
Einführung in die Informatik B

2. Schaltnetze und Schaltwerke

2.1 Bool'sche Algebra (\cong Aussagenlogik [InfoA])

- $(M; \cdot, +, ')$ mit M enthält mind. 2 Elemente und Operationen \cdot , $+$ und $'$
- **Huntington'sche Axiome**
 - Kommutativgesetz
 - Distributivgesetz
 - neutrale Elemente: $a \cdot 1 = a$; $a + 0 = a$
 - Komplementgesetz: $a \cdot a' = 0$; $a + a' = 1$
- **Dualitätsprinzip** (vertausche $+$, \cdot , 0 und 1)
- 0-1-Gesetze: $a \cdot 0 = 0$; $a + 1 = 1$ (weitere Gesetze \cong Aussagenlogik)
- Normalformen kNF und dNF, wie Aussagenlogik
- **kanonische** (= ausgezeichnete) **Normalformen** \Leftrightarrow jeder Term enthält alle Variablen (Minterme bei Konjunktion, Maxterme bei Disjunktion)
Herstellung durch Aufpumpen (Erweiterung mit $(a + a')$ bzw. $(a \cdot a')$)
 \Rightarrow Verfahren von Karnaugh-Veitch
- Gleichheit (syntaktisch) \neq Äquivalenz (semantisch)
- Aussagenlogik ist kleinstmögliche Bool'sche Algebra

2.2 Schaltalgebra (\subseteq Bool'sche Algebra)

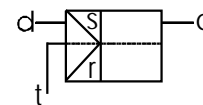
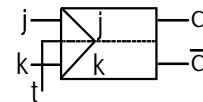
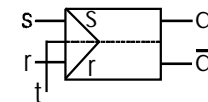
- Realisierung
 - Bool'sches Produkt durch Reihenschaltung
 - Bool'sche Summe durch Parallelschaltung
 - Bool'sches Komplement durch Ruhekontaktschaltung
- Verknüpfungsbasen $\{\wedge, \vee, \neg\}$, $\{\wedge, \neg\}$, $\{\vee, \neg\}$, $\{\downarrow\}$ (NOR), $\{\}\}$ (NAND)

2.3 Schaltungen

- **Gatter**
 - NOT-Gatter: $\text{---}[\text{O}]\text{---}$ (oder kurz: $\text{---}\text{O}\text{---}$)
 - OR-Gatter: $\text{---}[\geq 1]\text{---}$
 - AND-Gatter: $\text{---}[\&]\text{---}$
 - XOR-Gatter: $\text{---}[2k+1]\text{---}$
- **Schaltnetze**
 - $a = (a_1, \dots, a_n)$: Werte an den Eingängen
 - f_j : Schaltfunktionen
 - $f_j(a)$: Werte an den Ausgängen
 - kombinatorisch (nur abhängig von Eingangszuständen)
 - Beispiele
 - Halbaddierer (HA)
 - Volladdierer (VA)
 $s = (a \text{ XOR } b) \text{ XOR } \ddot{u}$
 $\ddot{u}' = (a \wedge b) \vee (a \wedge \ddot{u}) \vee (b \wedge \ddot{u})$
 - Ripple-Carry-Adder (Durchlaufender Addierer)
 - Carry-Select-Adder (\cong Divide-and-Conquer)

• **Schaltwerke**

- innere Zustände wichtig: z: innerer Zustand
- $g(a, z)$: Übergangsfunktion (innere Zustände die rückzuführen sind)
- τ : Verzögerungsglied
- stabil $\Leftrightarrow g(a, z) = z$
instabil, sonst
- synchrone Schaltwerke
 - diskrete Zeitpunkte $t \cdot \tau$
 - wähle $\tau >$ Schaltzeit
 - Takt wirkt wie eine Schleuse (zwei komplementäre Signale)
- Schaltwerke zur Speicherung
 - Forderungen:
 - mindestens 2 stabile Zustände (FF1)
 - definierte Einstellungen durch Eingangssignale (FF2)
 - Speicherinhalt muß (negiert oder nicht negiert) am Ausgang zur Verfügung stehen
 - asynchroner-RS-Flip-Flop
 - Reset, Set
 - 2 NOR-Gatter: $q = s \bar{r} \vee q^* \bar{r}$
 - synchroner-RS-Flip-Flop
 - zusätzlich 2 mit t getackete AND-Gatter
 - synchroner-JK-Flip-Flop
 - zusätzlich $j=k=1$ erlaubt
 - MS-RS-Flip-Flop
 - Master – Slave
 - nur ein Eingang d
 - $t=1$: Übernahme in Master; $t=0$ Weitergabe an Slave
 - Register
 - geordnete Menge von synchronen Flip-Flops mit identischer Taktleitung
 - seriell: Schieberegister (für Addition geeignet)
 - parallel: Speicherregister (mit Read/Write Signalen)



2.4 physikalische Realisierung

- Halbleiter
 - Transistoren (n-MOS, p-MOS)
 - zwischen *Source* und *Drain* bildet sich p- oder n-Kanal

| | n-MOS | p-MOS |
|----------------|---------------------|---------------------|
| Signal am Gate | | |
| 0 | n-Kanal gesperrt | p-Kanal durchlässig |
| 1 | n-Kanal durchlässig | p-Kanal gesperrt |

- CMOS- Technologie (n-MOS und p-MOS)
 - Inverter (Parallelschaltung von n-MOS und p-MOS)
 - NAND- Gatter (Schleife oben)
 - NOR- Gatter (Schleife unten)
 - Transfer- Gatter
 - Verzögerungsglied (2-Phasen-Takt)

2.5 Hardware Entwurf

- Vorgehensweise
 - Abstrakte Ebene
 - Register- Transfer- Ebene
 - Gatter- Ebene
 - Transistor- Ebene
 - Layout- Ebene
 - Fertigung
- Beschreibung durch **VHDL**:
 - extern beobachtbares, erwünschtes Verhalten
 - Systemstrukturen
- ASIC (application specific intergrated circuit)
- FPGA (field programmable gateway) \Rightarrow gewinnt an Bedeutung

2.6 Binary Decision Diagrams (BDD)

- Reduzierung
 - verschmelze identische Teilgraphen (z.B. jeweils nur eine "0" und "1")
 - eliminiere einen Knoten, falls beide Kanten auf selben Nachfolger zeigen
- **geordnete BDD (OBDD)**:
 - \Leftrightarrow Reihenfolge der Variablen auf dem Weg ist immer gleich
- unterschiedliche Reihenfolge der Variablen \Rightarrow unterschiedliche (O)BDDs

3. Codierung

3.2 Code und Codierung (Grundbegriffe)

- **Ordnung**
 - totale Ordnung: (A, \times) ist geordnetes Alphabet (Hinweis: Gleichheit nicht erlaubt)
 - $\text{ORD}(a)$: Position von a
 - $\text{CHR}(i)$: Zeichne an Position i
 - lexikographische Ordnung in A^* (\neq numerische Ordnung)
- **Codierung**: $c: A \rightarrow B^*$ ist eine Abbildung
 - Code: Bildbereich $c(A) := \{ c(a) \mid a \in A \}$
 - Codewort: alle $b \in c(A)$
 - Chiffrierung: $c: A \rightarrow B$
 - Block Codierung: $c: A \rightarrow B^n$
 - Binärcodierung: $c: A \rightarrow \text{IB}^*$ mit $\text{IB} = \{0,1\}$
 - n-Bit-Codierung: $c: A \rightarrow \text{IB}^n$
 - natürliche Fortsetzung: $c^*: A^* \rightarrow B^*$ (Hinweis: $c^*(wa) = c^*(w)c(a)$)
- **injektiv** $\Leftrightarrow \forall x_1, x_2 \in X: (x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2))$
injektiv $\Rightarrow \exists$ Umkehrabbildung zu c
- **Fano-Bedingung** erfüllt \Leftrightarrow kein Codewort ist Anfang eines anderen Codewortes
 $\Leftrightarrow \forall u, w \in c(A): \forall v \in B^*: (w = uv \Rightarrow v = \lambda)$
 c^* injektiv $\Rightarrow c$ injektiv
 c injektiv $\wedge c$ erfüllt Fano-Bedingung $\Rightarrow c^*$ injektiv
- **Fehlererkennung und -korrektur**
 - **Hammingabstand**: $h(x,y) = \sum h(x_i, y_i)$ mit $h(x_i, y_i) = 1$, falls $x_i \neq y_i$
entspricht: Anzahl der Stellen in denen sich x und y unterscheiden
 - **Hammingzahl**: $h_c := \min \{ h(x,y) \mid x,y \in c(A), x \neq y \}$
 - c ist **k-Fehler erkennbar** $\Leftrightarrow h_c \geq k + 1$
d.h. durch Verfälschung von bis zu k Stellen eines Codewortes kann kein anderes Codewort entstehen
 - c ist **k-Fehler korregierbar** $\Leftrightarrow h_c \geq 2k + 1$
d.h. aus einem durch Verfälschung von bis zu k Stellen eines Codewortes entstanden Wortes, kann das richtige Codewort eindeutig ermittelt werden
 - Parity-Bit: Anzahl der "1"-Bits gerade oder ungerade \rightarrow odd/even-Parity-Check
 - mehrere Prüfbits: mehrere Parity-Bits für bestimmte Stellen des Codewortes
- **Häufigkeitsabhängige Codierungen**
 - Voraussetzungen: Häufigkeitsverteilung bekannt $\wedge c$ injektiv, erfüllt Fano- Bedingung
 - Codelänge: $L_{A,p}(c) = \sum_{a \in A} p(a) |c(a)|$
 c optimal $\Leftrightarrow L_{A,p}(c)$ minimal
 - **Huffman- Algorithmus** (immer Elemente mit niedrigsten Wahrscheinlichkeiten zu einem Element zusammenfassen)
- Kryptographie, Kryptoanalyse

| | |
|---|--|
| • Cäsar- Chiffre (+k dazu) | |
| • Tabellen- Chiffre (anhand Tabelle) | |
| • Vigenere- Chiffre (Schlüssel und Tabelle) | |
| • Public-Key- Kryptosysteme | |

P: Public Key, S: Private Key
RSA- Kryptographie: $m^{PS} \bmod N = m$

3.3 Darstellung von Zeichen, Ziffern und Zahlen

- Kriterien
 - i. technische Realisierung einfach
 - ii. Codierung und Decodierung schnell und einfach
 - iii. Operationen schnell und einfach durchzuführen
- Zeichensätze
 - n-Bit-Codes
 - ASCII
 - ABCDIC
- Ziffern
 - Tetradencodierung als spezielle 4-Bit Codierung
 - BCD- Codierung (Dezimalsystem \rightarrow Binärsystem, Stellenwertigkeit [8421])
 - Exzeß-3-Codierung (d+3) \rightarrow "0000" und "1111" nicht zulässig
 - Aiken- Codierung (d, für {0,...,4} \wedge d + 6 sonst) \rightarrow symmetrisch
 - 5-Bit Codierung
 - 2-aus-5-Codierung (immer 2 Stellen "1" setzen, alle anderen "0", Stellenwertigkeit [74210])
 \Rightarrow 1-Fehler erkennbar
- Zahlen
 - allgemein:
 - $x = \sum b_i B^i$ mit B: Basis, x: Wert
 - $c_B(x) = (b_k b_{k-1} \dots b_1 b_0 b_{-1} \dots)_B$ heißt **B-adische Darstellung**
 - Umrechnungen nach dem **Horner Algorithmus**
 - Dekodierung aus B-adisch berechnen
 - $x \in \mathbb{N}_0$
 \Rightarrow Darstellung: $x = (\dots((b_k B + b_{k-1}) B + b_{k-2}) B \dots + b_1) B + b_0$
 - echter unendlicher Bruch
 \Rightarrow Darstellung: $x = (b_{-1} + (b_{-2} + (b_{-3} + \dots(b_{-m+1} + b_{-m} / B) / B) \dots) / B$
 - unechter endlicher Bruch
 \Rightarrow Kombination der beiden
 - Codierung berechnen
 - $x \in \mathbb{N}_0$

```

k := -1;
REPEAT
    k := k+1;
    b_k := x MOD b;
    x := x DIV b;
UNTIL x=0;

```
 - echter unendlicher Bruch

```

FOR i := 1 TO m DO
    x := x * b;
    b_{-k} := TRUNC(x);
    x := x - b_{-k};
END;

```
 - unechter endlicher Bruch
 \Rightarrow Kombination der beiden
 - Zahlendarstellung im Rechner
 - Dualsystem
 - Dezimalsystem
 - binärdezimale Codierung (Fortsetzung der BCD- Codierung, bei Addition ≥ 10 : $s := s + 6$;))
 - weitere Darstellungen

3.4 Dualdarstellung von Zahlen (B=2)

- **Natürliche Zahlen**
 - analog Schularithmetik, aber nicht abgeschlossen, Assoziativ- und Dissoziativgesetz gelten nicht
- **Ganze Zahlen**
 - Vorzeichen- Betrag- Darstellung
höchstwertiges Bit: "1" für negativ oder "0" für positiv
à symmetrisch, Berechnung zu aufwendig
 - Exzeß-q-Darstellung
 - $x = \sum_0^{n-1} b_i 2^i - q$ mit $q = 2^{n-1}$
 - Addition: Korrektur um $-c_{2,N}(q)$ notwendig
 - Multiplikation: aufwendig
 - 2- Komplement- Darstellung
 - $x = -b_{n-1} 2^{n-1} + \sum_0^{n-2} b_i 2^i$, d.h. *Kippen aller Bits und Addition von "1"*
 - Veranschaulichung: Zahlenring
 - Addition: $n > 0$: n Schritte gegen Uhrzeigersinn, $n < 0$: im Uhrzeigersinn
 - Subtraktion: Addition des Komplements
 - Multiplikation und Division: Rückführung auf Addition und Division
 - à unsymmetrisch, Überlaufbehandlung möglich/notwendig
 - 1- Komplement- Darstellung
 - $x = -b_{n-1} (2^{n-1} - 1) + \sum_0^{n-2} b_i 2^i$, d.h. *durch Kippen aller Bits*
 - Addition/Subtraktion: analog 2-Komplement
 - à symmetrisch, 2 Darstellungen der "0"
- **Reelle Zahlen**
 - Festpunktdarstellung
 - Gleitpunktzahendarstellung (GPZ)
 - $x = m B^e$ mit m: Mantisse, e: Exponent, B: Basis
 - normierte Gleitpunktzahlen (NGPZ) $\Leftrightarrow (1/B) \leq |m| \leq 1$
 - Addition/Subtraktion: $x_1 \pm x_2 = (m_1 \pm m_2 B^{e_2 - e_1}) B^{e_1}$
 - Multiplikation: $(m_1, e_1) \otimes (m_2, e_2) = (m_1 m_2, e_1 + e_2)$, falls $(1/B) \leq m_1 m_2 \wedge (m_1 m_2 B, e_1 + e_2)$ sonst
 - Dualdarstellung:
Mantisse: Vorzeichen- Betrag- Darstellung
Exponent: Exzeß- q- Darstellung (Charakteristik)
 - Probleme:
 - Abstände verdoppeln sich, wenn e um 1 anwächst
 - Ungenauigkeiten, Rundungsfehler bei der Darstellung
 - Überlauf/ Unterlauf des Exponenten
 - übliche Rechengesetze gelten i.A. nicht
 - "0" nicht darstellbar

4. Rechnerarchitektur und –organisation

4.1 von Neumann Rechner

- ist ein Rechnerarchitekturkonzept
- Bestandteile: Rechenwerk und Steuerwerk (= CPU), Speicherwerk, Eingabe-/Ausgabenwerk
- Eigenschaft: gemeinsamer Speicher für Programme und Daten
 - ↔ RAM Maschine: ist abstraktes Berechnungsmodell mit getrenntem Speicher für Programme und Daten
- Funktionsprinzip: Ausführung von Berechnungen durch Folgen globaler Zustandsänderungen
- Arbeitsspeicher:
 - gleichgroße, nummerierte (à Adresse) Speicherelemente
 - Byte- Maschine (Speicherelement = 1 Byte) ↔ Wort- Maschine (sonst)
 - wahlfreier Zugriff
 - Schema:
 - Adress Memory Port (AM)
 - Write Memory Port (WM)
 - Read Memory Port (RM)
 - Steuersignale:
 - A: Adress Strobe: Adresse übernehmen
 - D: Direction: Read/Write
 - T: Data Transfer Acknowledge
 - Speicherhierarchie: Register > Cache > Arbeitsspeicher (ROM + RAM)
- Rechenwerk
 - arithmetische, logische Operationen und Vergleiche
 - besteht aus Registern (z.B. Flip Flop) und Funktionseinheiten (z.B. Volladdierer)
- Steuereinheit
 - Instruktionsregister (IR): auszuführender Befehl
 - Befehlszähler (PC): Adresse des aktuellen Befehls
 - Aufgaben: FETCH à DECODE à EXECUTE
- Eingabe-/ Ausgabeeinheit
 - Steuerung
 - CPU als E/A Einheit
 - feste Zuordnung CPU – E/A Gerät
 - unterbrechungsgesteuerter E/A Vorgang ("Interrupts")
 - E/A Prozessor als E/A- Einheit
- Bus- Konzept
 - Bus = Datensammelweg; Übertragung von Daten, Befehlen, Kontroll- und Statusinformationen
 - Bandbreite: Anzahl parallel übertragbare Bits
 - Aufteilung
 - Datenbus: Übertragung nur von Daten (bidirektional)
 - Adressbus: stets zusammen mit Datenbus benutzen
 - Kontrollbus: für Steuersignale von/ zu CPU
 - Buszuteilung
 - zentrale Steuerung (Bussteeereinheit)
 - dezentrale Steuerung
 - parallele Anfrage (Sender bekommt Prioritäten)
 - Daisy-Chain (Verfügbarkeitssignal geht um, gemeinsame Meldeleitung)
 - zyklische Buszuteilung (Berechtigungssignal kreist)
 - CSMA (zusätzliche Kollisionserkennung und –behandlung)

4.2 von Neumann Engpaß

- **physikalischer Engpaß:** Transport zwischen CPU und Speicher dauert länger als Befehle
- **intellektueller Engpaß:** strenge Sequentialisierung aller Berechnungen
- Konzepte zur Vermeidung
 - **Datenstruktur- Rechner**
 - Verwendung von komplexen Datenstrukturen (z.B. ARRAY, Vektorrechnung)
 - semantische Kluft zwischen Daten im Speicher und Repräsentation dieser
 - **Assoziativer Zugriff auf Daten**
 - Suchregister, Maskenregister, Vergleichslogik
 - Vorteil: Suchvorgänge effizienter
 - Nachteil: komplexe Hardware (Vergleichslogik)
 - andere Organisationsprinzipien (funktional nicht prozedural)
 - **Datenflußrechner:** keine vorgegebene Reihenfolge → Datenabhängigkeitsanalyse
 - **Reduktionsmaschine:** nur was, nicht Reihenfolge festgelegt
 - Parallelverarbeitung
 - spezielle Prozessorteile mehrfach (feinkörnige Parallelisierung)
 - **Pipeline Rechner**
 - lineare Anordnung mehrerer spezialisierter Bearbeitungseinheiten (nacheinander ausgeführt)
 - Reduktion von $n\mu$ auf $n+(n-1)$ Takte
 - Einfunktions- Pipelining ↔ Mehrfunktions- Pipelining
 - **Feldrechner**
 - lineare oder matrixförmige Anordnung vieler identischer Bearbeitungseinheiten
 - z.B. für Bildverarbeitung
 - Operationsfolge invers zu Pipelinerechner
 - **Vektorrechner**
 - Vektor als skalarer Datentyp
 - vollständige Prozessorteile mehrfach (grobkörnige Parallelisierung)
 - **Multiprozessorsysteme (MPS)**
 - Job/Task Ebene
 - Art der Prozessoren: gleich ⇒ homogen, sonst inhomogen
 - Funktion der Prozessoren: gleich ⇒ symmetrisch, sonst unsymmetrisch
 - Art der Kopplung der Prozessoren:
 - stark (shared memory) ↔ schwach (nachrichtengekoppelt)
 - autonome Rechner
 - Verteilte Systeme (Rechnernetze!)
- Exkurs: Verbindungseinrichtungen
 - Ringleitung
 - Kopplung über zentralen Speicher / Bussystem
 - irreguläres Netzwerk (→ Internet)
 - reguläres Netzwerk
 - 2-dimensionales-Gitter
 - binärer Baum
 - Hypercube
 - Schmetterling
 - fat tree
 - Kreuzschienenverteiler

4.3 Mikroprozessoren

- monolithische Prozessoren
 - vollständige Rechner (Rechenwerk, Steuerwerk, Speicher) auf einem Chip
 - Unterscheidungsmerkmale: Arbeitsgeschwindigkeit, Wortlänge, Adreßraum, Technologie (n-MOS, CMOS, ...)
 - RISC: Reduced Instruction Set Computer: relativ kleine Menge relativ einfacher Befehle
 - CISC: Complex Instruction Set Computer: größere Menge komplexer Befehle
- Bitslice Mikroprozessoren
 - nur Teil des Rechenwerkes auf einem Chip
 - Rechenwerke beliebiger Wortlängen
 - relativ langer Befehlszyklus
 - veraltet

4.4 Speicher

- Eigenschaften:
 - (SP 1) Zugriff: wahlfrei, sequentiell oder block-adressierbar
 - (SP 2) Speicherkapazität
 - (SP 3) Zugriffszeit
 - (SP 4) Energieabhängigkeit: energieabhängiger Speicher verliert bei Stromausfall seinen Inhalt, energieunabhängiger nicht
- Organisation:
 - Cache: tag und Datenblock
- Externspeicher:
 - block-adressierbarer/ sequentieller Zugriff
 - permanente Speicherung
 - langsam, da mechanisch
- Magnetplattenspeicher
 - Spur: konzentrischer Kreis besteht aus Sektoren und Gaps
 - mehrere Spuren untereinander ergeben einen Zylinder
 - Adresse = (Zylinder#, Spur#, Sektor#)
- Magnetstreifenkarten (3 Spuren, z.B. VISA)
- Magnetbandspeicher
 - physikalischer Satz: rechteckige Matrix mit Zeilen: Spuren und Spalten: Frames
- optische Plattenspeicher
 - z.B. CD-ROM, CD- WORM, wiederbeschreibbare Platte, optische Speicherkarte
- Exkurs: Chipkarte
 - 8-bit MPR
 - Speicher (mit geheimem Teil)
 - Schnittstelle (8 Kontakte)

5. Programmiersprachen

- **Mikroprogrammierung**
 - Implementierung der Ausführung von Maschinenbefehlen
 - Mikrobefehle: einfachste Ablaufsteuerung
 - Setzen von Steuersignalen s_i
 - getacktet mit ϕ_1, \dots, ϕ_n
 - billiger und flexibler als direkte Verdrahtung
 - Mikroprogrammeinheit
 - Teil des Steuerwerks
 - Ablauflogik \rightarrow Mikrobefehlszeiger MPC \rightarrow Mikroprogramm Speicher MIS \rightarrow Mikroinstrukptionsregister MIR
 - horizontaler (direkter) \leftrightarrow vertikaler (codeirter) Mikrobefehl
 - Beispiel-Computer

| | | |
|--|---|--------|
| Phase 1: Verknüpfung mit ADDER | } | |
| Phase 2: Speichern des Ergebnisses in A, B, C, D, MDR, MAR | } | 22 Bit |
| Phase 3: Lesen/Schreiben in Hauptspeicher | } | |
| Phase 4: Adreßrechnung für nächste Anweisung | } | |
 - Phase 5: berechnete Mikro Speicheradresse \rightarrow MPC, nächste Anweisung \rightarrow MIR
 - Hinweis: bei direktem Sprung sind die ersten 10 Bit die Sprungadresse
 - Schreibweise:

| | |
|---|--|
| Teil 1: arithmetische Operation (Phase 1+2) | |
| Teil 2: Hauptspeicher Zugriff (Phase 3) | |
| Teil 3: Berechnung der Folgeadresse (Phase 4+5) | |
 - Bsp. $0 + 1 \rightarrow$ MAR; write; \langle MPC $\rangle + 1 \rightarrow$ MPC
- **Maschinensprache**
 - Maschinenbefehl: binäres Wort fester Länge
 - Maschinensprache: Menge aller Maschinenbefehle, niedrigste frei zugängliche Ebene
 - Aufbau: Operationscode + Operandenadressen \rightarrow 1 / 2 / 3- Adreß- Maschinen
 - Operationscode: LOAD, STORE, ADD, SUBTRACT, MULTIPLY, DIVIDE, JUMP, JUMPZERO, JUMPMSB, JUMPSUB, RETURN
 - komplexere Maschinensprache
 - Stapelverarbeitung }
 - verschiedene Adressierungsarten } RISC \leftrightarrow CISC
 - Gleitpunktzahlen }
 - Adressierungsarten: unmittelbare, absolute, indirekte, indizierte, relative, implizite Adressierung
- **Assemblersprachen**
 - symbolische Notation der Befehle
 - mnemonische Namen
 - Operandenadressen können Namen zugeordnet werden
 - Befehle können Marken zugeordnet werden
 - Pseudobefehle (EQU)
 - Festlegung der Programmanfangsadresse (ORG)
 - Assembler für Umsetzung in Maschinencode
 - Befehlsaufbau: Markenfeld, Operationsfeld, Operandenfeld, Kommentarfeld

- **problemorientierte (höhere) Programmiersprachen**
 - unabhängig von Hardware
 - orientiert an Semantik und Problemfeld
 - Beschreibung durch Syntax und Semantik
 - (Übersetzung bzw. Interpretierung der Sprache notwendig)
 - imperative Programmiersprachen
 - algorithmischen Charakter
 - an von Neumann Rechner orientiert
 - Bsp.: Modula2, BASIC, C
 - funktionale und applikative Programmiersprachen
 - funktionale Beschreibung
 - zeitliche Reihenfolge nicht festgelegt
 - Beschreibung von Abhängigkeiten
 - Bsp.: LISP, LOGO, Gofer
 - Vorteil: korrekte Programme leichter realisierbar
 - Nachteil: Effizienz
 - prädikative Programmiersprachen
 - Programme als Menge von Fakten und Regeln
 - für Programme, die sich in logischen Kalkül formulieren lassen
 - Problem: sehr ineffiziente Berechnung
 - Bsp.: PROLOG
 - objektorientierte Programmiersprachen
 - Objekte mit Eigenschaften, Methoden zusammengefaßt zu Klassen
 - Konzepte: Geheimnisprinzip, Vererbungsprinzip, dynamische Bindung
 - Bsp.: Eiffel, Java, C++
 - Aspekte der Sprachübersetzung
 - Assembler
 - Adreßrechnung relativ oder absolut
 - 2-PASS-Assembler
 - PASS1: Symboltabelle, Syntaxprüfung, Pseudobefehle ausführen
 - PASS2: Einsetzen Adresswerte, Generierung von Maschinencode
 - Interpreter
 - unmittelbare Ausführung
 - besonders Ausrichtung zum Dialogbetrieb
 - Bsp.: Perl, BASIC
 - Vorteil: bei Programmentwicklung: Fehlererkennung einfach
 - Nachteil: längere Rechenzeit \Rightarrow wegen Schleifen und Adressrechnung
 - \Rightarrow Kombination von Assembler und Interpreter optimal
 - Compiler
 - Übersetzung und Ausführung
 - Vorteil: beliebig häufige Ausführung
 - Nachteil: Fehlererkennung schwieriger
 - Phasen:
 - lexikalische Analyse (scanner)
Zerlegung des Programms in kleinste syntaktische Einheiten (tokens)
z.B.: $x < 1.5$ \Rightarrow Name < Real-Zahl
 - syntaktische Analyse (parser)
Erzeugung des Ableitungsbaumes, Fehlerbehandlung
 - semantische Analyse und Codegenerierung
Prüfung der Namensdeklarationen, Typverträglichkeit
Codegenerierung: Durchlaufen des Baumes und Generierung von Maschinencode

6. Betriebssysteme

- Betriebsmittel: alle Hard- und Softwarekomponenten, die zur Ausführung notwendig sind
- Mechanismen zur Steuerung
- Systemprogramme: übernehmen Verwaltungs-, Steuerungs- und Dienstleistungsaufgaben
 - Organisationsprogramme
 - Übersetzungsprogramme
 - Dienstprogramme
- Betriebssystem: Gesamtheit aller Systemprogramme
- wichtiges Entwurfsprinzip: Modularisierung \Rightarrow Schichtenhierarchie der Module
- Betriebsarten
 - Auftrag: Menge aller Teilaufträge
 - Teilauftrag: endliche Folge von Betriebssystemkommandos
 - **Stapelbetrieb (batch mode)**
 - kein Eingreifen möglich
 - vollständige Ausführung
 - Anwendungen: statistische Auswertungen \Rightarrow Maximierung des Durchsatzes
 - **Dialogbetrieb (time sharing mode)**
 - Teilhaberbetrieb: alle benutzen gleiches Programm (bsp.: Buchungssystem)
 - Teilnehmerprinzip: verschiedene Aufträge (Bsp. Programmentwicklung) \Rightarrow Minimierung der Antwortzeit
 - **Echtzeitbetrieb (real time mode)**
 - wie Dialogbetrieb jedoch Zeitrestriktionen realisiert durch Prioritäten
 - Bsp. Meß-/Regeltechnik, Kommunikation) \Rightarrow größtmögliche Verfügbarkeit gewährleisten
 - **Multiprogrammbetrieb (multi programm mode)**
 - zeitliche verzahnte Bearbeitung der Prozesse
 - Aufgaben: Aktivierung, Blockierung und gegenseitige Isolation \Rightarrow bessere Ausnutzung der Betriebsmittel: $T = T_E + T_V + T_A$ (vorher) $\hat{=} T \sim \max(T_i)$ (jetzt)
 - **Spoolbetrieb**
 - Pufferung von Ein- und Ausgabedaten auf Hintergrundspeicher
 - notwendig bei Multiprogrammbetrieb
 - Verwaltung der ein- und Ausgabe- Pufferbereiche auf Hintergrundspeicher
 - Druck Spooler
 - **Client-Server-Betrieb**
 - zentrale Systemfunktionen nur im Server, auf die mit Remote Procedure Call zugegriffen wird
 - Anstoß- Nachricht $\hat{=}$ Ausführung $\hat{=}$ Abschluß- Nachricht
 - i.A. mehrere Betriebsarten auf Rechenanlage möglich
 - nacheinander
 - gleichzeitig: gerechten Ausgleich zwischen konkurrierenden Anwendungen ermöglichen
- Prozeß
 - Prozeß : zeitlicher Ablauf einer Folge von Aktionen eines Rechners, die zu einer identifizierbaren funktionalen Einheit zusammengefaßt sind
 - Thread: leichtgewichtiger Prozeß; hat eigenen Kontrollfluß (wie Prozeß), aber gemeinsamen Adreßraum (Unterschied zu Prozeß)
 - Zustände
 - è initiiert noch nicht zur Bearbeitung zugelassen
 - è bereit: Prozessor noch nicht vorhanden, nur Betriebsmittel
 - è aktiv: wird ausgeführt
 - è blockiert: Prozeß wartet auf Eintritt eines Ereignisses
 - è terminiert: beendet; gibt Betriebsmittel frei
 - Auftragssteuerung (job sheduler): initiiert $\hat{=}$ bereit;
 - Prozessorverwaltung (CPU sheduler): bereit $\hat{=}$ aktiv, aktiv $\hat{=}$ bereit
 - Betriebsmittelverwaltung: aktiv $\hat{=}$ blockiert, blockiert $\hat{=}$ bereit, aktiv $\hat{=}$ terminiert

- konkrete Betriebssystemaufgaben
 - Zuteilung von Rechenzeit
 - einfache Verfahren: first come first served
 - Prioritätsgesteuerte Verfahren
Einteilung in Prioritätsklassen, z.B. shortest job first
 - Zeitscheibenverfahren
Zeitscheibe Z (time slice)
Schedule: preemptiv (Abbruch vor Ende möglich) \leftrightarrow nonpreemptiv
 - Hauptspeicherverwaltung mit Seitenwechsel (paging)
 - Adreßraum A, Speicherraum Υ
 - Hintergrundspeicher: virtueller Hauptspeicher mit virtueller Adresse v(P)
 - Seite (page): $\sigma: \{1, \dots, v-1\}$
 - Seitenrahmen (page frame): $\rho: \{0, \dots, M-1\}$
 - Zuweisung: benötigt Prozessor Speicherelement, das nicht im Hauptspeicher ist: Seitenwechsel von Hintergrundspeicher in Hauptspeicher

$$F(\sigma) = \begin{cases} \rho, & \text{falls Seite } \sigma \text{ ist im Seitenrahmen } \rho \\ n, & \text{falls Seite ist nicht im Hauptspeicher} \end{cases}$$
 - Synchronisation
 - gegenseitige Isolation von Prozessen
 - kritische Bereiche von Prozessen beachten!
 - Prozesse heißen voneinander abhängig, wenn Sie um das selbe Betriebsmittel konkurrieren
 - deshalb: Synchronisation notwendig
 - Varianten der Synchronisation
 - Prozeßkooperation: "Prozesse sprechen sich ab"
 - Wechselseitiger Ausschluß (mutual exclusion) durch Semaphor-Konzept
 - Prinzip der Verkehrsampel (rot/grün)
 - Gefahr bei wechselseitigem Ausschluß \Rightarrow Deadlock
 - Das Philosophen-Problem: n Philosophen, n Gabeln, P braucht 2 Gabeln zum Essen; Vorgehen, damit jeder hungrige Philosoph essen darf
 \Rightarrow deadlock, starvation, fairness

7. Dateiverwaltung

- Satz: Zusammenfassung logisch zusammenhängender Daten bestehend aus Feldern
 - Elementarfeld: ohne weitere Semantische Untergliederung, vom Typ Integer, Real, ...
 - strukturiertes Feld: RECORD, ARRAY mit fester Anzahl Komponenten
 - Wiederholungsgruppe: ARRAY[1...*] OF
- Satztyp: formaler Aufbau eines Satzes
- Datei (file): Zusammenfassung von Sätzen
- Schlüssel: identifizierend und minimal (\Rightarrow [DBIS I])
- logische Ebene: Sicht des Anwenderprogramms, Position durch PS bestimmt
- physische Ebene: abgespeicherte Datei aus Verwaltungs- und Datenteil
- Satzlänge: fest, variabel, undefiniert
- Blöcke: innerhalb eines Befehls nur Sätze einer Datei \wedge pro Datei: Blockgröße fest und vorgeschrieben
- Hintergrundspeicher: Gesamtspeicherraum, besteht aus N Blöcken, Umsetzung in physische Adresse
- Dateispeicherraum: Abspeicherung der Blöcke
 - zusammenhängend
 - logische Blocknummer = relative Blocknummer
 - logische Blocknummer \neq relative Blocknummer
 - verstreut
- Blockungsfaktor $BF(D) = \text{TRUNC}(\text{Blocklänge} / \text{Satzlänge}) (\geq 1)$
- relative Satznummer (RSN)

- **Dateiorganisation** | Hinweis: [S]: sequentielles Lesen
- **Primärorganisation (PO)** | [D]: direktes Lesen
 - Anordnung bei der physischen Speicherung
 - ⇒ erfordert effiziente Algorithmen fürs Abspeichern und Wiederauffinden
 - **sequentielle PO**
 - **unsortiert (ohne Primärschlüssel)**
 - Suchschlüssel ist Sekundärschlüssel
 - [D]: Sequentielles Suchen evtl. mit Zugriffspfad
 - [S]: Satz entfernen schlecht, wahlfreier Zugriff unmöglich
 - **sortiert (mit Primärschlüssel)**
 - [S]: Einfügen und Löschen aufwendig
 - [D]: Auffinden effizient mit BinarySearch
 - [D]: Löschen eher aufwendig, Reorganisation beizeiten notwendig
 - **gestreute PO (nur [D])**
 - Möglichkeiten der Zuordnung
 - **Hash-Organisation**
 - Anforderungen an Hash Funktion h:
h surjektiv, $h^{-1}(a)$ etwa gleichmäßig, h effizient berechenbar
 - M: Blockgröße
 - Divisionsmethode
 $h(s) = s \bmod M$, mit M möglichst Primzahl
 - multiplikative Methode
 $h(s) = \text{TRUNC} (M (s r - \text{TRUNC} (s r)))$ mit $r \in (0,1)$
goldener Schnitt: $r = 0,618\dots$
 - Mittelquadrat Methode
mittlere Ziffern von s^2 wählen (beliebig?)
 - falls $N > \sqrt{1/2 \pi M} \Rightarrow$ Kollisionswahrscheinlichkeit $> 50\%$
deshalb: Auflösung von Adresskollisionen durch
 - lineares Austesten: +1 solange bis frei
 - Speicherung in einem separatem Überlaufbereich (auf Blockebene)
 - (Anwendung einer 2. Hash-Funktion)
 - Nachteile: schlechte Speicherplatzausnutzung und schlechtes sequentielles Lesen und Schreiben
 - **Index-sequentielle Organisation**
 - Index := Inhaltsverzeichnis, enthält Sätze der Form: (s, adr(s))
 - Anlegen eines u.U. mehrstufigen Index
 - Vorteile: Suche nur in Hauptspeicher nötig
 - Nachteile: Speicherplatzausnutzung schlecht
 - Überlaufbehandlung mit Überlaufblöcken
neuen Block mit Pointer vom Ende des vorherigen erzeugen
(Index wird nicht verändert)
 - Überlaufbehandlung mit Blockteilung
neue und alte Elemente möglichst gleichmäßig auf 2 logisch aufeinanderfolgende Blöcke verteilen
(Blöcke werden zu Beginn nicht gefüllt)
 - beizeiten Reorganisation !
 - **Sekundärorganisation** (im WS 96/97 nicht behandelt!)
beeinflusst nicht physische Anordnung der Sätze, bietet jedoch eine andere Sicht auf Anordnung
⇒ häufig wünschenswert: Mehrfachzugriff