Einführung in die Informatik

Moderne Rechnerarchitektur

Meik Teßmer

Inhalte

Lernziele

- Erkenntnisse über Aufbau eines modernen Computers
- etablierte Rechnerarchitektur(en)

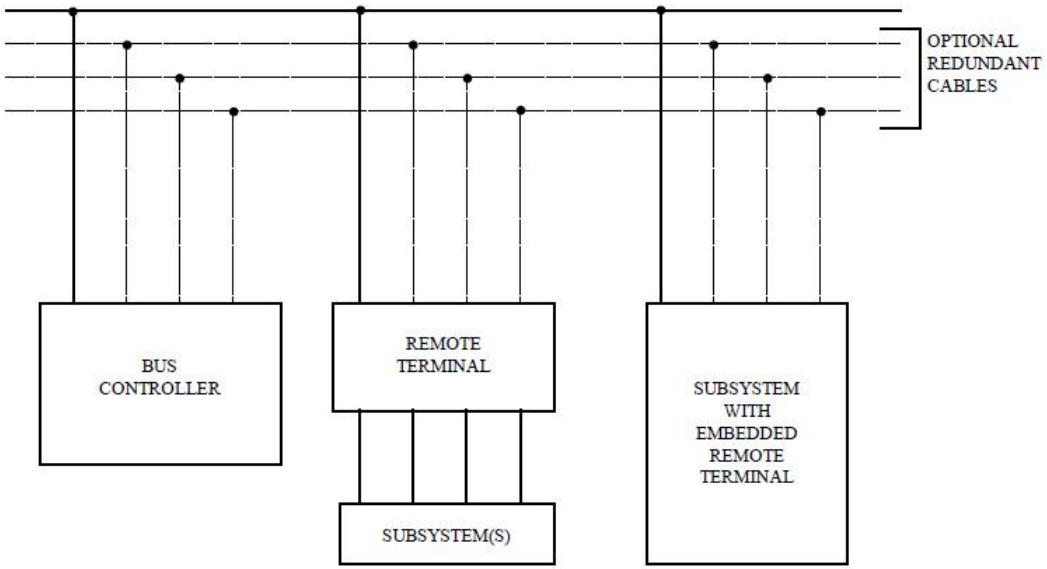
Aktueller Stand unseres Computer-Entwurfs

- Was haben wir? Grundbausteine:
  - logische Bausteine: And, Or, Xor
  - *große* Bausteine: Halb-/Volladdierer, Addierwerk
  - Kommunikation zwischen Bausteinen: De-/Multiplexer (Selektor)
- Was fehlt noch?
  - Bussysteme: Verbindung von Bausteinen
  - CPU (*Prozessor*)
  - Speicher
  - Peripherie
  - Startskript

Bausteine

Bussysteme

- Aufgabe: Verbindung aller *größeren* Bausteine (CPU, Speicher, Peripherie)
- BUS steht für *Binary Unit System*
- ursprünglich parallele Leitungen mit mehreren Anschlüssen, durch Multiplexer auch seriell möglich
- Idee: mehrere *Knoten* teilen sich Leitungen



Bussysteme

- Bussysteme können parallel oder seriell sein
- aktiver Teilnehmer: darf senden
- passiver Teilnehmer: empfängt/liest
- Adressierung des Empfängers: separater Adressbus

Bussysteme

- CPU-interner Bus: Verbindung der CPU-Bausteine (+Arbeitsspeicher)
- CPU-externer Bus (*Northbridge*): Anbindung weiterer Prozessoren, Arbeitspeicher, Peripherie-Bus (*Southbridge*)
- Southbridge: Anbindung langsamerer Geräte (bspw. USB)
- Vorteil: Entkoppelung der CPU (kann schneller arbeiten)

Bussysteme

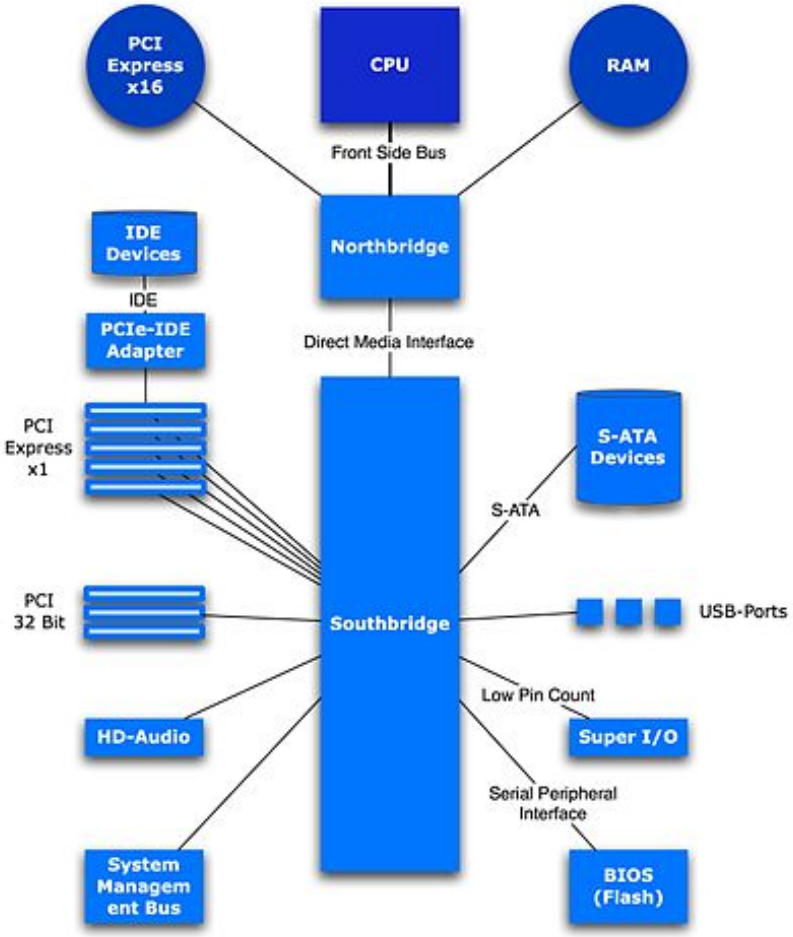


Abbildung 2: Schematischer Aufbau eines Rechners.

Bussysteme

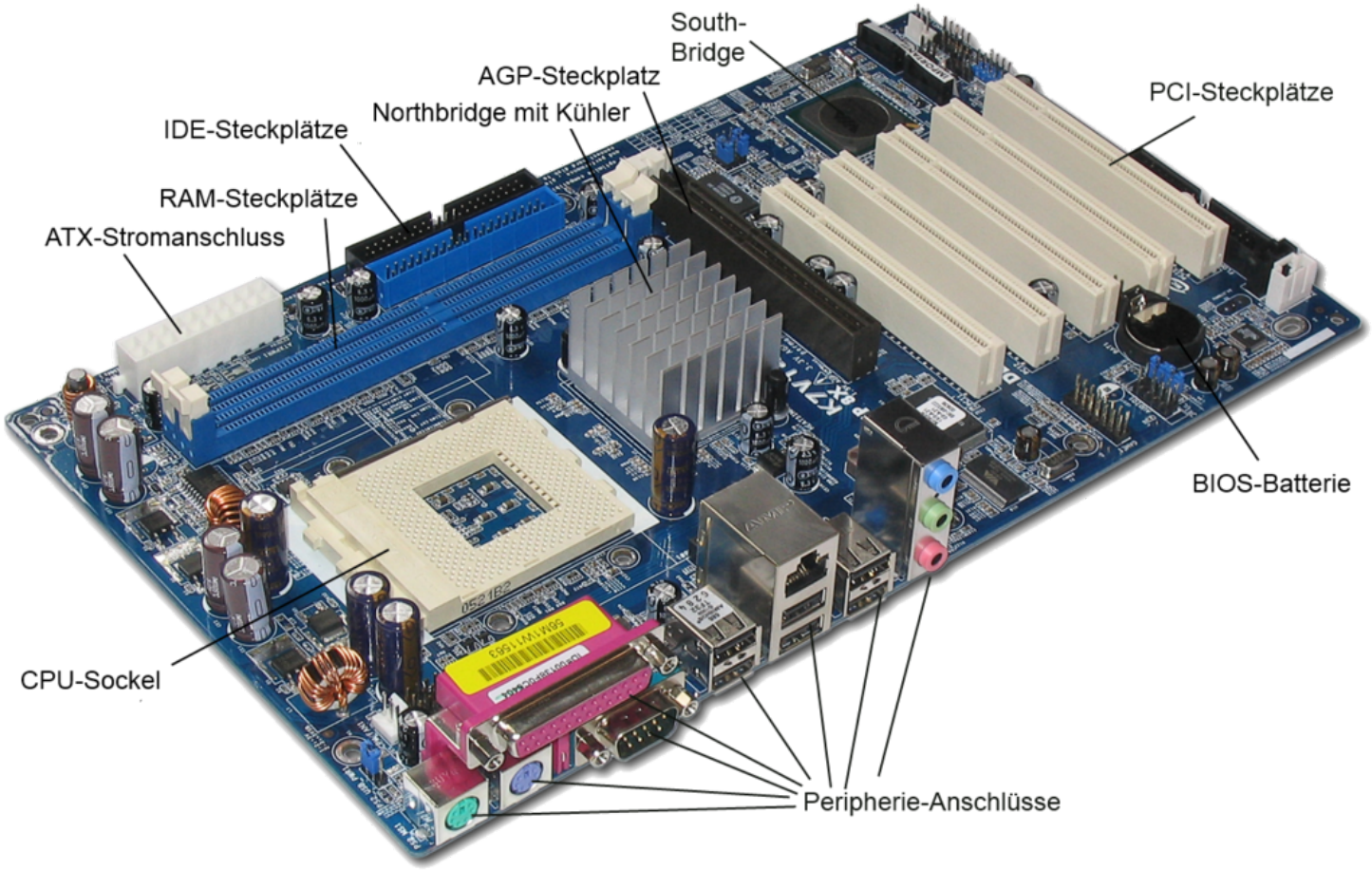


Abbildung 3: Motherboard (Hauptplatine).

CPU

Central Processing Unit (CPU)

zentraler Baustein der CPU: die *Arithmetic Logical Unit* (ALU)

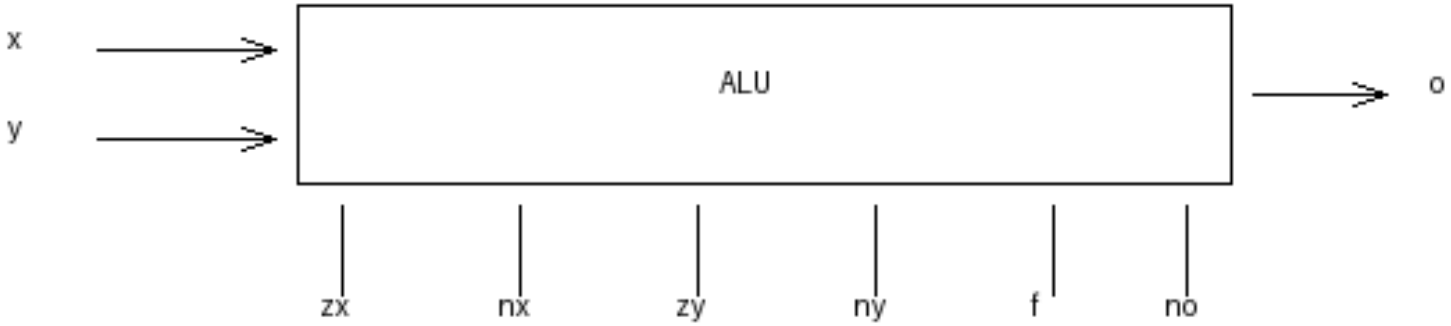
- Aufgaben:
  - Berechnungen ausführen

    Grundrechenarten haben wir schon evaluiert
  - bestimmte logische Operationen ausführen

    Beispiel: Invertieren
- Aufbau:
  - Eingänge: Werte x und y, 6 Bits für die Steuerung
  - Ausgang: Wert out

ALU

Schema-Zeichnung:



ALU

Control Bits: `zx`, `nx`, `zy`, `ny`, `f`, `no`

- für `x`:
  - `zx=1`: `x` komplett 0 setzen
  - `nx=1`: `x` negieren
- für `y`: dasselbe (`zy` und `ny`)
- `f`:
  - `f=0`: And(x, y)
  - `f=1`: $x + y$
- `no=1`: Ergebnis (`o`) invertieren

ALU: Beispiel $x - 1$

- x wird nicht verändert $\rightarrow zx = nx = 0$
- Subtraktion: entspricht Addition mit -1

  $\rightarrow$ Zweierkomplement von 1 bilden und addieren
- y auf 0 setzen: $zy = 1$, dann invertieren: $ny = 1$

  entspricht dem Zweierkomplement von -1
- mit x addieren $\rightarrow f = 1$
- Ergebnis bleibt unverändert $\rightarrow no = 0$

Control Bits:

zx	nx	zy	ny	f	no
0	0	1	1	1	0

ALU: Beispiel x+1

Idee: x invertieren und größtmögliche Zahl addieren:

- `x=1010` $\rightarrow 1010 + 1 = 1011$
- invertieren: `0101`
- größtmögliche Zahl: `1111`
- Addition: `(1)0100`
- Ergebnis invertieren: `1011`

ALU: Beispiel x+1

Control Bits:

- x wird nicht auf 0 gesetzt $\rightarrow zx = 0$
- x wird invertiert $\rightarrow nx = 1$
- y auf 0 und invertieren $\rightarrow zy = ny = 1$
- Addition $\rightarrow f = 1$
- Ergebnis invertieren: $no = 1$

Zusammenfassung ALU

alle wichtigen Operationen können mit einer ALU durchgeführt werden

→ wir haben eine Art Taschenrechner für einfache Rechenvorgänge

Problem: Was ist mit den Ein-/Ausgängen, wenn gerade nicht damit gerechnet wird?

Was fehlt?

Speicher

Zustände festhalten

- ALU: besteht aus kombinatorischen Schaltungen

  kann keine Zustände speichern
- Daten: müssen für eine bestimmte Zeit gespeichert/abgerufen werden können
- Speicherbausteine heißen auch *Sequential Chips*
- Idee: Bausteine verarbeiten Eingänge *im Takt*
- Clock: Taktgeber für das System (0, 1, 0, 1...)
- Cycle: Zeit zwischen Takten
- Multiplier: erhöhen Taktrate für Speicher und CPU

Speicherbausteine

- Grundbaustein: Flip-Flop (auch Digital Flip-Flop)
- Konzept: out(t) = in(t-1)
- Aufbau:
  - Eingänge: `in`, `clock`
  - Ausgang: `out`
- Entwurf?

Baustein:

Abbildung 4: Digital Flip-Flop.

Entwurf 1

Idee: Ausgang auf Eingang legen

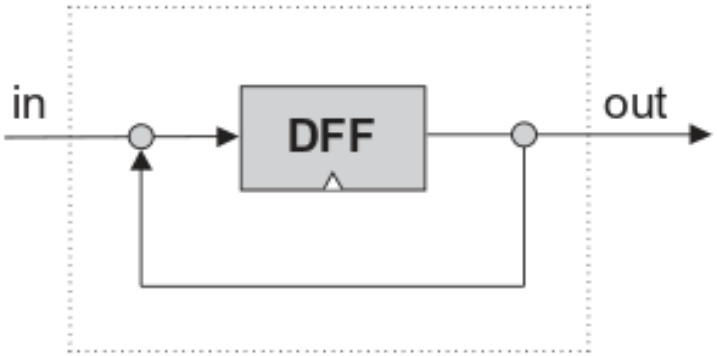


Abbildung 5: Versuch 1

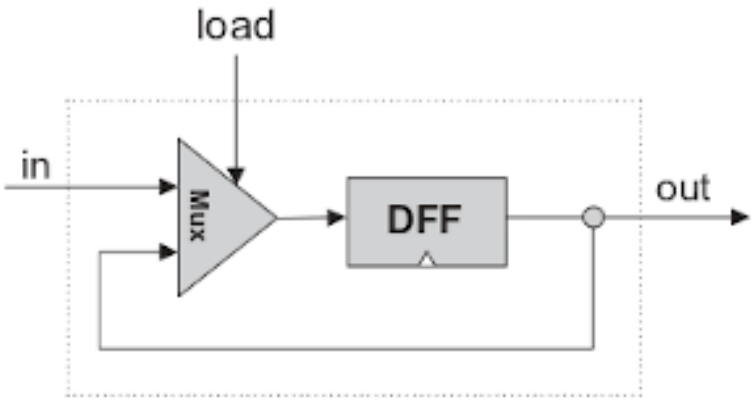
so kann ein Wert über mehrere Taktzyklen gespeichert werden

Problem?

Entwurf 2

Verbesserung: Multiplexer (Selektor) verwenden

- ist das Selektor-Bit (load) 1, dann wird der Wert von in gespeichert
- ist load 0, *überschreibt* sich das DFF mit seinem aktuellen Wert



if load(t–1) then out(t) = in(t–1)
else out(t) = out(t–1)

Abbildung 6: Versuch 2

Register

- nennt man *1-Bit-Register*
- mehrere 1-Bit-Register lassen sich kombinieren, bspw. zu 8, 16- oder 32-Bit-Registern

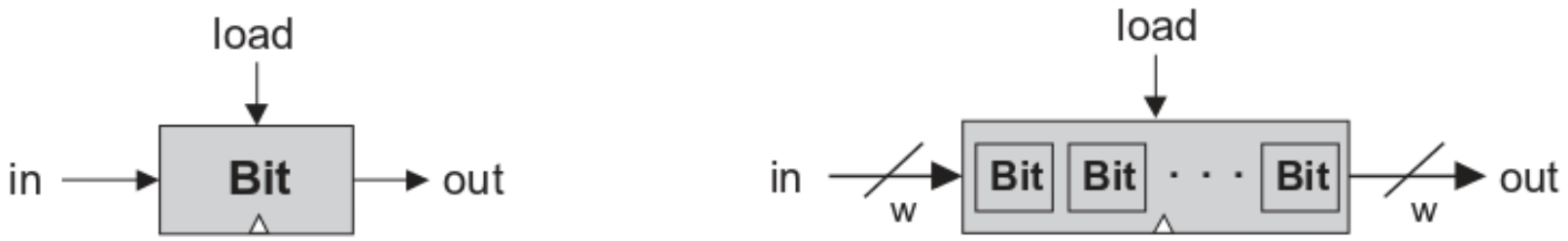


Abbildung 7: 1-Bit-Register

Speicherbaustein

- besteht aus mehreren Registern (*Stapel*)
- Ein- und Ausgang: Bitbreite entspricht der Registergröße

Was fehlt noch?

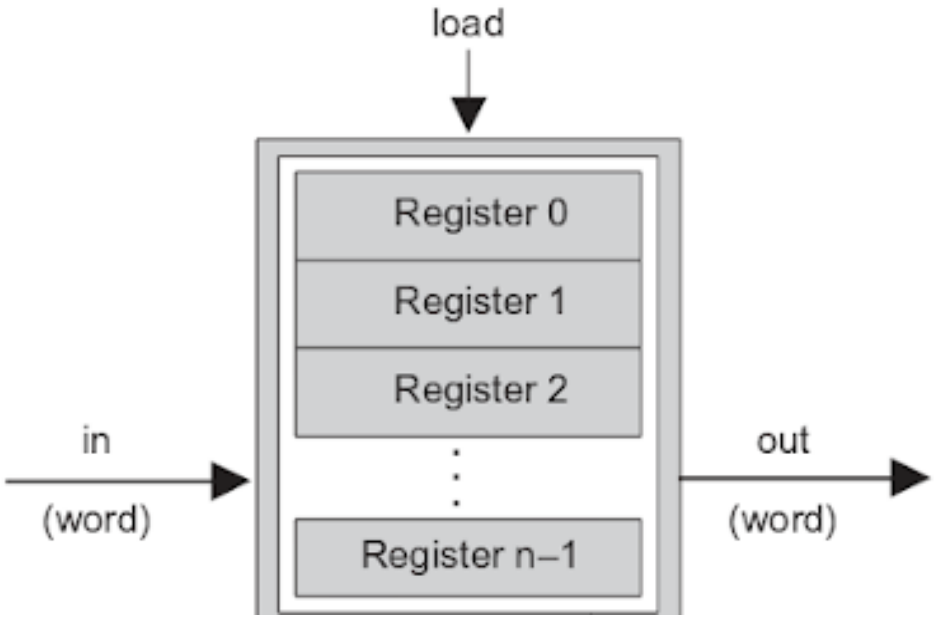


Abbildung 8: Speicher

Speicherbaustein

- benötigt: Zugriffslogik

  $\rightarrow$ *Adressierung* eines Registers

- Lösung:
  - Register bekommen eindeutige Bezeichner (0, 1, 2...)
  - Adressierungsbaustein: angelegte *Adresse* (=Bezeichner) wählt Register aus

- Registerstapel +Adressierungsbaustein = *Random Access Memory* (RAM)

RAM

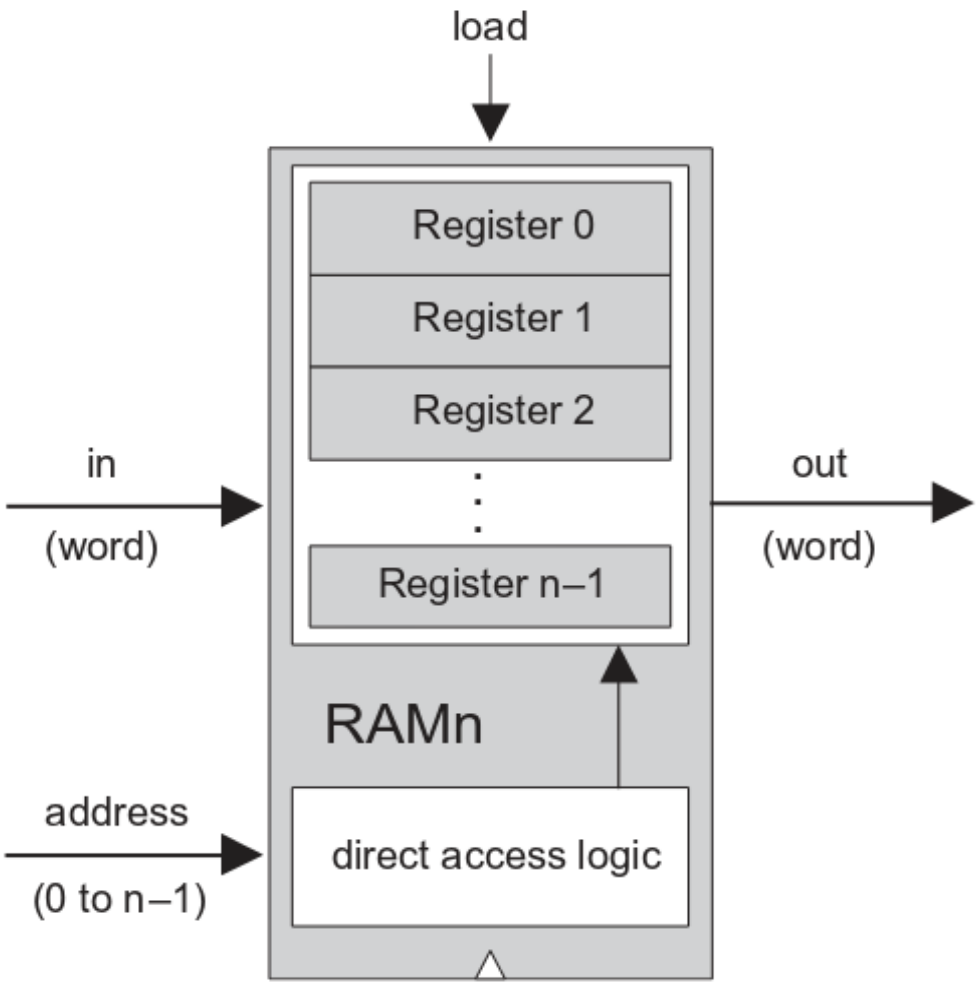


Abbildung 9: RAM-Baustein

Ergebnis

Was haben wir erreicht?

- Bussysteme
- ALU: kann rechnen bzw. logische Operationen auf Werten ausführen
- RAM: kann Werte speichern

Was fehlt noch?

- Anbindung Peripherie
- Gesamtkonstruktion

Ziel: Konstruktion eines universellen Computers

Computer

Gesamtkonstruktion

Grundidee: *stored program*-Konzept (von-Neumann-Architektur)

- 1930 von verschiedenen Mathematikern formuliert
- benannt nach John von Neumann
- Ansatz: Daten und das sie verarbeitende Programm befinden sich im *selben* Speicher
  - universeller Rechner
  - einfache und schnelle Programmänderungen

    $\rightarrow$ dieselbe Hardware verhält sich je nach Programm unterschiedlich

von-Neumann-Architektur

- CPU: ALU, Register und Steuerlogik (Verbindung: CPU-interner Bus)
- Speicher für Programm und Daten (Anbindung: Northbridge)
- Peripherie für Ein- und Ausgaben (Anbindung: Southbridge)

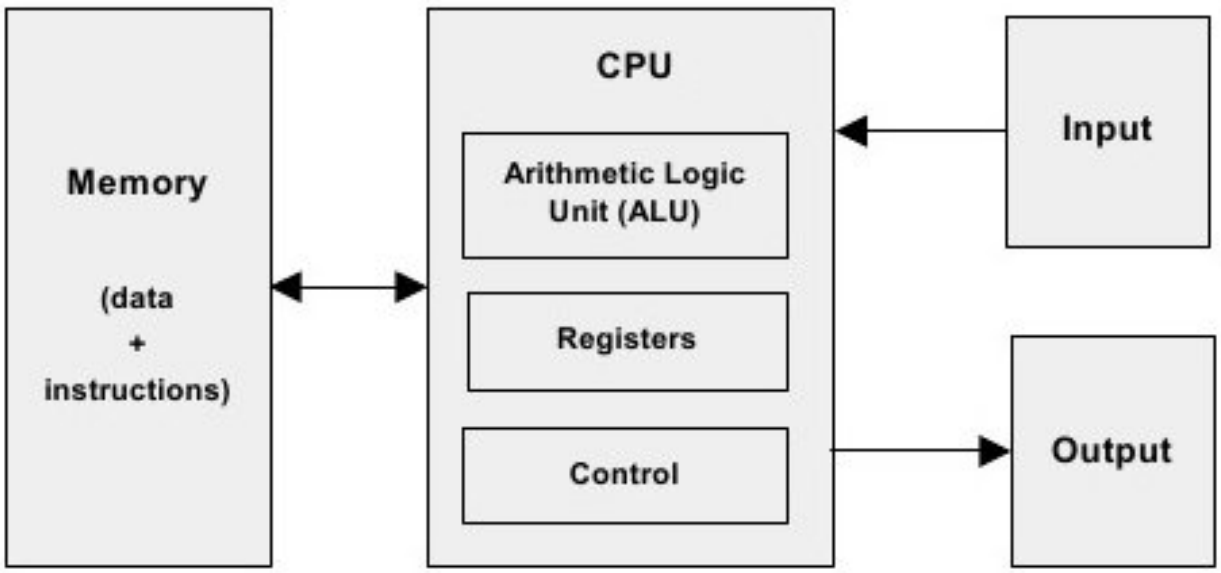


Abbildung 10: von Neumann-Architektur

Peripherie

- umfasst Bildschirme, Tastaturen, Scanner, Drucker, Netzwerkkarten, CD-/DVD-Laufwerke usw.
- einfache Form der Anbindung: *memory-mapped I/O*
- Idee: Gerät als Speicherbereich abbilden

  $\rightarrow$ *memory map* des Geräts
- Eingabe: memory map spiegelt Zustand des Eingabegeräts
- Ausgabe: memory map steuert den Zustand des Ausgabegeräts

Zusammenfassung

- Bussysteme + CPU + RAM + Peripherie = moderne Rechnerarchitektur
- stored program-Konzept: ermöglicht Konstruktion universeller Rechner
- memory-mapped I/O: Standardanbindung für Peripherie

Logik → Hardware

Unser Computer ist damit fertig!

Ende