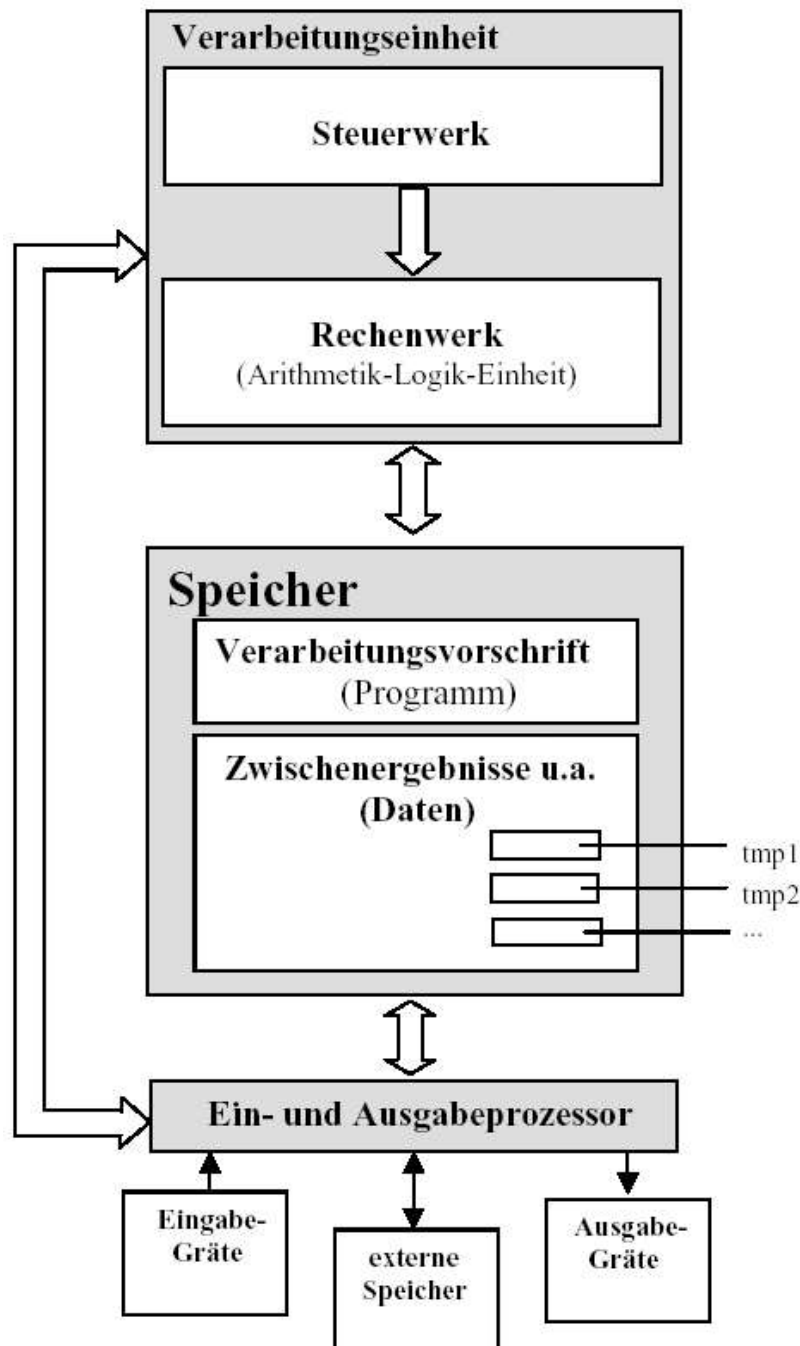


Von-Neumann-Architektur

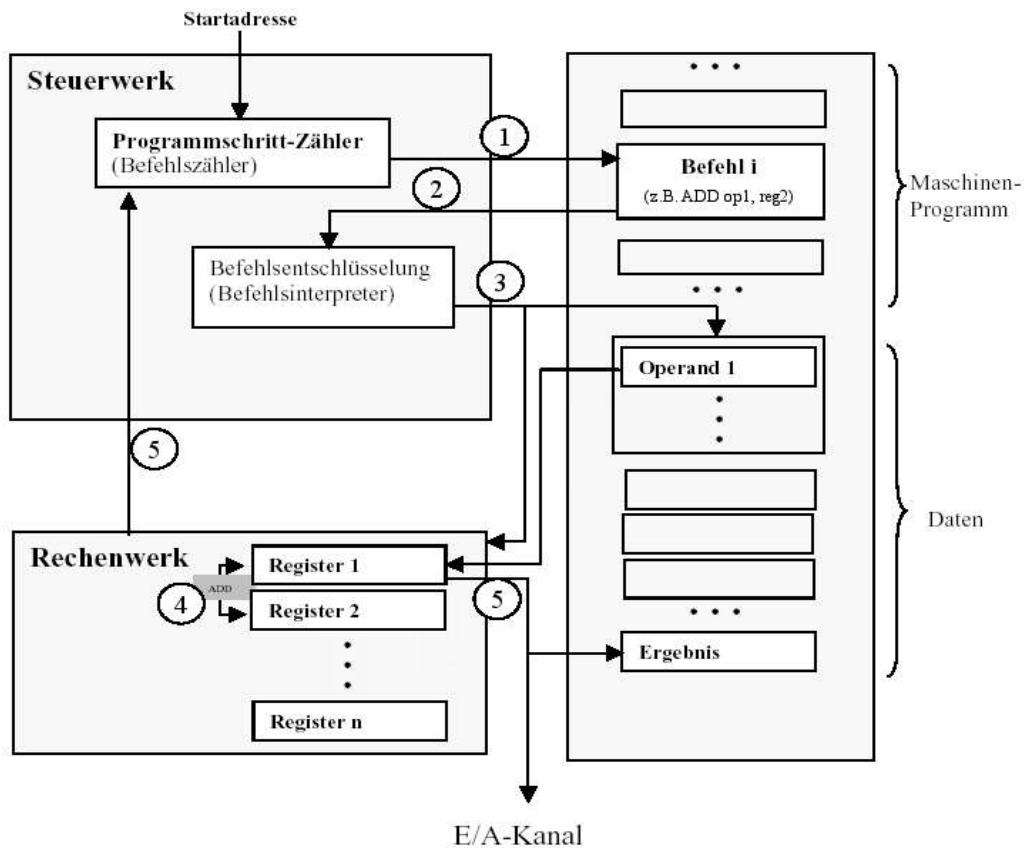
- benannt nach John von Neumann
- Schaltungskonzept zur Realisierung universeller Computer
- gemeinsamer Speicher für Programmcode und Daten
- Konzept beruht auf SISD-Prinzip (Single Instruction, Single Data) = serielle Befehlsbearbeitung

Blockdiagramm des Von-Neumann-Rechners



Von-Neumann-Zyklus

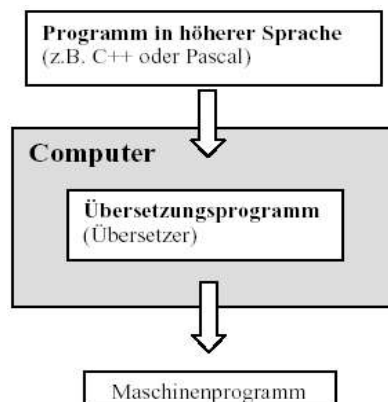
- Prozess der Befehlsverarbeitung:



- 5 Teilschritte:

1. **FETCH-Phase:** Nächsten auszuführenden Befehl holen (Befehlszähler) und in Befehlsregister schreiben.
2. **DECODE-Phase:** Übersetzen der Befehle in Schaltkreisinstruktionen für Rechenwerk.
3. **FETCH OPERANDS-Phase:** Speicheroperanden, die verarbeitet/geändert werden, holen.
4. **EXECUTE-Phase:** Operation vom Rechenwerk ausführen.
5. **UPDATE-Phase:** Ergebnis in Speicher schreiben und Befehlszähler inkrementieren

- Maschinenprogramm durch Compiler (Übersetzer) erzeugt:



Zahlensysteme

- Computer verarbeitet digitale Daten, also interne Repräsentation binär
- bekannte Zahlensysteme: Dezimal- (Basis 10), Dual- (Basis 2), Oktal- (Basis 8), Hexadezimal- (Basis 16) System
- jede Zahl in Polynomschreibweise darstellbar -> für jede Zahl Basis angebar: $235_{10}, 01101_2, A3B_{16}$

Dezimalsystem

- Basis: 10, Ziffern $z_k : 0 \dots 9$ ($k \in \mathbb{Z}$)
- Polynomschreibweise: $z_i * 10^i + z_{i-1} * 10^{i-1} + \dots + z_1 * 10^1 + z_0 * 10^0 + z_{-1} * 10^{-1} + \dots + z_{-j} * 10^{-j}$
- Beispiel: $235,5_{10} = 2 * 10^2 + 3 * 10^1 + 5 * 10^0 + 5 * 10^{-1} = 2 * 100 + 3 * 10 + 5 * 1 + 5 * 0,5$

Radixschreibweise

Dualsystem

- Basis 2, Ziffern $z_k : 0$ und 1 ($k \in \mathbb{Z}$) => Bits, 8 Bit = 1 Byte
- Polynomschreibweise: $z_i * 2^i + z_{i-1} * 2^{i-1} + \dots + z_1 * 2^1 + z_0 * 2^0 + z_{-1} * 2^{-1} + \dots + z_{-j} * 2^{-j}$
- Beispiel: $1010,1_2 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} = 1 * 8 + 0 * 4 + 1 * 2 + 0 * 1 + 1 * 0,5 = 10,5_{10}$
- Anwendung: Digitaltechnik, da leicht elektronisch realisierbar (Low-, Highpegel einer Spannung)

Oktalsystem

- Basis 8, Ziffern $z_k : 0 \dots 7$ ($k \in \mathbb{Z}$)
- Polynomschreibweise: $z_i * 8^i + z_{i-1} * 8^{i-1} + \dots + z_1 * 8^1 + z_0 * 8^0 + z_{-1} * 8^{-1} + \dots + z_{-j} * 8^{-j}$
- Beispiel: $235,5_8 = 2 * 8^2 + 3 * 8^1 + 5 * 8^0 + 5 * 8^{-1} = 2 * 64 + 3 * 8 + 5 * 1 + 5 * (1/8) = 157,625_{10}$
- Anwendung: gut geeignet zur Darstellung von Bitfolgen der Länge 3 (siehe unten) -> $8 = 2^3$

Hexadezimalsystem

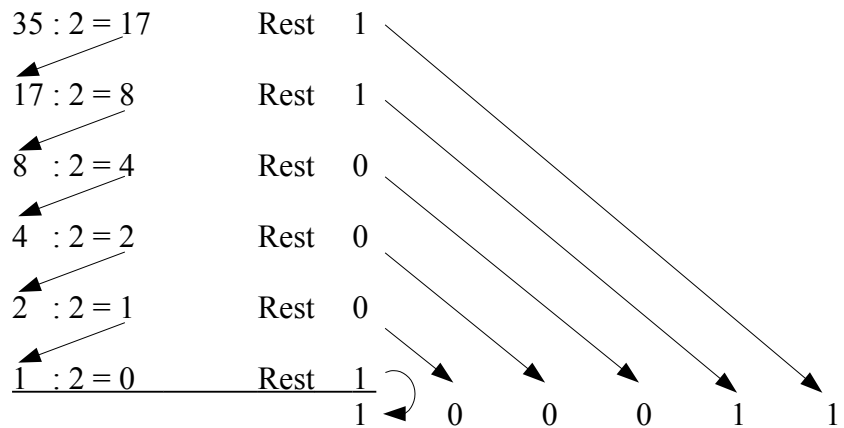
mehr Informationen pro Zeichen

- Basis 16, Ziffern $z_k : 0 \dots 9, A \dots F$ ($k \in \mathbb{Z}$)
- Polynomschreibweise: $z_i * 16^i + z_{i-1} * 16^{i-1} + \dots + z_1 * 16^1 + z_0 * 16^0 + z_{-1} * 16^{-1} + \dots + z_{-j} * 16^{-j}$
- Beispiel: $A3B0_{16} = 10 * 16^3 + 3 * 16^2 + 11 * 16^1 + 0 * 16^0 = 40960 + 768 + 176 + 0 = 41904_{10}$
- Anwendung: gut geeignet zur Darstellung von Bitfolgen der Länge 4 (siehe unten) -> $16 = 2^4$

Konvertieren von Zahlensystemen

Dezimalsystem => beliebiges Zahlensystem

- Algorithmus: teile solange mit Rest durch die Basis des Zielsystems, bis die Division 0 ergibt
- Beispiel: Dezimalsystem (35_{10}) => Dualsystem



Oktalsystem => Dualsystem

- die Basis des Oktalsystems ist eine 3-er Potenz der Basis des Dualsystems
=> deshalb kann man 3 Bits zusammenfassen zu einer Oktal-Ziffer
- Beispiel: Dualsystem (110101100111_2) => Oktalsystem

$$\begin{array}{cccc} 110 & 101 & 100 & 111 \\ \hline 6 & 5 & 4 & 7 \end{array} \Rightarrow 6547_8$$

- Rücktransformation geschieht einzeln für jede Oktalziffer wie Dezimalsystem => Dualsystem

Hexadezimalsystem => Dualsystem

- die Basis des Hexadezimalsystems ist eine 4-er Potenz der Basis des Dualsystems
=> deshalb kann man 4 Bits zusammenfassen zu einer Hexadezimal-Ziffer
- Beispiel: Dualsystem (110101100111_2) => Hexadezimalsystem

$$\begin{array}{ccc} 1101 & 0110 & 0111 \\ \hline D & 6 & 7 \end{array} \Rightarrow D67_{16}$$

- Rücktransformation geschieht einzeln für jede Hexadezimalziffer (ggf. Umwandlung in Zahlen 10 bis 15) wie Dezimalsystem => Dualsystem

Addition und Subtraktion im Dualsystem

Verknüpfungsregeln

Addition:

$$\begin{array}{l} 0 + 0 = 0 \\ 0 + 1 = 1 \\ 1 + 0 = 1 \\ 1 + 1 = 0 \text{ mit Übertrag 1} \end{array}$$

Subtraktion:

$$\begin{array}{l} 0 - 0 = 0 \\ 1 - 0 = 1 \\ 1 - 1 = 0 \\ 0 - 1 = 1 \text{ mit Mangelübertrag 1} \end{array}$$

- Beispiel:

$$\begin{array}{r} 10111,1 \quad 23,5_{10} \\ + 1011,0 \quad + 11,0_{10} \\ \hline 11111 \\ \hline 100010,1 \quad 34,5_{10} \end{array}$$

$$\begin{array}{r} 11001 \quad 25_{10} \\ - 1011 \quad - 11_{10} \\ \hline 111 \\ \hline 01110 \quad 14_{10} \end{array}$$

Darstellung negativer Zahlen

1. Realisiert mit Vorzeichenbit:

- Definition des höchstwertigen Bits als Vorzeichenbit
- daraus resultiert Festlegung auf konkrete Stellenzahl und ggf. notwendigem Auffüllen

$$\begin{array}{l} + 7_{10} = 00000111_2 \\ - 7_{10} = 10000111_2 \end{array}$$

$$00000111_2 + 10000111_2 = 10001110_2 (142_{10}) \\ \text{(wäre so nicht korrekt)}$$

- Nachteil: Addition einer pos. und einer neg. Zahl, muss als Subtraktion der neg. von der pos. Zahl erfolgen (mehr Rechenaufwand für Unterscheidung) => Bsp.: 00000111 + 10000111

2. Realisiert mit Einer-Komplement:

- Negieren aller Bits einer Dualzahl, so dass Addition neg. Zahlen einfach möglich wird
- auch hier Festlegung auf konkrete Stellenzahl und ggf. notwendigem Auffüllen

$$\begin{array}{l} + 7_{10} = 00000111_2 \\ - 7_{10} = 11111000_2 \end{array}$$

$$00000111_2 + 11111000_2 = 11111111_2 \\ \text{(dies ist korrekt, aber nicht perfekt)}$$

- Nachteil: die 0 hat zwei Werte => 00000000 und 11111111

3. Realisiert mit Zweier-Komplement:

- wie Einer-Komplement, jedoch mit zusätzlicher Addition von 1, um doppelte 0 zu vermeiden

$$\begin{array}{l} + 7_{10} = 00000111_2 \\ - 7_{10} = 11111001_2 \end{array}$$

$$00000111_2 + 11111001_2 = 00000000_2 \\ \text{(dies ist korrekt -> Überlauf wegwerfen)}$$

Betriebssystemkonzept am Beispiel von Unix

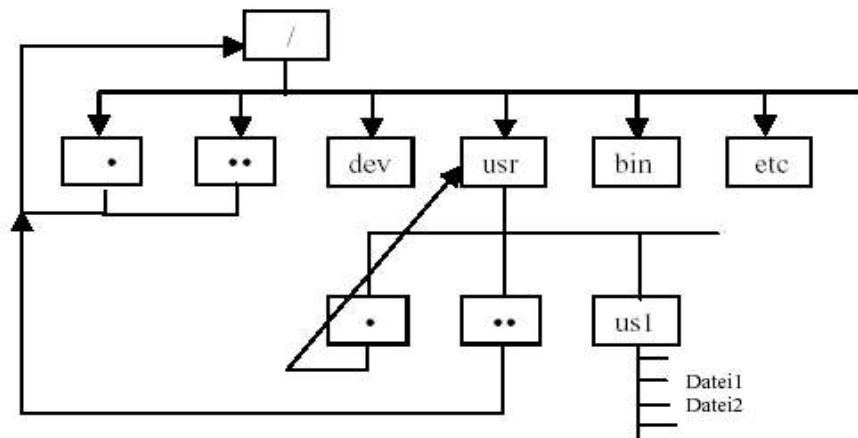
- Stark vereinfachte Modellvorstellung:



- Systemkern: enthält elementare Routinen zur Verwaltung von Ein-/Ausgabe, Prozessen etc.
- alles andere läuft auf Ebene von Dienstprogrammen, die sich der Kern-Funktionen bedienen
- Betriebssystem genauer bestimmt durch:
 - Dateisystemverwaltung
 - Prozess-Steuerung (Ablaufsteuerung, Ressourcenzuteilung, Ein-/Ausgabe, ...)
 - Shell-Konzept (Art der Interaktion mit Betriebssystem)

Dateisystemverwaltung unter Unix

- strukturiertes Dateisystem mit hierarchischer Struktur
- Zugriffsrechte auf Dateien, Verzeichnisse, Geräte etc. definiert
- Adressierung erfolgt über Pfadangaben



`.. /` - Wurzelverzeichnis (root)

`.. .` - Verweis auf den Namen des eigenen Verzeichnisses

`.. ..` - Verweis zu Verzeichnis der höheren Ebene (Vaterdirectory)