 (0 = falscher Zustand)
  - statischer 1-Hazard: Ausgangssignal geht kurzzeitig auf 1 (1 = falscher Zustand)

– bei disjunkten Normalformen entstehen stat. 1-Hazards beim Übergang von Eingangsbelegung α zu β, genau dann, wenn: 1. $f(\alpha) = f(\beta) = 1$ 2. kein Monom existiert, das sowohl für α, als auch für β den Wert 1 hat. 3. Die Monome $m_1(\alpha) = 1$ schalten schnell auf 0 und die Monome $m_2(\beta) = 1$ schalten langsam auf 1.

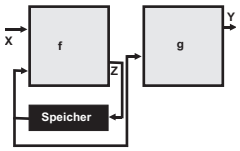
⇒ Überdecke alle Monome im K-Diagramm paarweise mit mindestens einem weiteren Monom.

- **Addier-Schaltungen:**
  - *Carry-Ripple-Addierer:* $C(CR_n) = n \cdot C(FA) = 5 \cdot n; \quad depth(CR_n) = 3 + 2 \cdot (n-1)$
  - *Carry-Save-Addierer:* paralleles Berechnen des Ergebnisses für $C_{in} = 0$ und $C_{in} = 1$ ⇒ Auswahl des richtigen Ergebnisses anhand untergeordnetem Carry
    $C(CSA_n) = 10 \cdot n^{\log 3} - 3 \cdot n - 2 = O(n \cdot \log n); \quad depth(CSA_n) \leq 3 \cdot \log n - 3 = O(\log n)$
  - *Carry-Lookahead-Addierer:* schnelle (parallele) Berechnung aller Übertragsbits $c_{-1}...c_{n-1}$ ⇒ $s_i = a_i \oplus b_i \oplus c_{i-1}$
    $C(CLA_n) \leq 11n = O(n); \quad depth(CSA_n) \leq 4\log(n) + 2 = O(\log n)$
  - *Überlauferkennung für 2-Komplement-Zahlen:*
    Überlauf ⇔ $a_n = b_n \neq s_n$

- **Schaltwerke:** Rückkopplungen sind erlaubt (im Gegensatz zu Schaltnetzen):
  - Flackern/metastabiler Zustand beim RS-FlipFlop, wenn $\overline{R} = \overline{S} = 0$, da dann $\overline{Q} = Q = 1$
  - **Automaten:** X: Eingabevektor; Y: Ausgabevektor; Z: Zustandsvektor
    Initialisierung beachten !!!

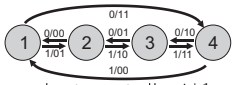


Mealy-Automat: $Y = f(X,Z); \quad Z = g(X,Z)$



Moore-Automat: $Y = f(Z); \quad Z = g(X,Z)$

  - **Schaltwerksentwurf:**
    1. Aufgabestellung: Modulo-4 Vorwärts-Rückwärtszähler
    2. Zustandsdefinition: Zustände $1, 2, 3, 4$ erforderlich
    3. Zustandsgraph
    4. Redundanzen eliminieren
    5. Zustandskodierung: $1 \rightarrow 00$; $2 \rightarrow 01$; $3 \rightarrow 10$; $4 \rightarrow 11$
    6. Zustandstabelle
    7. Boole'sche Ausdrücke für Übergangsfunktionen f, g, Minimierung ...
    8. Schaltbild



x	z_1^t	z_2^t	z_1^{t+1}	z_2^{t+1}	y_1	y_2
1	0	0	0	1	0	1
1	0	1	1	0	1	0
1	1	0	1	1	1	1
1	1	1	0	0	0	0
0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	1	0	0	1	0	1
0	1	1	1	0	1	0

- **Architektuen:**
  - *RISC:* einfache Befehle, in einem Schritt ausfhrbar, wenige Adressierungsarten (LOAD/STORE), viele Register
  - *CISC:* komplexe/mächtige Befehle, mehrere Datentypen, viele Adressierungsarten u.U. mit Mikroprogrammierung der Befehle, wenige Register, Befehle haben untersch. Ausfhrungszeiten
  - *von-Neumann:* gemeinsamer Programm- und Daten-Speicher — *Harvard:* getrennter Programm- und Daten-Speicher
  - *MIPS:* Mehrzyklen-Maschine mit RISC-Befehlsatz und LOAD/STORE-Architektur. (Phasen: instruction fetch, decode, execute, write back) ⇒ mehrfache Verwendung der ALU (auch zur Adressrechnung), Befehlsregister, Steuerautomat

- **Pipelining:**
  - *Beispiel:* instruction fetch – Decodieren/Operanden lesen (Reg) – Ausführung/Adressberechnung – Speicherzugriff (WR) – write-back (REG)
  - *Datenhazards:* Zugriff auf Daten, die noch in der Pipeline stecken, aber nicht im Register ⇒ forwarding der Daten (Hardware, auf Execute-Stufe), Einfügen von `NOP`s (Software) bzw. bubbles/waitstates (Hardware)
  - *Branch-Hazards:* einfrieren der Pipeline/`NOP`s einfügen (stalling), spekulative branch-prediction (z.B. feste Annahme durch Compiler/Hardware, Statistik → Schleifen!) ⇒ Pieline-Inhalt muss u.U. verworfen werden, dynamic scheduling (Operationsreihenfolge wird erst zur Laufzeit festgelegt)